Does Query Evaluation Tractability Help Query Containment?

Pablo Barceló
Department of Computer Science, Universidad de Chile
pbarcelo@dcc.uchile.cl

Miguel Romero
Department of Computer Science, Universidad de Chile
mromero@dcc.uchile.cl

Moshe Y. Vardi
Department of Computer Science, Rice University
vardi@cs.rice.edu

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: 2EXPTIME-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 Θ' in a UCQ Θ is NP-hard, in general, but better bounds can be obtained if Θ is restricted to belong to a *tractable* class of UCQs, e.g., a class of bounded treewidth or hypertreewidth. Also, each Datalog program Π is equivalent to an infinite union of CQs. This motivated us to study the question of whether restricting Θ to belong to a tractable class also helps alleviate the complexity of checking whether Π is contained in Θ.

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 Θ 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 Π in Θ is in EXPTIME if Θ is of treewidth one, or it is acyclic and the arity of the schema is fixed. In the case of UC2RPQs we show an 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 Π is contained in UC2RPQ Γ is in EXPTIME if Γ 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 2EXPTIME-complete.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Query Languages*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD/PODS'14, June 22–27, 2014, Salt Lake City, UT, USA.
Copyright ACM 978-1-4503-2375-8/14/06?$15.00.
http://dx.doi.org/10.1145/2594538.2594553.

Keywords

Containment; conjunctive queries; datalog; conjunctive regular path queries; tree automata.

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 Θ is contained in a Datalog program Π is EXPTIME-complete [16], while the opposite, deciding whether the Datalog program Π is contained in the UCQ Θ, is 2EXPTIME-complete [12]. From this we obtain that the important problem of checking whether the recursive program Π is equivalent to the nonrecursive (and, thus, typically easier to evaluate and optimize) query Θ can be solved in 2EXPTIME.

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 2EXPTIME 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 θ and θ'. This problem is NP-complete, but it follows from [10] that it can be solved efficiently if θ' 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 by 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

   $$\mathsf{TW}(1) \subset \mathsf{TW}(2) \subset \cdots \subset \mathsf{TW}(k) \subset \cdots,$$

   where $\mathsf{TW}(k)$ 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

   $$\mathsf{HW}(1) \subset \mathsf{HW}(2) \subset \cdots \subset \mathsf{HW}(k) \subset \cdots,$$

   where $\mathsf{HW}(k)$ corresponds to the class of CQs of *hypertreewidth* at most k, is tractable. Interestingly, the first level of this hierarchy, $\mathsf{HW}(1)$, 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 Θ is in $\mathsf{TW}(k)$ (resp. $\mathsf{HW}(k)$) iff each one of the CQs in Θ is in $\mathsf{TW}(k)$ (resp. $\mathsf{HW}(k)$). Tractability results for query evaluation and containment extend to these classes [31].

Returning to the problem of checking containment of a Datalog program Π in a UCQ Θ, it is well-known that each Datalog program Π 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 Θ to belong to one of the tractable classes $\mathsf{TW}(k)$ and $\mathsf{HW}(k)$ 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 Π) in Θ 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 $\mathsf{TW}(1)$ – helps to reduce the complexity of the Datalog containment problem. That is, checking whether $\Pi \subseteq \Theta$, for Π a Datalog program and Θ a UCQ, continues to be 2EXPTIME-complete even if (a) Θ is of $\mathsf{TW}(2)$, or (b) Θ is in $\mathsf{HW}(1)$, 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

   $$\mathsf{AC}_1 \subset \mathsf{AC}_2 \subset \cdots \subset \mathsf{AC}_k \subset \cdots,$$

   where AC_k denotes the class of UCQs Θ in AC such that no two atoms in Θ share more than k variables. We prove that checking $\Pi \subseteq \Theta$, for Θ in AC_k, can be solved in EXPTIME (being complete for this class).

   As corollaries, we obtain that checking $\Pi \subseteq \Theta$ can be solved in EXPTIME when Θ 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 2EXPTIME-complete, even if Θ is a UCQ in $\mathsf{HW}(2)$ or $\mathsf{TW}(2)$ 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 2EXPTIME.

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 $\mathsf{AC} = \mathsf{TW}(1)$ over graph databases). Since containment of Datalog in UCQs in $\mathsf{TW}(1)$ can be solved in 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 Π in a UC2RPQ Γ is 2EXPTIME-complete, even if Γ 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

   $$\mathsf{ACR}^1 \subset \mathsf{ACR}^2 \subset \cdots \subset \mathsf{ACR}^k \subset \cdots,$$

   where ACR^k is the class of UC2RPQs $\Gamma \in \mathsf{ACR}$ in which each pair of variables is connected by at most k atoms, and then prove that $\Pi \subseteq \Gamma$ can be solved in EXPTIME if Γ 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 Γ of (hyper-)treewidth two is 2EXPTIME-complete, even if each pair of variables in Γ 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 σ and databases $\mathcal{D}$. Recall that a conjunctive query (CQ) over σ is a logical formula in the $\exists, \wedge$-fragment of FO, i.e., a formula θ of the form

$$\exists\bar{y}(R_1(\bar{x}_1) \wedge \cdots \wedge R_m(\bar{x}_m)),$$

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

As usual, the semantics of CQs is defined in terms of *homomorphisms*. Formally, a homomorphism h from a CQ θ of the form $\exists\bar{y}(R_1(\bar{x}_1) \wedge \cdots \wedge R_m(\bar{x}_m))$ to a database $\mathcal{D}$ over σ is a mapping from the set of variables that appear in the $\bar{x}_i$'s (for $1 \leq i \leq m$) to the elements of $\mathcal{D}$, such that $R_i(h(\bar{x}_i)) \in \mathcal{D}$ for each $1 \leq i \leq m$. The evaluation of θ over $\mathcal{D}$, denoted $\theta(\mathcal{D})$, consists of the set of all tuples $h(\bar{x})$, for h a homomorphism from θ to $\mathcal{D}$.

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

Datalog. Extending UCQs with recursion yields the Datalog language. Formally, a Datalog *program* Π 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 Π and by $\text{IRels}(\Pi)$ the set of *intensional* relation symbols of Π, 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 in Π.

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

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

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

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

$$\begin{aligned} buys(x,y) &\;\leftarrow\; likes(x,y) \\ buys(x,y) &\;\leftarrow\; trendy(x), buys(z,y). \end{aligned}$$

The program Π_c is over the schema that consists of symbols $trendy$ and $likes$. The distinguished symbol is $buys$. □

Containment of queries. Let q_1 and q_2 be queries over the same schema σ 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 $q_1(\mathcal{D}) \subseteq q_2(\mathcal{D})$ for every database $\mathcal{D}$ over σ. Notice that q_1 is equivalent to q_2 iff $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ 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 $\mathcal{C}_1$ in $\mathcal{C}_2$ as follows:

PROBLEM :	CONT($\mathcal{C}_1$,$\mathcal{C}_2$).
INPUT :	Queries $q_1 \in \mathcal{C}_1$ and $q_2 \in \mathcal{C}_2$ over same schema σ.
QUESTION :	Is $q_1 \subseteq q_2$?

We also study the complexity of this problem for schemas of fixed arity. Formally, let c be a positive integer. We then write CONT$_c$($\mathcal{C}_1$,$\mathcal{C}_2$) to denote the restriction of the problem CONT($\mathcal{C}_1$,$\mathcal{C}_2$) to inputs in which the arity of σ is at most c.

Containment of UCQs. It has long been known that containment and evaluation of CQs are polynomially equivalent problems. In fact, let $\theta(\bar{x})$ and $\theta'(\bar{x})$ be CQs over σ and assume that θ is of the form $\exists\bar{y}(R_1(\bar{x}_1), \ldots, R_m(\bar{x}_m))$. Let $\mathcal{D}_\theta$ be the *canonical database* of θ, i.e.,

$$\mathcal{D}_\theta \;:=\; \{R_i(\bar{x}_i) \mid 1 \leq i \leq m\}.$$

It follows from the seminal work of Chandra and Merlin [10] that $\theta \subseteq \theta'$ iff $\bar{x} \in \theta'(\mathcal{D}_\theta)$. Furthermore, Sagiv and Yannakakis proved that if Θ and Θ' 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] CONT(UCQ,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], CONT(Datalog,UCQ). In full generality, this is a computationally expensive problem, even for schemas of fixed arity.

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

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

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

3. TRACTABLE CONTAINMENT FOR UCQ

While in general UCQ containment is an NP-complete problem, we obtain tractability by imposing that UCQs Θ' 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 $\mathcal{D}$, and a tuple $\bar{t}$ of the same arity than $\bar{x}$, checks whether $\bar{t} \in \Theta(\mathcal{D})$. Formally:

PROPOSITION 1. [10, 31] *Let* $\mathcal{C}$ *be a tractable class of UCQs. Then* CONT(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 θ of the form $\exists \bar{y}(R_1(\bar{x}_1), \dots, R_m(\bar{x}_m))$ is the undirected graph G_θ whose nodes are the variables of θ, and there is an edge in G_θ from variable x to x' iff 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, λ), 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, λ) 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 Θ is the maximum treewidth of G_θ, for $\theta \in \Theta$. We denote by $\mathsf{TW}(k)$ the class of UCQs of treewidth at most k, for $k \geq 1$.

EXAMPLE 3. The CQ $E(x_1, x_2) \wedge \cdots \wedge E(x_{n-1}, x_n)$ is in $\mathsf{TW}(1)$, 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 $\mathsf{TW}(1) \subset \mathsf{TW}(2) \subset \cdots \subset \mathsf{TW}(k) \subset \cdots$ is tractable. From Proposition 1 we thus have:

THEOREM 3. *Let* $k \geq 1$. *Then* CONT(UCQ,$\mathsf{TW}(k)$) *can be solved in* 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 $\mathsf{HW}(1) \subset \mathsf{HW}(2) \subset \cdots \subset \mathsf{HW}(k) \subset \cdots$, where $\mathsf{HW}(k)$ is the class of UCQs of hypertreewidth at most k, is tractable [24]. We skip the definition of the $\mathsf{HW}(k)$'s since it is rather technical and not crucial for our purposes (the interested reader can find the definition in [24]).

Since the $\mathsf{HW}(k)$'s are tractable, we obtain from Proposition 1 the tractability of CONT(UCQ,$\mathsf{HW}(k)$):

THEOREM 4. *Let* $k \geq 1$. *Then* CONT(UCQ,$\mathsf{HW}(k)$) *can be solved in* PTIME.

It is important for us to notice that the lowest level $\mathsf{HW}(1)$ of the hierarchy coincides with the well-known class AC of *acyclic* UCQs [36], which admits a simple definition. A UCQ Θ is in AC 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 θ, such that for each variable x in θ 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 $\theta_n := \bigwedge_{1 \leq i < j \leq n} E(x_i, x_j) \wedge T_n(x_1, \dots, x_n)$. Each CQ $\theta_n \in \mathcal{C}$ is acyclic; this is witnessed by the join tree T whose root r corresponds to the atom $T_n(x_1, \dots, x_n)$ 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 $\{E(x_1, x_2) \wedge \cdots \wedge E(x_{n-1}, x_n) \wedge R(x_1, x_n) \mid n \geq 3\}$. Each CQ in $\mathcal{C}$ is in $\mathsf{TW}(2)$, but no such CQ is acyclic.

4. CONTAINMENT OF DATALOG IN TRACTABLE CLASSES OF UCQS

It is well-known that each Datalog program Π can be expressed as an infinite union $\bigcup_{i \geq 1} \theta_i^\Pi$ of CQs. The θ_i^Π's are called the *expansions* of Π (see, e.g., [29]). Therefore, checking containment of Π in a UCQ Θ reduces to checking containment of the infinite UCQ $\bigcup_{i \geq 1} \theta_i^\Pi$ in Θ. Restricting Θ to belong to one of the tractable classes $\mathsf{TW}(k)$ and $\mathsf{HW}(k)$ 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 $\bigcup_{i \geq 1} \theta_i^\Pi$ (and, thus, of Π) in Θ. 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 Π be a Datalog program over σ with distinguished symbol Q. Then:

1. The nodes of an expansion tree of Π are labeled with instances of rules in Π.

2. The root of an expansion tree of Π 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 ρ and $R_1(\bar{x}_1), \dots, R_\ell(\bar{x}_\ell)$ are the atoms of the body of ρ over the schema $\mathrm{IRels}(\Pi)$, it is the case that u has exactly ℓ children $u_1, \dots, u_\ell$ in the expansion tree and each u_j is labeled with an instance ρ_j of a rule in Π whose head is $R_j(\bar{x}_j)$, for $1 \leq j \leq \ell$.

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

4. For every node u of an expansion tree labeled ρ, it is the case that every variable y mentioned in the body of ρ either appears in the head of ρ 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 Π can be used.

Clearly, the number of different expansion trees of a program Π might be infinite (in particular, when Π is *recursive*).

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

This last condition is usually rephrased in terms of the existence of a *containment mapping* from θ' to τ [12], i.e. a mapping μ from the variables of θ' to the variables that label the atoms of τ, such that (i) μ is the identity over $\bar{x}$, and (ii) for each atom $R(\bar{y})$ in θ' it is the case that $R(\mu(\bar{y}))$ appears in the label of a node of τ. Clearly, $\bar{x} \in \theta'(\mathcal{D}_{\theta_\tau})$ iff there is a containment mapping from θ' to τ. Summing up:

PROPOSITION 2. *Let Π be a Datalog program and Θ a UCQ. Then $\Pi \subseteq \Theta$ iff for every expansion tree τ of Π there is a CQ $\theta \in \Theta$ and a containment mapping from θ to τ.*

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 CONT(Datalog,UCQ). In particular, we prove that 2EXPTIME-hardness for checking containment of a Datalog program Π in a UCQ Θ holds even if Θ is in $\mathsf{HW}(1) = \mathsf{AC}$, or it is in $\mathsf{TW}(2)$. These negative results are quite resilient: They hold even for UCQs in $\mathsf{TW}(2)$ or $\mathsf{HW}(2)$ over schemas of fixed arity. Formally:

THEOREM 5. *The following problems are* 2EXPTIME-*hard:*

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

None of these results follows from existing hardness results for 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 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 δ^ℓ and δ^r. 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 δ^ℓ two ternary relations I^ℓ, F^ℓ and a 4-ary relation B^ℓ on symbols that characterize the left successor relation. If $\bar{a} = a_1 \cdots a_m$ and $\bar{b} = b_1, \cdots, b_m$ are two configurations, then $\bar{b}$ is a left successor of $\bar{a}$ iff $(a_1, a_2, b_1) \in I^\ell$, $(a_{m-1}, a_m, b_m) \in F^\ell$ and $(a_{i-1}, a_i, a_{i+1}, b_i) \in B^\ell$, for each $1 < i < m$. Analogously, we associate with δ^r the relations I^r, F^r and B^r.

We construct a Datalog program Π and a UCQ Θ in AC that encode accepting computations of M. The expansions of Π will correspond to configuration trees, and each disjunct in Θ will detect a particular *error* that prevents an expansion from being an accepting computation. Thus, if an expansion τ 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 Π and Θ consists of a symbol A of arity $n+8$, a unary symbol $Start$ and a unary symbol Q_s, 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 Π. The program Π contains an intensional relation symbol B of arity $n+7$ and a 0-ary distinguished symbol C. Expansion trees of Π correspond to computation trees of M. In an expansion tree, the predicate B propagates the information while it generates atoms of the form $A(x, y, z, z', a_1, \dots, a_n, u, v, w, t), Q_s(z)$. Intuitively, this represents that, in the current configuration, the position at address $a_1 \cdots a_n$ contains the symbol s.

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

$$\begin{aligned} &B(x,y,z,a_1,\dots,a_n,u,v,w,t)\\ &\quad\leftarrow B(x,y,z,a_1,\dots a_{i-1},x,a_{i+1}\dots,a_n,u,v,w,t).\\ &B(x,y,z,a_1,\dots,a_n,u,v,w,t)\\ &\quad\leftarrow B(x,y,z,a_1,\dots a_{i-1},y,a_{i+1}\dots,a_n,u,v,w,t).\end{aligned}$$

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, Π contains the rule:

$$\begin{aligned} &B(x,y,z,a_1,\dots,a_n,u,v,w,t)\\ &\quad\leftarrow A(x,y,z,z',a_1,\dots,a_n,u,v,w,t),\\ &\qquad Q_s(z), B(x,y,z',a_1,\dots,a_n,u,v,w,t).\end{aligned}$$

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:

$$\begin{aligned} &B(x,y,z,a_1,\dots,a_n,u,v,w,x)\\ &\quad\leftarrow A(x,y,z,z',a_1,\dots,a_n,u,v,w,x),\\ &\qquad Q_s(z), B(x,y,z',a_1,\dots,a_n,u',u,w',y).\\ &B(x,y,z,a_1,\dots,a_n,u,v,w,x)\\ &\quad\leftarrow A(x,y,z,z',a_1,\dots,a_n,u,v,w,x),\\ &\qquad Q_s(z), B(x,y,z',a_1,\dots,a_n,u',v',u,y).\end{aligned}$$

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 :

$$\begin{aligned} B(x, y,z, a_1, \dots, a_n, u, v, w, y) & \\ \leftarrow A(x, y, z, z', a_1, \dots, a_n, u, v, w, y), Q_s(z) & \\ B(x, y, z', a_1, \dots, a_n, u', u, w', x), & \\ B(x, y, z', a_1, \dots, a_n, u', v', u, x). & \end{aligned}$$

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

$$C \leftarrow Start(z), B(x, y, z, x, \dots, x, u, v, w, x).$$

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

$$\begin{aligned} B(x, y,z, a_1, \dots, a_n, u, v, w, t) & \\ \leftarrow Q_s(z), A(x, y, z, z', a_1, \dots, a_n, u, v, w, t), & \end{aligned}$$

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

The UCQ Θ. Now we show how to detect errors in the expansion trees of Π using a Boolean acyclic UCQ Θ. Each disjunct of Θ 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 $0, \dots, 0$, (b) there are two consecutive addresses $\bar{a}$ and $\bar{b}$ such that $\bar{b} \neq \bar{a} + 1 \pmod{2^n}$, (c) a configuration does not change when the address is $1, \dots, 1$, and (d) a configuration changes when the address is not $1, \dots, 1$. 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 δ^ℓ and δ^r 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 $(a, b, c, d) \notin B^\ell$ we add the CQ that is defined by the conjunction of every atom in the set:

$$\begin{aligned} \Phi(a, b, c, d) \;:=\; \{ & A(x, y, z_1, z_2, \bar{a}_1, u, v, w, u_1),\ Q_a(z_1), \\ & A(x, y, z_2, z_3, \bar{a}_2, u, v, w, u_2),\ Q_b(z_2), \\ & A(x, y, z_3, z_4, \bar{a}_3, u, v, w, u_3),\ Q_c(z_3), \\ & A(x, y, z, z', \bar{a}_2, u', u, w', u_4), Q_d(z)\}. \end{aligned}$$

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 $\bar{a}_2$ 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 $(a, b, c) \notin I^\ell$, for each $(a, b, c) \notin F^\ell$, and for detecting errors in δ^r.

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 Π and Θ can be carried out in polynomial time, it follows that CONT(Datalog,AC) is 2EXPTIME-hard. □

4.2 Better bounds for queries in AC_k

Since restricting to a tractable class of UCQs the right-hand side of the problem 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 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 $\Phi(a, b, c, d)$, for $(a, b, c, d) \notin B^\ell$). 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 AC_k, for $k \geq 1$, to be the class of queries Θ in AC such that no two distinct atoms in a CQ in Θ share more than k variables. Clearly:

$$\mathsf{AC}_1 \subset \mathsf{AC}_2 \subset \cdots \subset \mathsf{AC}_k \subset \cdots \subset \mathsf{AC}.$$

EXAMPLE 4. The CQ $\bigwedge_{1 \leq i < n} E(x_i, x_{i+1})$, for $n \geq 2$, is in AC_1. The CQ $\bigwedge_{1 \leq i < j \leq n} E(x_i, x_j) \wedge T_n(x_1, \dots, x_n)$, for $n \geq 2$, is in AC_2. □

We prove next that CONT(Datalog,AC_k) can be solved in single-exponential time, for each $k \geq 1$.

THEOREM 6. *Let $k \geq 1$. Then* CONT(Datalog,AC_k) *is* EXPTIME-*complete.*

Proof (Sketch): We only sketch the upper bound. We assume familiarity with standard (one-way) *nondeterministic tree automata* [32], 1NTA, and *two-way alternating tree automata* [34], 2ATA, which extend 1NTA with both upward moves and alternation.

Overall idea. It is known that each expansion tree of a Datalog program Π can be represented as a ranked tree over a finite alphabet Σ_Π. Such representations are called the *proof trees* of Π. The language of all proof trees of Π can be defined by a 1NTA $\mathcal{A}_\Pi$ over Σ_Π, such that $\mathcal{A}_\Pi$ can be constructed in single exponential time from Π [12]. Consider first the case when Θ is an arbitrary UCQ (i.e., not necessarily in AC_k). Then it is possible to construct in single exponential time from Π and a UCQ Θ, a 1NTA $\mathcal{A}_\Pi^\Theta$ that accepts precisely the proof trees ν of Π such that there is a CQ $\theta \in \Theta$ and a containment mapping from θ to the expansion tree τ represented by ν [12]. (Both $\mathcal{A}_\Pi$ and $\mathcal{A}_\Pi^\Theta$ 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 1NTA $\mathcal{A}_\Pi$ is contained in the one defined by $\mathcal{A}_\Pi^\Theta$. The latter can be done in double exponential time in the size of Π and Θ [32].

In our case, that is, when Θ belongs to AC_k, we can follow a slightly different approach and check containment of Π in Θ in single exponential time. We start again by constructing in EXPTIME the 1NTA $\mathcal{A}_\Pi$ over Σ_Π that defines the set of proof trees of Π. The difference is that we now construct in exponential time from Π and Θ a 2ATA $\mathcal{B}_\Pi^\Theta$ of *polynomial size*[1], that accepts all trees T over Σ_Π such that there exists a CQ $\theta \in \Theta$ and a *strong* containment mapping from θ to T. Notice that, in opposition to $\mathcal{A}_\Pi^\Theta$, the 2ATA $\mathcal{B}_\Pi^\Theta$ may accept trees over Σ_Π that are not proof trees of Π.

Checking whether $\Pi \subseteq \Theta$ reduces to check if the language defined by $\mathcal{A}_\Pi$ is contained in the one defined by $\mathcal{B}_\Pi^\Theta$. In order to

[1] This means that even if $|\Sigma_\Pi|$ is exponential, both the set of states $\mathcal{S}$ of $\mathcal{B}_\Pi^\Theta$ and the sets of the form $\delta(s, a)$, where δ is the transition function of $\mathcal{B}_\Pi^\Theta$, $s \in \mathcal{S}$ and $a \in \Sigma_\Pi$, are of polynomial size.

do the latter, we first compute from $\mathcal{B}_\Pi^\Theta$ a 1NTA that defines the complement of $\mathcal{B}_\Pi^\Theta$, and then check for nonemptiness the intersection of this 1NTA with $\mathcal{A}_\Pi$. This tells us whether $\Pi \not\subseteq \Theta$. The result now follows from EXPTIME being closed under complement and the following facts: (1) A 1NTA that defines the complement of $\mathcal{B}_\Pi^\Theta$ can be constructed in single exponential time from Π and Θ [17], and (2) checking if the language defined by the intersection of this 1NTA with $\mathcal{A}_\Pi$ is nonempty can be solved in time exponential in the size of Π and Θ.

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

In an expansion tree, when we "unfold" a node we take a "fresh" copy of a rule r in Π. 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 Π, 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 Π can be described by a tree automaton $\mathcal{A}_\Pi$. Formally, let Σ_Π be the alphabet that consists of all instances of rules of Π that can be formed with variables from $vars(\Pi)$. Then:

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

Before following with the construction, it is necessary to understand when two occurences of the same variable in a proof tree ν denote the same element in the expansion tree represented by ν. Let T be a tree over Σ_Π (e.g., T might be a proof tree of Π). Assume n_1 and n_2 are nodes of T with lowest common ancestor n, and x_1 and x_2 are occurences of 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 Π in Θ can be rephrased in terms of the existence of *strong* containment mappings from the CQs in Θ to trees T over Σ_Π [12]. Let $\theta(\bar{x})$ be a CQ in Θ and T a tree over Σ_Π such that the head of the rule that labels the root of T is $Q(\bar{x}')$, for $|\bar{x}| = |\bar{x}'|$. A strong containment mapping μ from $\theta(\bar{x})$ to T maps occurrences of variables in θ to occurrences of variables in $vars(\Pi)$, in such a way that the following holds: (i) if y_1 and y_2 are two occurences of the same variable in θ, then $\mu(y_1)$ and $\mu(y_2)$ are connected occurrences of the same variable in T, (ii) for each occurence x of the i-th variable x_i of $\bar{x}$ in θ, it is the case that $\mu(x)$ is a distinguished occurence in T of the i-th variable x_i' of $\bar{x}'$, and (iii) for each atom in θ 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 θ to a proof tree ν enforces that different occurences of the same variable y in θ are mapped to the same element in the expansion tree τ represented by ν. With this idea in mind, it is not hard to prove that there is a strong containment mapping from θ to ν iff there is a containment mapping from θ to τ. We obtain the following from Proposition 2:

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

In order to complete the proof of the theorem, it will be sufficient to prove Lemma 3 below. This lemma states that, if $\Theta \in \mathsf{AC}_k$, for $k \geq 1$, it is possible to construct in single exponential time a 2ATA $\mathcal{B}_\Pi^\Theta$ of polynomial size, that accepts precisely the trees T over Σ_Π for which there is a CQ $\theta \in \Theta$ and a strong containment mapping from θ to T. It will follow then from Lemmas 1, 2 and 3, that checking $\Pi \subseteq \Theta$ reduces to checking $\mathcal{A}_\Pi \subseteq \mathcal{B}_\Pi^\Theta$. As mentioned before, this can be done in EXPTIME using known techniques.

LEMMA 3. *Let $k \geq 1$. There is an* EXPTIME *algorithm that, given a Datalog program Π and a UCQ $\Theta \in \mathsf{AC}_k$, constructs a 2ATA $\mathcal{B}_\Pi^\Theta$, whose size is polynomial in the size of Π and Θ, such that the language defined by $\mathcal{B}_\Pi^\Theta$ is the set of trees T over Σ_Π for which there is a CQ $\theta \in \Theta$ and a strong containment mapping from θ to T.*

The 2ATA $\mathcal{B}_\Pi^\Theta$ looks for a strong containment mapping from some CQ in Θ to the tree T. In order to do this, it first guesses a CQ $\theta \in \Theta$, and then operates in a top-down fashion over the join tree of θ (which exists since θ is acyclic). At each point, $\mathcal{B}_\Pi^\Theta$ is in a state that summarizes the following information: the atom A of the join tree of θ that is being scanned (which is the one that $\mathcal{B}_\Pi^\Theta$ is currently trying to match in T), and an assignment M from the variables of θ 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 $\mathcal{B}_\Pi^\Theta$ is defined as $\bigcup_{\theta \in \Theta} \mathcal{B}_\Pi^\theta$. We define 2ATAs of the form $\mathcal{B}_\Pi^\theta$ below. Let J be a join tree of θ. We write J also to refer to the set of atoms of θ. Assume that $x_1, \dots, x_m$ are the distinguished variables of θ. For an atom A of θ, we define $dvar(A)$ to be the set of indices in $\{1, \dots, m\}$ such that x_i appears in A.

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

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

The transition function δ of $\mathcal{B}_\Pi^\theta$ is (in broad terms) defined as follows. Consider first a pair $((A, M), R(\bar{t}) \leftarrow \rho) \in \mathcal{S} \times \Sigma_\Pi$, where (A, M) is an atom state, i.e., $A \in J$ and $M \in \mathcal{M}_k$, and $R(\bar{t}) \leftarrow \rho$ is an instance of a rule in Π over $vars(\Pi)$. Then:

1. It is the case that $\delta((A, M), R(\bar{t}) \leftarrow \rho)$ contains an *atom mapping* transition of the form:

   $$\langle 0, (A_1, M_1) \rangle \wedge \cdots \wedge \langle 0, (A_n, M_n) \rangle \wedge \\ \langle 0, (j_1, x'_{j_1}) \rangle \wedge \cdots \wedge \langle 0, (j_p, x'_{j_p}) \rangle,$$

   if $A_1, \dots, A_n$ 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 ρ.

(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) $dvar(A) = \{j_1, \ldots, j_p\}$ 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 $\mathcal{B}^{\theta}_{\Pi}$ 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 $\langle 0, (A_i, M_i)\rangle$ 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 $\langle 0, (j_1, x'_{j_1})\rangle$ in the atom mapping transition, as we will see below.

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

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

   (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 $\bar{t}$.

   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 $((j, x), R(\bar{t}) \leftarrow \rho)$, where (j, x) is a variable atom, i.e. $j \in \{1, \ldots, m\}$ and $x \in vars(\Pi)$, and $R(\bar{t}) \leftarrow \rho$ is an instance of a rule of Π over $vars(\Pi)$. Then $\delta((j, x), R(\bar{t}) \leftarrow \rho)$ contains a "variable checking" transition $\langle -1, (j, x)\rangle$ if the variable x is in $\bar{t}$. In addition, it contains a *variable checking* transition $\langle 0, accept\rangle$ if the j-th variable of $\bar{t}$ is x. The goal of the *variable checking* transitions is to ensure that each distinguished occurence of a variable x in θ 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 $\mathcal{B}^{\theta}_{\Pi}$ defines the set of trees T over Σ_{Π} for which there is a strong containment mapping from θ to T. Notice that the number of states of $\mathcal{B}^{\theta}_{\Pi}$ is $O(||\theta||^{k+1}||\Pi||^{k})$, i.e., it is polynomial in the size of Π and θ. Also, for each state s of $\mathcal{B}^{\theta}_{\Pi}$ and symbol $a \in \Sigma_{\Pi}$, the size of $\delta(s, a)$ is polynomial. This finishes our sketch. □

4.2.1 *Results for* AC *and* TW(1)

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

COROLLARY 1. *The following problems are complete for* EXPTIME*:*

1. CONT$_c$(Datalog,AC)*, for each* $c \geq 1$.
2. CONT(Datalog,TW(1)).

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

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

4.2.2 *Results about equivalence*

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

COROLLARY 2. *Let* $k \geq 1$. *The following problem is complete for* EXPTIME*: Given a Datalog program* Π *and a UCQ* Θ *in* AC$_k$*, decide if* Π *is equivalent to* Θ.

4.2.3 *Queries in* AC$_k$ *modulo equivalence*

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

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

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

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

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

3. The free variables of θ' are exactly the free variables of θ.

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

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

Let T be a join tree of θ. 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 θ, and another one – based on the fact that T can also be interpreted as a tree decomposition of θ – in which each such node is associated with the set of variables in V_{θ} that are mentioned in A. Consider now a node of T that is associated with some atom A that is not in θ'. Assume that the set of variables mentioned in A is S (notice that $S \not\subseteq V_{\theta'}$, since θ' is a strong induced subquery of θ). Let

$S_{\theta'}$ be the restriction of S to $V_{\theta'}$, and assume for the time being that node A is replaced in T by $S_{\theta'}$. By condition (4), there is a node of T that is associated with some set S' of variables such that $S_{\theta'} \subseteq S' \subseteq V_{\theta'}$. Thus, $S_{\theta'}$ is subsumed by node S', which in turn only contains variables in θ'. Since $S_{\theta'}$ is contained in S', we can apply standard tree decomposition techniques (see, e.g., [21]), and transform T into a new tree T' by "contracting" the node $S_{\theta'}$ into the node S' (that is, $S_{\theta'}$ but not S' is removed from T), in such a way that the connectivity properties of the variables in $V_{\theta'}$ are preserved in T'. It is easy to see that by iteratively applying this transformation to each node of T associated with an atom A in θ but not in θ', we will end up with a join tree for θ'.

We show next that if $\theta \in \mathsf{AC}_k$, then its *core* [26, 10] θ' is a strong induced query. It follows from well-known core properties [26] that conditions (1), (2) and (3) hold. We prove next that condition (4) also holds. Again by well-known properties of cores [26], we have that there is a homomorphism μ from θ to $\mathcal{D}_{\theta'}$ such that $\mu(x) = x$, for each $x \in V_{\theta'}$. Let $A(x_1, \dots, x_m)$ be an atom in θ but not in θ'. Then it is the case that $A(\mu(x_1), \dots, \mu(x_m))$ is an atom of θ'. Since μ is the identity in V_A, we conclude that condition (4) holds.

Now, it is easy to prove the following observation: A CQ θ is in $\mathcal{H}(\mathsf{AC}_k)$ iff its core θ' is in AC_k. In fact, assume first that $\theta' \in \mathsf{AC}_k$. Since θ is equivalent to θ', we have that $\theta \in \mathcal{H}(\mathsf{AC}_k)$. Assume, on the other hand, that θ is equivalent to a CQ θ^* in AC_k. It is well-known that cores of equivalent CQs are isomorphic [26]. But the core of θ^* is in AC_k (since θ^* is in AC_k), and, therefore, the core of θ' is in AC_k.

The latter can be used to prove the following: Let $k \geq 1$. There is an EXPTIME algorithm that, given a UCQ $\Theta \in \mathcal{H}(\mathsf{AC}_k)$, constructs a UCQ $\Theta^* \in \mathsf{AC}_k$ such that (i) Θ^* is equivalent to Θ, and (ii) the size of Θ^* is at most the size of Θ. In fact, let $\Theta \in \mathcal{H}(\mathsf{AC}_k)$. Then there exists an equivalent $\Theta' \in \mathsf{AC}_k$. Let Θ_{min} be a subset of the CQs in Θ such that (1) Θ_{min} is equivalent to Θ, and (2) no proper subset of Θ_{min} is equivalent to Θ. Analogously, we define Θ'_{min} to be "minimally" equivalent to Θ'. By minimality, there are no distinct CQs θ_1 and θ_2 in Θ_{min} such that $\theta_2 \subseteq \theta_1$. Moreover, since Θ_{min} and Θ'_{min} are equivalent, it follows that, for each $\theta \in \Theta_{min}$, there is an equivalent CQ θ' in Θ'_{min}. Thus, each $\theta \in \Theta_{min}$ belongs to $\mathcal{H}(\mathsf{AC}_k)$. Let Θ^* be the UCQ Θ_{min}, where each CQ is replaced by its core. By our previous observations, it follows that $\Theta^* \in \mathsf{AC}_k$. Moreover, Θ^* is equivalent to Θ and its size is at most the size of Θ. The algorithm then proceeds as follows: It first computes Θ_{min} from Θ, and then it constructs Θ^* from Θ_{min}. It is not hard to prove that each step of the algorithm can be carried out in single exponential time.

This implies that in order to check if the Datalog program Π is contained in the UCQ $\Theta \in \mathcal{H}(\mathsf{AC}_k)$, we can first construct Θ^* from Θ in EXPTIME, and then check whether $\Pi \subseteq \Theta^*$. The latter can be done in EXPTIME from Theorem 6. □

Is it decidable to check membership in $\mathcal{H}(\mathsf{AC}_k)$? It follows from the techniques developed in the proof of the previous proposition that this is indeed the case. In addition, those techniques allow us to pinpoint the precise complexity of the problem.

PROPOSITION 4. *Let $k \geq 1$. Then checking if an UCQ Θ is in $\mathcal{H}(\mathsf{AC}_k)$ is* NP*-complete.*

5. CONTAINMENT OF DATALOG IN TRACTABLE CLASSES OF UC2RPQS

We now switch to study the containment problem in the context of graph databases. We start by introducing the basic notions used in this section.

5.1 Preliminaries

Graph databases. Let Σ be a finite alphabet. A *graph database* $\mathcal{G}$ over Σ 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 a-labeled edge from v to v' in $\mathcal{G}$. A *path* in a graph database $\mathcal{G} = (V, E)$ is a sequence

$$\eta = v_0 a_0 v_1 a_1 v_2 \cdots v_{k-1} a_{k-1} v_k,$$

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 η, denoted $\lambda(\eta)$, is the string $a_0 a_1 \cdots a_{k-1} \in \Sigma^*$. Notice that v is a path, for each $v \in V$. The label of such path is the empty string ϵ.

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 Σ is a regular expression over the alphabet $\Sigma^{\pm}$ that extends Σ with the *inverse* a^- 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 $\mathcal{G}^{\pm}$. This is the graph database over $\Sigma^{\pm}$ 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 2RPQ L over $\mathcal{G}$ to be the set of pairs (v, v') of nodes in V such that there is a path η in $\mathcal{G}^{\pm}$ 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 γ over Σ is an expression of the form

$$\exists \bar{z}(L_1(x_1, y_1) \wedge \cdots \wedge L_m(x_m, y_m)),$$

where each L_i is a 2RPQ over Σ, for $1 \leq i \leq m$, the x_i's and y_i's are variables, and $\bar{z}$ 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 γ.

A homomorphism from a C2RPQ $\gamma = \exists \bar{z}(L_1(x_1, y_1) \wedge \cdots \wedge L_m(x_m, y_m))$ to a graph database $\mathcal{G} = (V, E)$ is a mapping h from the set of variables used in γ 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 $\gamma(\bar{x})$ over $\mathcal{G}$ is the set of tuples $h(\bar{x})$, for h a homomorphism from γ to $\mathcal{G}$.

A UC2RPQ Γ is a finite set $\{\gamma_1(\bar{x}), \dots, \gamma_k(\bar{x})\}$ of C2RPQs over Σ 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 Σ 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 Π over Σ 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 Π can be of arbitrary arity. It is a well-known fact that Datalog subsume 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 CONT(Datalog,$\mathcal{C}$). Formally, CONT(Datalog,$\mathcal{C}$) is defined as follows: Given a Datalog program Π and a UC2RPQ $\Gamma \in \mathcal{C}$ over the same finite alphabet Σ, 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 CONT(Datalog,UC2RPQ) is in general not only decidable, but also not more expensive than the problem CONT(Datalog,UCQ).

THEOREM 7. [8] *The problem* CONT(Datalog,UC2RPQ) *is complete for* 2EXPTIME.

5.2 Containment of Datalog in Acyclic UC2RPQs

In general, evaluation of C2RPQs is NP-complete [1], but a tractable class can be obtained by restricting the syntactic shape of queries to be *acyclic* [3]. This notion is typically defined in terms of the acyclicity of its *underlying conjunctive query*. Let $\gamma = \exists \bar{y} \bigwedge_{1 \leq i \leq m} L_i(x_i, y_i)$ be a C2RPQ. Its underlying CQ is the query over the schema of binary relation symbols $T_1, \dots, T_m$ defined as: $\exists \bar{y} \bigwedge_{1 \leq i \leq m} T_i(x_i, y_i)$. Intuitively, this underlying conjunctive query represents the structure of γ when the regular languages that label the atoms of γ are turned into relation symbols. A C2RPQ is *acyclic* if its underlying CQ is acyclic (or, equivalently, if it belongs to $\mathsf{TW}(1)$, since $\mathsf{AC} = \mathsf{TW}(1)$ for binary schemas). A UC2RPQ is acyclic if each one of its C2RPQs is acyclic. We denote by ACR the class of acyclic UC2RPQs.

EXAMPLE 5. Let L_1, L_2 and L_3 be regular expressions over Σ. The C2RPQ $L_1(x,x) \wedge L_2(x,y) \wedge L_3(y,x)$ is acyclic. The C2RPQ $L_1(x,y) \wedge L_2(y,z) \wedge L_3(z,x)$ is not acyclic. □

We proved in Theorem 6 that containment of Datalog in UCQs in $\mathsf{TW}(1)$ is in 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 queries in ACR also helps alleviate the complexity of CONT(Datalog,UC2RPQs). We prove that this is not the case, by showing that the 2EXPTIME-hardness proof for CONT(Datalog,UC2RPQ) in [8] can be carried out even with UC2RPQs in ACR.

THEOREM 8. *The problem* CONT(Datalog,ACR) *is complete for* 2EXPTIME.

5.3 Better bounds for queries in ACR^k

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

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

$$\mathsf{ACR}^1 \subset \mathsf{ACR}^2 \subset \cdots \subset \mathsf{ACR}^k \subset \cdots \subset \mathsf{ACR}.$$

Interestingly, queries in ACR^1 have been previously studied under the name of *strongly acyclic* UC2RPQs [1].

EXAMPLE 6. The C2RPQ $L_1(x,x) \wedge L_2(x,y) \wedge L_3(y,x)$ is in ACR^2. □

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

THEOREM 9. *Let* $k \geq 1$. *Then* CONT(Datalog,ACR^k) *is complete for* EXPTIME.

Proof (Sketch): Containment of a Datalog program Π in a UC2RPQ Γ can be stated in terms of the existence of strong containment mappings from the *expansions* of the C2RPQs in Γ to the proof trees of Π. An expansion of a C2RPQ $\gamma = \exists \bar{z} \bigwedge_{1 \leq i \leq m} L_i(x_i, y_i)$ over Σ is a CQ over $\sigma(\Sigma^{\pm})$ of the form $\phi = \exists \bar{z} \bigwedge_{1 \leq i \leq m} \theta_i(x_i, y_i)$, where for each i with $1 \leq i \leq m$ it is the case that $\theta_i(x_i, y_i)$ is of the form:

$$\exists u_1 \cdots u_{n-1} (E_{a_1}(x_i, u_1) \wedge E_{a_2}(u_1, u_2) \wedge \cdots \wedge E_{a_n}(u_{n-1}, y_i)),$$

for $a_1 a_2 \cdots a_n$ a word over $\Sigma^{\pm}$ that satisfies L_i. We assume no two distinct θ_i's share an existentially quantified variable. Strong containment mappings μ from the expansion ϕ to the proof tree ν of Π are defined as in the proof of Theorem 6, save for condition (iii) that is restated as follows: for each atom A in θ we have that if A is of the form $E_a(x,y)$, for $a \in \Sigma$, then $E_a(\mu(x), \mu(y))$ appears in ν, and if A is of the form $E_{a^-}(x,y)$, for $a \in \Sigma$, then $E_a(\mu(y), \mu(x))$ appears in ν. Then:

LEMMA 4. [8] *It is the case that* $\Pi \subseteq \Gamma$ *iff for every proof tree* ν *of* Π *there is a C2RPQ* $\gamma \in \Gamma$, *an expansion* ϕ *of* γ, *and a strong containment mapping from* ϕ *to* ν.

Consider the 1NTA $\mathcal{A}_\Pi$ over Σ_Π constructed in the proof of Theorem 6. Assume that $\Gamma \in \mathsf{ACR}^k$, for $k \geq 1$. We construct in EXPTIME a 2ATA $\mathcal{B}_\Pi^\Gamma := \bigcup_{\gamma \in \Gamma} \mathcal{B}_\Pi^\gamma$, whose size is polynomial in the size of Π and Γ, such that $\mathcal{B}_\Pi^\gamma$ satisfies the following: Given a proof tree ν over Σ_Π, the 2ATA $\mathcal{B}_\Pi^\gamma$ accepts ν iff there is an expansion ϕ of γ and a strong containment mapping from ϕ to ν. It follows from Lemmas 1 and 4 that $\Pi \subseteq \Gamma$ iff $\mathcal{A}_\Pi \subseteq \mathcal{B}_\Pi^\Gamma$. It is known that the latter can be checked in EXPTIME from Π and Γ [34].

We explain now how to construct $\mathcal{B}_\Pi^\gamma$. For each pair (x, x') of variables in γ, we denote by $\gamma(x,x')$ the set of atoms of the form $L(x,x')$ or $L(x',x)$ in γ. Consider the undirected graph G_γ whose vertices are the variables of γ, and there is an undirected edge between x and x' in G_γ iff $\gamma(x,x')$ is nonempty. Since γ is acyclic, the graph G_γ is a tree. We assume for the sake of this sketch that G_γ is connected and contains no loops, but both cases can be handled at the cost of a more cumbersome construction.

The 2ATA $\mathcal{B}_\Pi^\gamma$ reads G_γ in a top-down fashion, looking for the existence of an expansion ϕ of γ and a strong containment mapping μ from ϕ to ν. After scanning node x in G_γ, 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_γ the mapping μ can be extended to a strong containment mapping from some expansion of the atoms in $\gamma(x,y)$ to ν.

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

Suppose now that γ belongs to ACR^k, 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 ν. We explain next how to do this without incurring in an exponential blowup in the construction of $\mathcal{B}_\Pi^\gamma$ (recall that $\mathcal{B}_\Pi^\gamma$ must be of polynomial size).

The 2ATA $\mathcal{B}_\Pi^\gamma$ 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 ν. This idea is implemented in $\mathcal{B}_\Pi^\gamma$ using *multiedge* 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 $\mathcal{B}_\Pi^\gamma$ 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 $\mathcal{B}_\Pi^\gamma$ are of the form $(L_i)_{s,s'}(u, v)$ or $(L_i)_s(u, v)$, where s and s' are states in L_i, and u and v are variables mentioned in ν (i.e., $u, v \in vars(\Pi)$). When the automaton is in state $(L_i)_{s,s'}(u, v)$ (the case $(L_i)_s(u, v)$ is analogous), then u and v are variables in the rule r that labels the node currently being scanned by $\mathcal{B}_\Pi^\gamma$. The 2ATA $\mathcal{B}_\Pi^\gamma$ then looks for an expansion $\theta = \exists u_1 \dots \exists u_{n-1}(E_{a_1}(x', u_1) \wedge \cdots \wedge E_{a_n}(u_{n-1}, y'))$ of the C2RPQ $(L_i)_{s,s'}(x', y')$, where x' and y' are fresh variables, and a strong containment mapping μ_θ from θ to ν such that $\mu_\theta(x')$ corresponds to an occurence of u in ν that is connected to the occurrence of u in r, and similarly for $\mu_\theta(y')$ and v. Although the implementation of atom states requires some work, we can adapt techniques from [8] in order to construct the modules of $\mathcal{B}_\Pi^\gamma$ 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 multiedge states associated with $\gamma(x, y)$ in $\mathcal{B}_\Pi^\gamma$ 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 ν. When processing $\gamma(x, y)$, the automaton looks for expansions θ_1 and θ_2 of $L_1(x, y)$ and $L_2(x, y)$, respectively, and a strong containment mapping μ_{L_1,L_2} from these expansions to ν. The mapping μ_{L_1,L_2} must be consistent: the two occurrences of y in θ_1 and θ_2 must be mapped to connected occurrences of the same variable in ν. Intuitively, a multiedge state indicates the "suffixes" of the expansions θ_1 and θ_2 that remain to be mapped to ν. When processing $\gamma(x, y)$, the automaton starts in state $\gamma_{x,y}(i_1, i_2; u_x, u_x)$, where i_1 and i_2 are the initial states of L_1 and L_2, respectively, and u_x is a variable in the rule r that is currently being scanned by $\mathcal{B}_\Pi^\gamma$, such that the strong containment mapping constructed so far from the expansion ϕ of G_γ to ν maps x to u_x (formally, each image via this mapping of an occurrence of x in γ is connected to the occurrence of u_x in r).

When $\mathcal{B}_\Pi^\gamma$ is in state $\gamma_{x,y}(s_1, s_2; u_1, u_2)$, for u_1 and u_2 variables that appear in the rule r currently being scanned, the automaton looks for expansions θ_1' of $(L_1)_{s_1}(z_1, y)$ and θ_2' of $(L_2)_{s_2}(z_2, y)$, where z_1, z_2 are fresh variables, and strong containment mappings μ'_{L_1} and μ'_{L_2} from θ_1' and θ_2' to ν, respectively, such that (1) $\mu'_{L_1}(z_1)$ is an occurence of u_1 that is connected to the occurence of u_1 in r, and similarly for $\mu'_{L_2}(z_2)$ and v_2, and (2) $\mu'_{L_1}(y)$ and $\mu'_{L_2}(y)$ are connected occurrences of the same variable w in ν. We explain next how this is done.

We consider two cases:

1. The rule r currently being scanned contains an occurence of w that is connected to $\mu'_{L_1}(y)$ and $\mu'_{L_2}(y)$. This case is simple since we can continue processing θ_1' and θ_2' independently from the current node. This is done by performing a universal transition $\langle 0, (L_1)_{s_1}(u_1, w)\rangle \wedge \langle 0, (L_2)_{s_2}(u_2, w)\rangle$.

2. Otherwise, it is easy to prove that there is $j \in \{-1, 1, \dots \ell\}$, where ℓ 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 ν_j of ν rooted at the j-th child of the node currently being scanned, if $j \geq 1$, and to the subtree ν_{-1} of ν 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 ν_j does not contain any occurrence of a variable connected to u_1 or u_2. Indeed, some "prefixes" of the expansions θ_1' and θ_2' could be mapped outside ν_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 $\mathcal{B}_\Pi^\gamma$ guesses "prefixes" of the expansions θ_1' and θ_2' that are mapped outside ν_j via μ'_{L_1} and μ'_{L_2}, and continues processing the corresponding "suffixes".

   More formally, let A be the first atom in the expansion θ_1' that is mapped via μ'_{L_1} inside ν_j. If A is the first atom in the expansion θ_1', then we let u_1' to be u_1. Otherwise, let A' be the atom that precedes A in θ_1' (notice that A' must be mapped via μ'_{L_1} outside ν_j). In this case, we let u_1' to be the one variable A' and A have in common. Analogously, we define u_2' with respect to θ_2' and μ'_{L_2}. Notice that rule r must contain occurrences of u_1' and u_2'. Moreover, the subtree ν_j must contain occurrences of u_1' and u_2' that are connected to the occurrences of u_1' and u_2' in r, respectively. Now the automaton $\mathcal{B}_\Pi^\gamma$ is ready to move in direction j. This is done by performing the following universal transitions:

   - $\langle 0, (L_1)_{s_1,s_1'}(u_1, u_1')\rangle$ and $\langle 0, (L_2)_{s_2,s_2'}(u_2, u_2')\rangle$, for s_1' and s_2' suitable states in L_1 and L_2, respectively.
   - $\langle j, \gamma_{x,y}(s_1', s_2'; u_1', u_2')\rangle$. This corresponds to guessing the "suffixes" 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 θ_1 and θ_2 of $L_1(x, y)$ and $L_2(x, y)$ have been correctly mapped to ν.

In general, the number of multiedges 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 $\mathcal{B}_\Pi^\gamma$ is polynomial in the size of Π and γ. In addition, there is a polynomial that bounds the size of $\delta(t, a)$, for each state t of $\mathcal{B}_\Pi^\gamma$ and symbol $a \in \Sigma_\Pi$. 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 $\mathsf{TWR}(2)$ be the class of UC2RPQs Γ such that for each C2RPQ $\gamma \in \Gamma$ we have that (1) the underlying CQ of γ 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 γ for each pair (x, x') of distinct variables. Then:

Proposition 5. *The problem* Cont(Datalog,$\mathsf{TWR}(2)$) *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 Cont(Datalog,UCQ) and Cont(Datalog,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 Π in the UCQ Θ is 2Exptime-complete even if Π 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 Θ to be in $\mathsf{TW}(k)$ or $\mathsf{HW}(k)$, for $k \geq 1$.

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

7. REFERENCES

[1] P. Barceló. Querying graph databases. In *PODS* 2013, pages 175-188.

[2] P. Barceló, R. Pichler (Eds.). *Datalog in Academia and Industry*. LNCS 7494, Springer 2012.

[3] P. Barceló, M. Romero, M. Y. Vardi. Semantic acyclicity on graph databases. In *PODS* 2013, pages 237-248.

[4] C. Beeri, R. Fagin, D. Maier, A. O. Mendelzon, J. D. Ullman, M. Yannakakis. Properties of acyclic database schemes. In *STOC* 1981, pages 355-362.

[5] M. Benedikt, P. Bourhis, P. Senellart. Monadic datalog containment. In *ICALP* 2012, pages 79-91.

[6] P. Buneman, S. B. Davidson, G. G. Hillebrand, D. Suciu. A query language and optimization techniques for unstructured data. In *SIGMOD* 1996, pages 505-516.

[7] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'00*, pages 176–185.

[8] D. Calvanese, G. de Giacomo, M. Y. Vardi. Decidable containment of recursive queries. *Theor. Comput. Sci.* 336(1), pages 33-56, 2005.

[9] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.

[10] A. Chandra, Ph. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC* 1977, pp. 77-90.

[11] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim. Optimizing queries with materialized views. In *ICDE* 1995, pages 190-200.

[12] S. Chaudhuri, M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *J. Comput. Syst. Sci.* 54(1), pages 61-78, 1997.

[13] S. Chaudhuri, M. Y. Vardi. On the complexity of equivalence between recursive and nonrecursive Datalog programs. In *PODS* 1994, pages 107-116.

[14] C. Chekuri, A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.* 239(2), pages 211-229, 2000.

[15] H. Chen, V. Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In *CP* 2005, pages 167-181.

[16] S. S. Cosmadakis, P. C. Kanellakis. Parallel Evaluation of Recursive Rule Queries. In *PODS* 1986, pages 280-293.

[17] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, M. Y. Vardi. Decidable optimization problems for database logic programs (Preliminary report). In *STOC* 1988, pages 477-490.

[18] V. Dalmau, P. Kolaitis, M. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP* 2002, pp. 310-326.

[19] O. de Moor, G. Gottlob, T. Furche, A. J. Sellers (Eds.). *Datalog Reloaded*. LNCS 6702, Springer 2011.

[20] M. F. Fernández, D. Florescu, A. Y. Levy, D. Suciu. Verifying integrity constraints on web sites. In *IJCAI* 1999, pages 614-619.

[21] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[22] M. Friedman, A. Y. Levy, T. D. Millstein. Navigational plans For data integration. In *AAAI/IAAI* 1999, pages 67-73.

[23] R. Fagin, M. Y. Vardi. The theory of data dependencies - An overview. In *ICALP* 1984, pages 1-22.

[24] G. Gottlob, N. Leone, F. Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.* 64(3), pages 579–627, 2002.

[25] M. Grohe, D. Marx. Constraint solving via fractional edge covers. In *SODA* 2006, pages 289-298.

[26] P. Hell, J. Nešetřil. The core of a graph. *Discr. Math.* 109, 1995.

[27] T. Imielinski, W. Lipski Jr. Incomplete information in relational databases. *J. of the ACM* 31(4), pages 761-791, 1984.

[28] S. Malik and L. Zhang. Boolean satisfiability: from theoretical hardness to practical success. *CACM* 52(8), 76–82, 2009.

[29] J. Naughton. Data independent recursion in deductive databases. *JCSS* 38, pages 259-289, 1989.

[30] A. Robinson, A. Voronkov, eds. *Handbook of Automated Reasoning*. The MIT Press, 2001.

[31] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operator. *J. of the ACM* 27(4), 1980, pages 633–655.

[32] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.* 19(3), pages 424-437, 1990.

[33] O. Shmueli. Equivalence of DATALOG queries is undecidable. em J. Log. Program. 15(3), pages 231-241, 1993.

[34] G. Slutzki. Alternating tree automata. *TCS* 41, pp. 305-318, 1985.

[35] P. T. Wood. Query languages for graph databases. *SIGMOD Record* 41(1), pages 50-60, 2012.

[36] M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB* 1981, pages 82-94.