Monadic Datalog Containment

Michael Benedikt, Pierre Bourhis, Pierre Senellart

Monadic Datalog Containment

Michael Benedikt[1], Pierre Bourhis[1], and Pierre Senellart[2]

[1] Oxford University, Oxford, United Kingdom
firstname.lastname@cs.ox.ac.uk
[2] Institut Mines–Télécom; Télécom ParisTech; CNRS LTCI, Paris, France
pierre.senellart@telecom-paristech.fr

Abstract. We reconsider the problem of containment of monadic datalog (MDL) queries in unions of conjunctive queries (UCQs). Prior work has dealt with special cases, but has left the precise complexity characterization open. We begin by establishing a 2EXPTIME lower bound on the MDL/UCQ containment problem, resolving an open problem from the early 90's. We then present a general approach for getting tighter bounds on the complexity, based on analysis of the number of mappings of queries into tree-like instances. We use the machinery to present an important case of the MDL/UCQ containment problem that is in co-NEXPTIME, and a case that is in EXPTIME. We then show that the technique can be used to get a new tight upper bound for containment of tree automata in UCQs. We show that the new MDL/UCQ upper bounds are tight.

1 Introduction

Context. Datalog represents a standard model for querying data with recursion. The basic problems of evaluation, equivalence, and containment thus have been the object of study for several decades. Shmueli [1] showed that containment (and equivalence) of datalog is undecidable. Decidable subclasses were subsequently isolated [2,3], focusing on restricting the form of recursion used. Chaudhuri and Vardi [4,5] provide an extensive study of the containment of datalog queries in nonrecursive datalog queries, showing in particular that the problem is decidable. They also show that it is 2EXPTIME-complete to decide containment of a datalog query within a union of conjunctive queries (UCQ).

In this article we focus on *monadic datalog* (MDL) – the fragment in which all intensional predicates are unary. Cosmodakis, Gaifman, Kanellakis, and Vardi [6] showed that containment of monadic datalog queries is in 2EXPTIME, and is EXPTIME-hard, leaving open the question of a tight bound. In another article, Chaudhuri and Vardi [7] prove a co-NEXPTIME upper bound for containment of a unary MDL query (i.e., a query with one output variable) in a union of unary connected conjunctive queries. Their co-NEXPTIME upper bound does not apply to Boolean queries, and, in fact, we will show that the problem is 2EXPTIME-hard even for connected Boolean queries. It also does not apply when either the MDL or the conjunctive queries have constants.

Since the work of Chaudhuri and Vardi, the "fine structure" of the containment problem between recursive and non-recursive queries has remained

A. Czumaj et al. (Eds.): ICALP 2012, Part II, LNCS 7392, pp. 79–91, 2012.

mysterious. What computation can one code up in a MDL/UCQ containment problem? What features of recursive queries make the containment problem hard? Put another way: What restriction on the queries or on the instances can make the problem easier? A better understanding of these issues can shed light on a number of other questions concerning containment between recursive and non-recursive queries: e.g., containment under limited access patterns [8,9], and containment of unions of conjunctive two-way regular path queries [10] in non-recursive queries.

Contributions. We start by showing that containment of MDL in UCQs is 2EXPTIME-complete, with the upper bound holding in the presence of constants and the lower bound even in their absence. This resolves the main open question from [7]. Since this is a special case of monadic datalog containment, it also shows that MDL containment is 2EXPTIME-complete, closing the gap of [6].

We then look to understand the factors leading to high complexity. We start by revisiting the 2EXPTIME upper bound for containment of Chaudhuri and Vardi, giving a refinement of the main two tools used there: reduction to tree-shaped models, and counting the number of different types of nodes in the models. We present a property of a class of instances, the Unique Mapping Condition, that suffices to show that the number of types reduces from doubly-exponential to exponential. We then show that whenever this condition holds, containment in UCQs goes down to co-NEXPTIME, and containment of constant-free queries goes down to EXPTIME. We give two settings where the Unique Mapping Condition holds. One is that of "globally extensionally limited" MDL (GEMDL) queries over general relational structures: we limit the number of occurrences of an extensional predicate in the program. A second is when the models are trees. In the second case, we get new upper bounds of containment of tree automata in UCQs. We believe that the upper bound techniques give new insight into implication problems for recursive queries in UCQs, and more generally for the understanding of homomorphisms of conjunctive queries in tree-like structures.

We round off these upper bound results with two complementary lower bounds, showing that bounds on containment of GEMDL queries are tight, and showing that slight generalizations of the GEMDL condition push the complexity of containment in UCQs back to 2EXPTIME. The lower bound arguments require a much more involved coding argument than for general MDL.

Organization. Section 2 contains the basic definitions. Section 3 discusses monadic datalog containment in general, and presents our main lower bound, exploring its consequences. Section 4 deals with the main new subclasses for which better bounds can be provided. Section 5 gives conclusions and discusses related work.

2 Background

Datalog. An *atom* over a relational signature S is an expression $R(x_1 \dots x_n)$ where R is an n-ary predicate of S and the x_i are either variables (from some countable set) or constants.

We consider the following three simple positive query languages over S: (a) conjunctive queries (CQs); (b) unions of conjunctive queries (UCQs); (c) positive

queries (PQs). They are respectively defined as the fragments of first-order logic over S built up from relation atoms and, respectively, (a) $\exists$ and $\wedge$; (b) disjunctions of CQs; (c) $\exists$, $\vee$, and $\wedge$.

A *datalog program* over S consists of: (i) A relational signature S. (ii) A set of rules of the form $A \leftarrow \varphi$, where φ is a conjunction of atoms over S, and A is an atom over S. A is the *head* of the rule and φ is the *body* of the rule. We will often identify φ with the set of atomic formulas in it, writing $A_1(\boldsymbol{x}_1), \ldots, A_n(\boldsymbol{x}_n)$ instead of $A_1(\boldsymbol{x}_1) \wedge \ldots \wedge A_n(\boldsymbol{x}_n)$. A variable that occurs in the head of rule r is a *free variable* of r. Other variables are *bound* in r. We require that every free variable occurs in the body. (iii) A distinguished predicate *Goal* of S which occurs in the head of a rule, referred to as the *goal predicate*.

The relational symbols that do not occur in the head of any rule are the *input* or *extensional predicates*, while the others are *intensional predicates*. Monadic datalog (MDL) denotes the sublanguage where all intensional predicates are monadic (unary). For a datalog program Q, we denote by $\mathrm{Const}(Q)$ the constants used in Q.

Semantics. For a datalog program, an intensional predicate P, a structure D interpreting the input predicates and constants, we define the output of P in D, denoted $P(D)$, as the fixpoint of the relations $P_i(D)$ defined via the following process, starting with $P_0(D) = \emptyset$:

- Let D^i be the expansion of D with $P_i(D)$ for all intensional P.
- If r is a rule with $P(x_1 \ldots x_l)$ in the head, $\boldsymbol{w}$ the bound variables of r, and $\varphi(\boldsymbol{x}, \boldsymbol{w})$ the body of r, let $P^r_{i+1}(D)$ be defined by: $\{\boldsymbol{c} \in \mathrm{dom}(D)^l : D^i \models \exists \boldsymbol{w}\, \varphi(\boldsymbol{c}, \boldsymbol{w})\}$ where $\mathrm{dom}(D)$ is the active domain of D.

We let $P_{i+1}(D)$ denote the union of $P^r_{i+1}(D)$ over all r with P in the head. Finally, the *query result* of Q on D is the evaluation of the goal predicate of Q on D. We will often assume $Goal()$ is nullary, in which case the result of the query on D is the Boolean true iff $Goal()$ holds in D. We will alternatively refer to a *datalog query* to emphasize that we are only interested in the output on the goal predicate. Note that, unlike in many works on datalog, we allow constants. Under these semantics, it is easy to check that any UCQ can be transformed in linear-time into an equivalent MDL query.

Containment. The main problem we deal with in this work is the classical notion of query containment.

Definition 1. *Let Q and Q' be two queries over a relational structure S. We say Q is* contained *in Q', denoted $Q \sqsubseteq Q'$, if, for any database D over S, $Q(D) \subseteq Q'(D)$.*

We focus on containment for Boolean queries in the remainder of the paper. However, our results apply to the unary case as well, thanks to the following:

Proposition 1. *There are polynomial-time one-to-one reductions between the containment of Boolean MDL queries in Boolean UCQs and that of MDL queries in UCQs. A similar statement holds for positive queries.*

The direction from non-Boolean to Boolean in the proposition above will imply that our lower bounds apply to unary MDL as well. The other direction will

be used to transfer upper bounds. Our upper bounds will be proven for several restricted classes of Boolean MDL. The analogous definitions for non-Boolean MDL will be obvious, and all of these classes will easily be seen to be preserved by the transformation from above. Hence the argument above implies that these upper bounds also apply to the corresponding non-Boolean problem.

Note that this simple argument does not apply to the results of Chaudhuri and Vardi [7] on containment of connected unary MDL queries into union of unary connected queries since connectedness is not preserved by the reduction.

Tree-Like Instances. Before discussing the complexity of the containment problem, we introduce the notion of *tree-like instance* that will be used in various proofs throughout the paper.

A *tree-like instance* for an MDL query Q over S is an instance I for the relations of S, plus a tree $\mathrm{T}(I)$ whose nodes n are associated with both a value v appearing in I and a collection of atoms of I, called the *bag* of n and denoted $\mathrm{bag}(n)$. We say v is the *output value* of the node n and we require that:

1. all values appearing in atoms of $\mathrm{bag}(n)$ are either in $\mathrm{Const}(Q)$, are output values of n, or are output values of one of n's children in the tree;
2. for every value $v \notin \mathrm{Const}(Q)$ that appears in I, v is the output value of exactly one node n, and any atoms containing v must be in the bag of n or its parent.

We denote a node n by a pair $(v, \mathrm{bag}(n))$ of its output value and bag of facts.

A fundamental fact is that there is always a tree-like witness for non-containment of a monadic datalog query in a UCQ. A similar result appears in various other places in the literature, such as [7–9, 11], but we will require a version of this result valid also in the presence of constants. Note that tree-like instances are a special kind of tree-decomposition, with additional structure.

Proposition 2. *If an MDL query Q is not contained in a UCQ Q', then there is a tree-like instance I such that I satisfies $Q \wedge \neg Q'$. Furthermore, the size of the bags of I can be taken to be polynomial in the size of Q and Q'.*

3 Monadic Datalog Containment in General

We first note that containment of monadic datalog queries in positive queries is known to be in 2EXPTIME: in the absence of constants, this follows from results of ten Cate and Segoufin [12], who define a language closed under negation that subsumes both MDL and PQs. This also holds in the presence of constants:

Theorem 1 (Proposition 4.11 of [13]). *The problem of containment of monadic datalog (with constants) in PQs (with constants) is in* 2EXPTIME.

Indeed, [13] shows the same bound for containment of datalog in positive queries with constants. We now show the first main result of this article, a corresponding lower bound of 2EXPTIME-hardness holding even in the absence of constants.

Theorem 2. *Containment of monadic datalog in UCQs is* 2EXPTIME*-hard, even for connected Boolean queries. Hence containment of MDL in MDL is* 2EXPTIME*-hard.*

The proof proceeds by creating a Datalog program Q and UCQ Q' such that models of $Q \wedge \neg Q'$ code the computation of an alternating EXPSPACE Turing Machine. The proof relies on a coding trick developed by Björklund, Martens, and Schwentick in the context of trees [14]. The trick allows one to check, given two paths coding an n-length bit string, whether they code the same string. Under a naive coding one needs a positive query to do this, checking a conjunction of disjunctions; in the Björklund et. al. coding, one can use a UCQ to do the check. The coding of the machine uses a datalog program of a very restricted form: extensional relations have arity at most 2, rule bodies are acyclic, and the program is nonpersistent [7]. Note that the co-NEXPTIME membership result of [7] of nonpersistent datalog programs in unions of connected queries only holds for non-Boolean queries.

4 Reducing the Complexity

To lower the complexity, we consider a restriction on the form of MDL queries that bounds the use of extensional predicates across all rules. As we will show, this restriction is relevant to the problem of containment under limited access patterns [8, 9]. Enforcing this bound will suffice to ensure the existence of witnesses of non-containment in any UCQ of size at most exponential, which easily shows that the complexity reduces to co-NEXPTIME. Furthermore, if constants are forbidden, we will show the problem is in EXPTIME. In addition of giving tight complexity bounds for this restricted language, we also provide lemmas about the structure of witnesses to non-containment, that are of interest in themselves and can serve to prove similar complexity results in other settings.

4.1 Equivalence of Nodes in Tree-Like Instances

Before considering our restricted language, we revisit the argument showing the 2EXPTIME upper bound of containment of monadic datalog in unions of conjunctive queries. This result is proved in [4, 5] in the absence of constants, and is generalized by Theorem 1.

Note that *in all of the discussion of upper bounds in this paper, we usually reason with Q' being a single CQ, but then state results for UCQs.* The extension of the upper bound techniques and results to UCQs will always be straightforward.

By Proposition 2 in considering containment we can restrict ourselves to tree-like instances. Then we introduce a notion of "special subquery" which is used to define an equivalence relation on the nodes of a tree like instance, called *IQ-equivalence*. Finally, we show that the tree-like counterexamples to containment are given by a collection of tree automata whose states are IQ equivalence classes. This allows us to relate the time complexity of finding a counterexample by the number of IQ equivalence classes. We show how this gives a generic recipe for getting bounds on the containment problem.

IQ Classes. How big does a tree-like counterexample to containment have to be? Intuitively, one only needs nodes that represent the different kinds of behavior with respect to Q', since two nodes with the same behavior can be collapsed. A crude notion of "same behavior w.r.t. Q'" would be to identify a node with the collection of subqueries, substituted with constants, that simultaneously hold at that node. Such an abstraction would easily lead to a doubly-exponential bound on the size of a counterexample to containment. Our main contribution is a finer notion of similarity that takes advantage of the restricted structure of tree-like instances. Intuitively, we do not care about all subqueries that map to a node, but only about the way that the subqueries impact what is happening at other nodes. That interaction is captured by the restriction of the mapping to the root, and by the variables that are mapped to constants, since constants cut across the tree structure. We can thus think of fixing both a mapping to constants (derived subquery below) and a root interface. Naïvely, our root interface would require us to specify the mapping to the root bag completely. Instead we allow ourselves to fix only two things: 1) the set of variables that map to the output value of the root; 2) for each single atom at a time, and each connected component in the query obtained by removing this atom, information about whether the corresponding query is *mappable*. We formalize this idea of "mappable interface queries" below.

A *derived query* of Q' is one obtained by substituting variables with constants (where multiple variables can map to the same constant).

Given a query Q', an atom A of Q', and a subset X of the variables of Q', a subquery Q'' of Q' is *fresh-connected for X and A* iff A occurs in Q'' and:

- (closure) Q'' is closed under variables not in X – if it contains one atom with such a variable x, it contains all atoms containing x.
- (connectedness) for every variable x of Q'' not in X there is a path from x to a variable in atom A, where the path is in the graph connecting two variables if they co-occur in an atom of Q'';
- (fresh witnesses) every atom in Q'' other than A must contain a variable outside of X.

A query Q'' and an atom A are *pointed mapped* relative to a subset X of its variables and a node n in a tree-like instance iff there exists a homomorphism h from Q'' to the subtree rooted at n such that the set of variables mapping to the output value of n is equal to X, no variable of Q'' is mapped to a query constant, and $h(A) \in \text{bag}(n)$. Thus variables outside of X map to values that are *fresh* for n – neither a query constant nor the output value of n.

We define an equivalence relation on nodes n in tree-like instances.

Two nodes $n = (v, b)$ and $n' = (v', b')$ are *IQ-equivalent* (short for "Interface Query equivalent") with respect to a set C of constants iff

- either a) v is in C and $v' = v$; or b) neither v nor v' is in C;
- for each atom A and derived subquery Q'' of Q', for each subset X of the variables of Q'', for each fresh-connected subquery Q''' of Q'' for X and A, Q''' is pointed mapped to the subtree rooted at n relative to X iff Q''' is pointed mapped to the subtree rooted at n' relative to X.

Composition of IQ Classes. Let C be the constants occurring in the MDL query Q and in the CQ Q'. Let I be a tree-like instance, n and n' two nodes of $\mathrm{T}(I)$ with respective sets of children $\{n_1, \ldots n_k\}$ and $\{n'_1 \ldots n'_{k'}\}$. An isomorphism φ from the values of $\mathrm{bag}(n)$ to the values of $\mathrm{bag}(n')$ is a *C-isomorphism* iff it preserves constants of C and the output value of n maps to the output value of n'.

We can now state our composition lemma: in the presence of a C-isomorphism, IQ-equivalence of the children of two nodes implies IQ-equivalence of the nodes.

Lemma 1. *Let I be a tree-like instance. Let n, $n_1 \ldots n_k$ be a node of I and its children, and n', $n'_1 \ldots n'_k$ a node of I and its children. Suppose n_i is IQ-equivalent to n'_i w.r.t. constants C for $1 \leqslant i \leqslant k$ and that there exists a C-isomorphism φ from $\mathrm{bag}(n)$ to $\mathrm{bag}(n')$ such that if the output value of n_i is in domain(φ), then $\varphi(n_i)$ is the output value of n'_i, and vice versa. Then n and n' are IQ-equivalent w.r.t. to C.*

From IQ Classes to Automata. It is easy to use the composition lemma to show that given a counterexample to containment of MDL Q in UCQ Q', another counterexample can be obtained where any non-root nodes have distinct IQ classes – i.e., one can bound the size of a counterexample instance for containment by the number of IQ classes. In fact, IQ classes capture the "state" of a node in the tree skeleton in a literal sense: they can be used as states in a tree automaton that accepts such skeletons. Let $B(C, d, S)$ be the set of C-isomorphism types of bags with d atoms over schema S. Then every tree-like instance I is associated with a $B(C, d, S)$-labeled (ranked) tree $\mathrm{Code}(I)$. Note that the C-isomorphism type indicates which value is the output value.

Lemma 2. *For each C, d, S, and CQ Q', there is a non-deterministic bottom-up tree automaton $A = \bigcup_{1 \leqslant i \leqslant k} A_i$ that accepts the set of all $\mathrm{Code}(I)$ with I a tree-like instance not satisfying Q', where k is bounded by a polynomial in the number of subsets of connected subqueries of derived queries of Q', and the size of each A_i is bounded by a polynomial in the number of IQ classes of Q'.*

How does this help us? We can intersect the automaton A above with an automaton restricting the shape of tree-like instances: e.g., to tree-like models of an MDL query Q, or special tree-like models. Using the fact that non-empty automata have examples with DAG representations polynomial in the automaton size, we see that whenever containment fails, there is a counterexample of size at most the maximum of $|A_i|$, which is bounded by the number of IQ classes. This gives a non-deterministic algorithm for guessing a counterexample to containment. But the result also gives a deterministic time bound in the size of A. In the general case of MDL containment in UCQs, the number of IQ classes is doubly-exponential, and the number k above is also doubly-exponential, and so the automata-theoretic approach gives the same bound as in Chaudhuri and Vardi [5]. As we have seen in the previous section, this is the best we can do, since the bound is tight. We can do better if we can restrict to a class of instances that satisfies the Unique Mapping Condition:

Definition 2. *A class of tree-like instances satisfies the Unique Mapping Condition (UMC) if for any node n, any conjunctive query Q', and any atom A there*

exists at most one triple Q'', X, and Q''' such that: Q'' is a derived subquery of Q', X is a subset of the variables, Q''' is fresh-connected for X, A, and there is a pointed map of Q''' to n for X.

The UMC should shed more light on the conditions of fresh-connected queries: it is easy to see that for a disconnected query we will not have a unique mapping even for very restricted structures; and if we do not look at connected queries that are somehow "maximal," we cannot get uniqueness. We will show that the Unique Mapping Condition suffices to get better upper bounds. The following is easy to see:

Proposition 3. *Restricted to a class of instances where the UMC applies, the number of IQ classes is exponential. Thus satisfiability of UCQs restricted to such a class is in co-*NEXPTIME.

Proof. By the UMC, the number of classes is bounded by the number of functions from atoms to triples (Q'', X, Q'''), which is exponential in Q'. The existence of an exponential-sized bound on counterexample size follows using Lemma 2. □

If the queries are required not to have constants, the problem moves from NEXPTIME to EXPTIME.

Theorem 3. *Suppose we have a class C of instances that has the UMC, has no constants, and whose codes can be described by a tree automaton of size at most exponential. Then the problem of determining whether a member of C satisfies the negation of a* UCQ *is in* EXPTIME.

Proof. The number of IQ classes is exponential in Q'. Due to the absence of constants, there exists only one derived query from Q', which is Q' itself. So the number of connected subqueries of Q' is polynomial in Q'. This in turn implies that the number of subsets is exponential in the size of Q'. The rest follows from Lemma 2 and the fact that emptiness of tree-automata is in PTIME. □

4.2 Global Restrictions and Diversified Instances

We are now ready to get an improved upper bound. We first look at arbitrary instances, but put a strong restriction on queries.

A monadic datalog query, possibly with constants, is globally extensionally restricted (GEMDL) if every extensional predicate appears in only one rule, and occurs only once in that rule. An MDL query is almost globally extensionally restricted (AGEMDL) if the goal predicate never occurs in the body of a rule, and every extensional predicate has only one occurrence in a rule other than a rule for the goal predicate.

Informally, AGEMDL queries allow UCQs built over intensional predicates, where extensional predicates are partitioned into classes where each rule uses predicates in a particular class.

An interesting example of AGEMDL comes from the area of *limited-access querying* [15]. In limited-access querying, one has a schema where each relation of arity l comes with a set of access methods, each with a subset of $[1 \dots l]$, the *input positions*. In an instance of the database, the accessible data is limited to

what can be extracted starting with a set of seed constants, iteratively putting the values in for input positions and getting the resulting output [16]. Restrict for now to access methods having at most one output position. Formally, we can consider the monadic datalog program with rules

$$\begin{cases} \text{Accessible}(c) \leftarrow & \text{where } c \text{ is any constant in the seed set} \\ \text{Accessible}(x_{j_k}) \leftarrow R(\boldsymbol{x}), \text{Accessible}(x_{j_1}), \ldots, \text{Accessible}(x_{j_p}) \end{cases}$$

whenever relation l has an access method with input positions $j_1 \ldots j_p$ and output position j_k. Given an instance I and set of constants C, we let $\text{Accessible}(I, C)$ be the output of Accessible where C is the set of constants.

Given queries Q and Q' that use constants C the problem of query containment of Q in Q' relative to access patterns asks whether Q will always return a subset of Q' when applied to tuples that are accessible. That is, it asks whether the set $\{\boldsymbol{c} \mid \boldsymbol{c} \in Q(I) \wedge \bigwedge_i c_i \in \text{Accessible}(I, C)\}$ is contained in $\{\boldsymbol{c} \mid \boldsymbol{c} \in Q'(I)\}$.

Consider the case where every relation has only one access method. This is the standard assumption in much of the literature [17]. The rules above then satisfy the GEMDL restriction. We can thus express the evaluation of UCQ Q over accessible data using a single additional goal predicate, expressing Q and the restriction that all witness variables are accessible. This will be an AGEMDL query. It follows that the question of *containment of unions of conjunctive queries under limited access patterns* (see [9,17] for the formal definition) can be expressed as the containment of a AGEMDL query in a UCQ. We now claim:

Theorem 4. *The containment problem for AGEMDL queries in union of conjunctive queries (with constants) is in co-*NEXPTIME *(and is co-*NEXPTIME*-complete).*

From this and the observation above we have the following result, also shown in [9]:

Corollary 1. *Containment of UCQ queries with constants under limited access patterns, where every relation has only a single access method and every access method has at most one output position, is complete for co-*NEXPTIME*.*

The technique can also be used for access methods with arbitrary numbers of positions, but this requires extending the definition of AGEMDL to allow rules with multiple head predicates, with the restriction that these rules need to be nonpersistent [7].

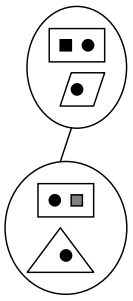

Fig. 1. Diversified tree-like instance

We now begin the proof of the upper bound in Theorem 4. The AGEMDL restriction implies a stronger kind of tree-like instance. A *diversified tree-like instance* is a tree-like instance in which: (i) for each node n which is not the root, for each relation R, there exists at most one fact in $\text{bag}(n)$ having the relation name R; (ii) we cannot have a fresh value v (a value that is not a constant or the output value of the root) such that there are two distinct facts in the instance with the same relation name which have v in the same position.

The first condition is a local one, saying we do not have self-joins within a bag, while the second one is global, saying self-joins across bags must not have the joined variable in the same position. A figure showing a diversified tree-like instance is given in Figure 1. The large ellipses represent bags, with different relations represented by different shapes in a bag. The dark disc represents a common value shared across and within bags.

We now have the following refinement of Lemma 2.

Proposition 4. *If a AGEMDL query Q is not contained in UCQ Q', then there is a diversified tree-like instance I such that I satisfies $Q \wedge \neg Q'$ and for each n of I, the size of its bag is polynomial in Q.*

The following links diversified instances to the IQ-class machinery:

Theorem 5. *The class of diversified tree-like instances satisfies the UMC.*

Note that X and Q'' always determine a single Q''', so the key is that, in this restricted setting, we have only one way to map variables to constants, and only one way to select which variables map to the output value of the root. The idea is that if we had a choice of how to map an atom, the choice would be between either atoms in the same bag (forbidden by the first condition of diversified instances) or atoms in neighboring bags (forbidden by the second condition).

From the result above, Lemma 3 and Theorem 3 we get upper bounds for AGEMDL; we also have matching lower bounds:

Corollary 2. *The containment of an AGEMDL query with constants in a UCQ with constants can be decided in co-*NEXPTIME*. In the absence of constants the problem is in* EXPTIME*.*

Theorem 6. *The problem of determining whether an* AGEMDL *query is contained in a* UCQ *is* EXPTIME*-hard, and with constants the problem is co-*NEXPTIME*-hard.*

The co-NEXPTIME argument is by reduction to a prior result in [9]. The EXPTIME-hardness is proven via an encoding of the execution of a PSPACE alternating Turing machine. The coding is simpler than for Theorem 2, since a configuration can be directly coded as a polynomial-sized vertical chain in a tree.

AGEMDL forbids a relation from occurring more than once outside of the goal predicate, which is a strong restriction. A simple generalization is to consider the class of MDL queries where relations can occur in a bounded number of rules. Let k-GEMDL be the class obtained by replacing "in only one rule", with "in at most k rules" in the definition of GEMDL (while still restricting to one occurrence per rule). In this case, the complexity jumps back up to 2EXPTIME.

Theorem 7. *There is a k such that the problem of k-GEMDL containment in a UCQ is* 2EXPTIME*-hard.*

The proof is quite involved. The main challenge, compared to that of Theorem 2, is that we cannot use a generic *Child* relation to code the tree structure of an alternating EXPSPACE computation, since this would require two occurrences of the same atom in one rule. We also cannot use separate predicates for left and

Table 1. Complexity of containment of monadic datalog queries in UCQs and PQs. In all cases, problems are complete in the class shown.

	UCQ	**PQ**
MDL	2EXPTIME	2EXPTIME
MDL with constants	2EXPTIME	2EXPTIME
GEMDL	EXPTIME	2EXPTIME
GEMDL with constants	co-NEXPTIME	2EXPTIME
k-GEMDL with constants	2EXPTIME	2EXPTIME

right child to code the structure: this would make it impossible to use a UCQ to compare paths of the trees that represent addresses of cells. Thus we develop a new coding relying on a combination of a linear structure and "value joins".

We also note that, for positive queries, the GEMDL restriction does not help:

Theorem 8. *GEMDL containment in a positive query is* 2EXPTIME*-hard.*

The lower bound proof uses a simple coding of an alternating EXPSPACE computation, using the power of positive queries to compare paths.

A diversified tree-like instance still allows joins. What about "pure tree structures" – tree-like instances representing labeled unranked trees with the child relation? These are not a special case of diversified tree-like instances, since one cannot have all child facts of a given vertex in the same bag. Still, it is not surprising that the UMC also applies to pure trees:

Theorem 9. *Tree structures have the UMC.*

From the theorem above and the small model machinery presented earlier, we get the following new bound, which answers a question left open from [14].

Corollary 3. *Containment of a tree automaton in a UCQ is in* EXPTIME.

The above results complements a recent theorem of Björklund et al. [18], which shows that the above problem is hard for EXPTIME.

5 Conclusions and Related Work

In this paper we have revisited the containment problem for recursive queries in UCQs. We started by showing that the problem is hard for doubly-exponential time in general. We then analyzed the phenomenon of tree-like models for recursive queries in more detail, and give a parameter – the number of IQ classes – that controls the size of a minimal counterexample to containment. We have shown that if a logic has models that are "very tree-like", then the number of distinct ways a CQ can map into the model is limited, and thus an exponential bound can be shown on the number of IQ classes. We have applied this analysis to two logics and two collections of instances – GEMDL and tree automata. But we believe that it can be applied to other fragments of MDL.

Our results on containment of MDL are summarized in Table 1.

Special cases of the containment problem of monadic datalog in UCQs have been studied in the past. The original Chaudhuri and Vardi article [7] proved a co-NEXPTIME bound of containment of connected unary MDL queries in UCQs, where the queries were unary. Their paper extends earlier work by Courcelle [11], who noted the connection with graph decompositions.

Segoufin and ten Cate [12] define the language UNFP which can express the conjunction of an MDL query with a negated UCQ; they show that the satisfiability for this language is 2EXPTIME-complete. An EXPTIME bound on satisfiability is shown for a fragment called Simple UNFP; the fragment cannot express UCQs, much less the negation of a UCQ conjoined with an MDL query. Björklund et al. [14] study containment of tree automata in UCQs with child and descendant. We make use of their lower bound technique in our first result, while also refining one of their upper bounds in the absence of descendant.

Our main upper bound technique originates from our work on limited access querying [9] – there we showed a co-NEXPTIME bound for a particular kind of MDL/UCQ containment problem, using a special case of the technique. Our upper bounds here are an abstraction of the idea in [9], relating it to tree-like instances. Our lower bounds can be seen as exploring the limits of this method. Note that [9] also contains errors that are discussed (and in one case, corrected) in [13] – see the end of the related work section of [13].

Acknowledgments. Bourhis is supported by EPSRC EP/G004021/1 (the Engineering and Physical Sciences Research Council, UK). Senellart is partly supported by ERC grant Webdam, agreement 226513.

References

1. Shmueli, O.: Equivalence of datalog queries is undecidable. J. Log. Prog. 15(3) (1993)
2. Bonatti, P.A.: On the decidability of containment of recursive datalog queries. In: PODS (2004)
3. Calvanese, D., De Giacomo, G., Vardi, M.Y.: Decidable containment of recursive queries. Theoretical Computer Science 336(1) (2005)
4. Chaudhuri, S., Vardi, M.Y.: On the equivalence of recursive and nonrecursive Datalog programs. In: PODS (1992)
5. Chaudhuri, S., Vardi, M.Y.: On the equivalence of recursive and nonrecursive Datalog programs. JCSS 54(1) (1997)
6. Cosmadakis, S.S., Gaifman, H., Kanellakis, P.C., Vardi, M.Y.: Decidable optimization problems for database logic programs. In: STOC (1988)
7. Chaudhuri, S., Vardi, M.Y.: On the complexity of equivalence between recursive and nonrecursive Datalog programs. In: PODS (1994)
8. Calì, A., Martinenghi, D.: Conjunctive Query Containment under Access Limitations. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 326–340. Springer, Heidelberg (2008)
9. Benedikt, M., Gottlob, G., Senellart, P.: Determining relevance of accesses at runtime. In: PODS (2011)
10. Calvanese, D., Giacomo, G.D., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: KR (2000)

11. Courcelle, B.: Recursive queries and context-free graph grammars. Theoretical Computer Science 78(1) (1991)
12. ten Cate, B., Segoufin, L.: Unary negation. In: STACS (2011)
13. Benedikt, M., Bourhis, P., Ley, C.: Querying schemas with access paths. PVLDB (2012)
14. Björklund, H., Martens, W., Schwentick, T.: Optimizing Conjunctive Queries over Trees Using Schema Information. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 132–143. Springer, Heidelberg (2008)
15. Rajaraman, A., Sagiv, Y., Ullman, J.D.: Answering queries using templates with binding patterns. In: PODS (1995)
16. Li, C., Chang, E.Y.: Answering queries with useful bindings. ACM TODS 26(3) (2001)
17. Calì, A., Martinenghi, D.: Querying the deep web. In: EDBT (2010)
18. Björklund, H., Martens, W., Schwentick, T.: Validity of tree pattern queries with respect to schema information (2012) (unpublished draft)