Monadic Datalog, Tree Validity, and Limited Access Containment

Michael Benedikt, Pierre Bourhis, Georg Gottlob, Pierre Senellart

Monadic Datalog, Tree Validity, and Limited Access Containment

MICHAEL BENEDIKT, The University of Oxford, United Kingdom
PIERRE BOURHIS, CRIStAL, CNRS, University of Lille & Inria, France
GEORG GOTTLOB, The University of Oxford, United Kingdom
PIERRE SENELLART, DI ENS, ENS, CNRS, PSL University & Inria & LTCI, Télécom Paris, IP Paris, France

We reconsider the problem of containment of monadic datalog (MDL) queries in unions of conjunctive queries (UCQs). Prior work has dealt with special cases of the problem, but has left the precise complexity characterization open. In addition, the complexity of one important special case, that of containment under access patterns, was not known before. We start by revisiting the connection between MDL/UCQ containment and containment problems involving regular tree languages. We then present a general approach for getting tighter bounds on the complexity of query containment, based on analysis of the number of mappings of queries into tree-like instances. We give two applications of the machinery. We first give an important special case of the MDL/UCQ containment problem that is in EXPTIME, and use this bound to show an EXPTIME bound on containment under access patterns. Secondly we show that the same technique can be used to get a new tight upper bound for containment of tree automata in UCQs. We finally show that the new MDL/UCQ upper bounds are tight. We establish a 2EXPTIME lower bound on the MDL/UCQ containment problem, resolving an open problem from the early 1990s. This bound holds for the MDL/CQ containment problem as well. We also show that changes to the conditions given in our special cases can not be eliminated, and that in particular slight variations of the problem of containment under access patterns become 2EXPTIME-complete.

Additional Key Words and Phrases: Access patterns, binding patterns, deep Web, monadic datalog, query containment

ACM Reference Format:
Michael Benedikt, Pierre Bourhis, Georg Gottlob, and Pierre Senellart. 2019. Monadic Datalog, Tree Validity, and Limited Access Containment. *ACM Trans. Comput. Logic* 21, 1, Article 6 (October 2019), 51 pages. https://doi.org/10.1145/3344514

1 INTRODUCTION

Context. Datalog represents a standard model for querying data with recursion. The basic problems of evaluation, equivalence, and containment of datalog thus have been the object of study for several decades. Shmueli [31] showed that containment (and equivalence) of datalog is undecidable. Decidable subclasses were subsequently isolated [11, 14], focusing on restricting the form of recursion used. Chaudhuri and Vardi [17, 19] provide an extensive study of the containment of datalog queries in non-recursive 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 [22] 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 [18] 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 *connected*

Authors' addresses: Michael Benedikt, The University of Oxford, Oxford, United Kingdom; Pierre Bourhis, CRIStAL, CNRS, University of Lille & Inria, Lille, France; Georg Gottlob, The University of Oxford, Oxford, United Kingdom; Pierre Senellart, DI ENS, ENS, CNRS, PSL University & Inria & LTCI, Télécom Paris, IP Paris, Paris, France.

2019. 1529-3785/2019/10-ART6
https://doi.org/10.1145/3344514

unary 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 Boolean conjunctive queries.

Since the work of Chaudhuri and Vardi, the "fine structure" of the containment problem between recursive and non-recursive queries has remained mysterious. What computation can one code up in an MDL/UCQ containment problem (or, even, in an MDL/CQ containment problem)? What features of recursive queries make the containment problem hard? A better understanding of these issues can shed light on a number of other questions, e.g., containment of UC2RPQs [15] in non-recursive queries. The problem gained additional importance from the question of querying under limited access patterns [6, 13]: a schema with limited access patterns specifies that certain relations can only be accessed via lookups, providing values for a fixed set of input positions. The input values cannot be guessed, but instead must either be in a particular set of known "seed values", or be the outputs of earlier accesses. Thus, given any instance I of the schema, there is a corresponding subset of the facts in the instance, denoted here by $\mathsf{AccFacts}(I)$, that can be obtained via use of the accesses.

Example 1.1. Consider a schema with relations as follows: there is a binary relation $R(x, y)$, having an access a_R that requires a value for both positions; given inputs v, w the access returns true exactly when $R(v, w)$. There is also a unary relation $U(x)$ having an access a_U that requires no input: hence a call to a_U will return all values of U. The accessible facts from this schema can be seen to be all facts of the form $U(c)$ and all facts $R(c, d)$ where $c, d \in U$.

The containment problem for limited access patterns is the following: given two queries Q and Q' and a schema with limited access patterns, is it true that, for every instance I, $Q(\mathsf{AccFacts}(I)) \subseteq Q'(\mathsf{AccFacts}(I))$?

Example 1.2. In the example schema above, consider queries $Q = \exists x \exists y\ R(x, y)$ and $Q' = \exists x \exists y\ R(x, y) \wedge U(x)$. These are certainly not equivalent, but are equivalent over the accessible data for this schema: indeed, the only way to access facts from relation R is to first access U, implying that all facts of the form $R(a, b)$ that are accessible are such that both $U(a)$ and $U(b)$ are accessible.

It was noticed early on [27] that the problem of containment of unions of conjunctive queries under access patterns is a special case of monadic datalog containment in a UCQ. First, the problem can be seen to be equivalent to:

$$\forall I\ Q(\mathsf{AccFacts}(I)) \subseteq Q'(I)$$

One direction of the equivalence is immediate from monotonicity of UCQs, since $Q'(\mathsf{AccFacts}(I)) \subseteq Q'(I)$. For the other direction, given a counterexample to containment under access patterns, we can modify it by replacing I with $\mathsf{AccFacts}(I)$, giving a counterexample to the above.

Assuming each relation mentioned in the query has at least one access method (this is without loss of generality, since containment is trivial otherwise), the query $Q'(\mathsf{AccFacts}(I))$ can be defined by restricting Q' to the values that occur in $\mathsf{AccFacts}(I)$, and the latter can in turn be defined by an MDL program, using the rules:

$$\mathsf{AccValues}(x_j) \leftarrow R(\mathbf{x}) \wedge \mathsf{AccValues}(x_{m_1}) \wedge \cdots \wedge \mathsf{AccValues}(x_{m_k})$$

whenever there is an access with input positions $m_1 \ldots m_k$.

Thus *containment under access limitations is a special case of MDL/UCQ containment.* But again the only upper bound observed for containment under access patterns was the 2EXPTIME bound of Chaudhuri and Vardi.

Another problem closely related to the MDL/UCQ containment problem is *boundedness* of MDL, i.e., determining whether the evaluation of a MDL program reaches a fixpoint after a fixed, data-independent, number of iterations. Though boundedness of Datalog programs is undecidable in

general, it was shown in [22] that it is decidable for MDL, though the precise complexity was not assessed. As shown in [7] (Claim 40 of the extended version), containment of MDL in a UCQ reduces to MDL boundedness.

Contributions. In this work, we will ascertain both the exact complexity of MDL/UCQ containment, and the exact complexity of limited-access containment. We will also settle some questions on the complexity of *query validity problems on trees*: this is the problem of determining, given a tree schema (e.g. given as an automaton or DTD) and a query, whether the schema implies the query. We will show that monadic datalog containment and tree validity problems are tightly connected.

We start by introducing a new upper bound technique, revisiting the 2EXPTIME upper bound for containment of Chaudhuri and Vardi. We will refine 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 EXPTIME.

We give two settings where the Unique Mapping Condition holds. The first is that of "almost globally extensionally limited" MDL (AGEMDL) queries over general relational instances: we limit the number of occurrences of an extensional predicate in the program. We show that containment under limited access in the case of a single-access per relation reduces to GEMDL containment in a UCQ. Thus our EXPTIME bounds apply to this case. We show that this reduction can be bootstrapped to give the same bounds for the general limited-access containment problem.

A second application is when the models are trees. We show that the UMC for this case gives us new upper bounds on the validity problem of tree automata in UCQs.

We then turn to lower bounds. We first show that all of our upper bounds on tree validity are tight. The key ingredient is an adaptation of a 2EXPTIME-hardness argument due to Bjorklund, Martens and Schwentick [8–10]. We then give reductions from tree validity problems to MDL and limited access containment problems to show that the upper bounds we provide for these problems are tight as well. In particular, we show that containment of MDL in CQs (and hence, UCQs) is 2EXPTIME-complete. This resolves the main open question from [18] and also implies that MDL boundedness is 2EXPTIME-hard, as noted in [7]. Since containment of MDL in a CQ is a special case of monadic datalog containment, it also shows that monadic datalog containment is 2EXPTIME-complete, closing the gap in the bounds of [22]. We finally use the technique to show that when we move to a slightly broader collection of MDL programs, compared to those arising from limited access containment, we get a 2EXPTIME-complete query containment problem.

Organization. Section 2 contains the basic definitions. Section 3 reviews prior work and summarizes the main results in detail. Section 4 defines the main new subclasses for which tighter upper bounds can be provided, and then gives our new upper bound proofs. Section 5 deals with lower bounds, showing that all previously given upper bounds are tight. Section 6 discusses related work, while Section 7 gives conclusions, open issues, and future directions.

Limitations. For simplicity, we do not consider two important variants of the MDL/UCQ containment problem and of limited-access containment. First, we assume that all queries do not contain any constants, i.e., that all relation atoms are made up of variables. Though the general MDL/UCQ problem is known to also be in 2EXPTIME in the presence of constants [3, 4], constants typically do affect the complexity of containment problems, as they make it easier to code hardness proofs. Second, contrarily to some of the limited-access literature [6, 13], we assume that our queries work over relational databases in which there are no restrictions on the values that occur in a column. The alternative would be to allow a schema to specify a *type* or *abstract domain* for each attribute.

Though we do not consider abstract domains, the corresponding constraints could easily be coded by a disjunction of conjuncts, so we believe our results on limited-access containment of UCQs would not be affected.

2 BACKGROUND

We introduce in this section the main notions studied in this article (some from the literature, others introduced in this work), along with some general preliminary results. We go over the following concepts: monadic datalog and query containment (Section 2.1); expansion trees of monadic datalog queries (Section 2.2); tree validity problems (Section 2.3); querying under limited access constraints (Section 2.4).

2.1 Monadic Datalog Containment

Basics. A *relational signature* σ consists of a set of *relation symbols* (or simply *relations*), each with an associated arity. For a relation R any number $i < \text{arity}(R)$ is referred to as a *position* of R. An *instance* I for the signature σ interprets each relation symbol in the signature of arity k by a set of k-tuples, where the values are taken in some infinite set of values (independent of the instance). For every tuple $(v_1, \ldots, v_n)$ in the interpretation of a relation symbol R by I, we say that $R(v_1, \ldots, v_n)$ is a *fact* of I. We denote by $\text{dom}(I)$ the *active domain* of the instance I, i.e., the set of values appearing in I.

An *atom* over a relational signature σ is an expression $R(x_1 \ldots x_n)$ where R is a k-ary predicate of σ and the x_i's are variables (from some countable set). We refer to x_i as "the variable at position i" of the atom. We emphasize that an atom never refers to a domain constant.

We consider the following two simple positive query languages over σ: (a) conjunctive queries (CQs); (b) unions of conjunctive queries (UCQs). They are respectively defined as the fragments of first-order logic over σ consisting of, respectively, (a) existentially quantified conjunctions of atoms; (b) disjunctions of CQs. A conjunctive query is *connected* if its Gaifman graph (whose vertices are the variables of the query and where there is an edge between two variable if they appear in the same atom) is connected. A *subquery* of a CQ is a subset of its atoms; a subquery of a UCQ is a subquery of one of its conjuncts (in particular, a subquery is always a CQ).

Datalog. A *datalog program* [1] over σ consists of:

(i) A set of rules of the form $A \leftarrow \varphi$, where φ is a conjunction of atoms over σ, and A is an atom over σ. We say A is the *head* and φ the *body* of the rule. We require that every variable occurring in the head of a rule r also occurs in its body.

(ii) A distinguished predicate Goal of σ 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*. Similarly, the *extensional* (resp., *intensional*) signature of a program is the set of *extensional* (resp., *intensional*) predicates used by the program. Monadic datalog (MDL) denotes the sublanguage where all intensional predicates are monadic (unary), except for the goal predicate which can be either unary or nullary (in the latter case, we say that the program is *Boolean*).

For a datalog program P, an intensional predicate R of P, and an instance I interpreting the input predicates, we define the evaluation of R on I, denoted $R(I)$, as the union of the relations $P_k(R, I)$ defined via the following process, starting with $P_0(R, I) = \emptyset$:

- Let I^k be the expansion of I with $P_k(R, I)$ for all intensional R.

- If r is a rule with $R(x_1 \dots x_l)$ in the head, $\mathbf{w}$ the variables of r not present in the head, and $\varphi(\mathbf{x}, \mathbf{w})$ the body of r, let $P_{k+1}(r, I)$ be defined by: $\{\mathbf{c} \in \mathrm{dom}(I)^l \mid I^k \models \exists \mathbf{w}\, \varphi(\mathbf{c}, \mathbf{w})\}$ where $\mathrm{dom}(I)$ is the active domain of I.
- Let then $P_{k+1}(R, I)$ denote the union of $P_{k+1}(r, I)$ over all r with R in the head.

Finally, the *query result* of P on I, denoted $P(I)$, is the evaluation of the goal predicate of P on I. We often assume P is Boolean, in which case the result of the program on I is the Boolean "true" iff Goal() holds in I, and we simply say that I is a *model of* P or that I *satisfies* P. We alternatively refer to a *datalog query*, rather than to a datalog program, to emphasize that we are only interested in the evaluation on the goal predicate.

Under these semantics, it is easy to check that any UCQ can be transformed in linear-time into an equivalent MDL query, that does not involve any intensional predicate apart from Goal.

Containment. The main problem we deal with in this work is the classical notion of *query containment* [1].

Definition 2.1. Let Q and Q' be two queries over a signature σ. We say Q is *contained* in Q', denoted $Q \sqsubseteq Q'$, if, for any instance I over σ, $Q(I) \subseteq Q'(I)$.

Above we have defined containment in terms of the evaluation of Q over *all instances, finite and infinite*. But a simple (and well-known) argument shows that this coincides with containment when only finite instances are considered. If there is an instance I and tuple t such that $t \in Q(I)$, $t \notin Q'(I)$, then the fact that $t \in Q(I)$ is guaranteed by a finite collection of facts I_0 in I. Thus I_0 witnesses that Q is not contained in Q' over finite instances. Given this equivalence, *throughout this work we will assume that instances are finite*. For finite instances I, there will be a finite k such that the evaluation of datalog Q will be $P_{k+1}(\mathrm{Goal}, I)$ for Goal the goal predicate.

Example 2.2. Consider the following MDL program P that determines whether there is a path in a graph G from a node marked with the unary predicate S to one marked with the unary predicate T:

$$\mathrm{Goal}() \leftarrow T(x) \wedge \mathit{Reachable}(x)$$
$$\mathit{Reachable}(y) \leftarrow G(x, y) \wedge \mathit{Reachable}(x)$$
$$\mathit{Reachable}(x) \leftarrow R(x)$$

Now consider the UCQs:

$$Q_1 : \exists x \exists y\, R(x) \wedge G(x, y) \wedge T(y)$$
$$Q_2 : (\exists x\, R(x) \wedge T(x)) \vee (\exists x' \exists y'\, G(x', y'))$$

We have that $P \not\sqsubseteq Q_1$ but $P \sqsubseteq Q_2$: indeed, if I is the instance made of the facts $R(a)$ and $T(a)$, I is a model of P, but not a model of Q_1. And in any model of P, either the first rule defining *Reachable* is used, and then the second disjunct of Q_2 holds, or only the second rule defining *Reachable* is used, and then the first disjunct of Q_2 holds.

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 2.3. *There are polynomial time one-to-one reductions in both directions between the containment of Boolean MDL queries in Boolean UCQs and that of MDL queries in UCQs.*

Proof. Given a containment problem for Boolean MDL query Q_1 in UCQ Q_2, we can create a unary containment problem by adding an additional unary intensional predicate U to the signature. We create a new unary MDL query that returns the content of U whenever Q_1 is true. We similarly create a new UCQ by adding a conjunct $U(x)$ to all CQs.

In the other direction, given unary MDL Q_1 and UCQ Q_2, we again add a unary predicate to the signature. We create a new Boolean MDL query Q_1' that holds if Q_1 intersected with the unary predicate is non-empty, and similarly for Boolean UCQ Q_2' derived from Q_2. It is easy to see that containment is preserved by this reduction, since if Q_1 is not contained in Q_2 on some instance I one can expand to the larger signature by choosing the unary predicate to be the symmetric difference of Q_1 and Q_2 on I. □

The direction from non-Boolean to Boolean in the proposition above implies that lower bounds on Boolean containment of MDL into UCQ applies to unary MDL as well. The other direction is 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 [18] on containment of connected unary MDL queries into a union of unary connected queries, since connectedness is not preserved by the reduction.

Relying on Proposition 2.3, **we will only consider Boolean queries in the rest of this paper**. Because of this, note that we can assume that variables are not reused across conjuncts of a UCQ.

2.2 Monadic Expansion Trees

Before discussing the complexity of the containment problem, we introduce the notion of *monadic expansion tree of a Boolean MDL query* that will be used in various proofs throughout the paper. Monadic expansion trees are inspired by the notion of *expansion trees* of [18, 19]; a monadic expansion tree is not a special case of an expansion tree, but a refinement of this notion that adds structure to it exploiting the fact that the datalog query is monadic. Monadic expansion trees can also be seen as a special kind of *tree decomposition* [30].

Definition 2.4. A *monadic expansion tree* over some signature σ is an instance I over σ, together with a finite, rooted, ordered, unranked tree $\mathrm{T}(I)$. Every node n of $\mathrm{T}(I)$ is associated with a set of facts of I, called the *bag* of n and denoted $\mathrm{bag}(n)$. In addition, all nodes except for the root are associated with an element v from $\mathrm{dom}(I)$, called the *output element* of the node n. We require that:

(i) every fact $R(\mathbf{a}) \in I$ appears in a $\mathrm{bag}(n)$ for some $n \in \mathrm{T}(I)$;
(ii) for every $n \in \mathrm{T}(I)$, every value a appearing in atoms of $\mathrm{bag}(n) \in \mathrm{dom}(n)$ is either the output element of n or the output element of one of n's children in $\mathrm{T}(I)$;
(iii) for every non-root $n \in \mathrm{T}(I)$ with parent n', the output element of v appears in $\mathrm{bag}(n')$;
(iv) the output elements of nodes are all distinct.

We denote a node n by a pair $(v, \mathrm{bag}(n))$ consisting of its output value and bag of facts. The *rank* of a monadic expansion tree I is the maximal number of children of any node in $\mathrm{T}(I)$.

Example 2.5. Consider the following monadic expansion tree $(I, \mathrm{T}(I))$ over the same signature as in Example 2.2. It is represented as its tree $\mathrm{T}(I)$ (with the root to the left):

$$\{T(c)\} \text{ ——— } (c, \{G(b,c)\}) \text{ —— } (b, \{G(a,b)\}) \text{ —— } (a, \{S(a)\})$$

This tree happens to have no branching (i.e., it has rank 1). The corresponding instance is $I = \{S(a), G(a,b), G(b,c), T(c)\}$, verifying requirement (i). It is easy to check that the other requirements are satisfied.

Note that, for each domain element a, the nodes of a monadic expansion tree containing a form a connected subtree of size at most 2: indeed, requirement (ii) implies that any bag containing a must be the bag n that has a as output element – which is unique by requirement (iv) – or the

parent of n (if n is not the root). Together with the other conditions, this means that a monadic expansion tree is a special kind of tree decomposition [30].

A fundamental fact is that there always exists a monadic expansion tree that is a 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 [23, Theorem 5.2], [18, Proposition 3.7], or [13, Theorem 1] in the specific setting of limited access containment. One distinction of the result presented here with respect to those works is that our definition of monadic expansion tree enforces a stricter structure, in particular through the use of output elements associated to nodes, that we will need, further on, to prove upper bounds on the containment problem.

Proposition 2.6. *Let Q be an MDL query with at most k atoms in the body of each rule, and at most p intensional predicates. Let Q' be a UCQ over the extensional signature of Q. If Q is not contained in Q', there exists a monadic expansion tree $(I, \mathrm{T}(I))$, with at most $p \times k$ facts per bag, such that I satisfies $Q \wedge \neg Q'$.*

Proof. This proof is based on the notion of *expansion tree* of a datalog query, as defined in Section 2.4 of [18]. An *expansion tree* of a Boolean MDL query Q, as defined in that work, is a finite tree where each node is labeled by an *instantiated* rule r of Q. We will assume that this instantiation does not introduce any spurious equalities. Specifically, we fix for each node n of the tree a *one-to-one* mapping φ_n from the variables of the rule to some set of variables. We annotate the node n with the instantiated rule $\varphi(r)$. Furthermore, if a node is labeled with an instantiation of a rule r, it has a child for each intensional predicate atom A appearing in r. The label of that child is a rule r' with A in the head, with the variables of A mapped to the same variables as in r, and other variables of r mapped to fresh variables. The root of the tree is labeled with a rule that has the goal predicate as its head.

For any expansion tree t of an MDL query Q, we let $\Pi_{\mathrm{ext}}(t)$ be the set of all extensional atoms appearing in nodes of t. A critical observation made in [18] is that Q is equivalent to the infinite disjunction of the $\Pi_{\mathrm{ext}}(t)$, with t an expansion tree of Q, each $\Pi_{\mathrm{ext}}(t)$ being seen as a conjunctive query, where we conjoin all atoms and existentially quantify all variables. In particular, if some instance I_0 satisfies $Q \wedge \neg Q'$, then there exists an expansion tree t such that $I_0 \models \Pi_{\mathrm{ext}}(t)$ and, of course, we still have $I_0 \not\models Q'$.

Example 2.7. Returning to the Datalog program in Example 2.2, where we label the rules: r_1 : $\mathrm{Goal}() \leftarrow T(x) \wedge \mathit{Reachable}(x)$, r_2 : $\mathit{Reachable}(y) \leftarrow G(x, y) \wedge \mathit{Reachable}(x)$, r_3 : $\mathit{Reachable}(x) \leftarrow R(x)$. One expansion tree would have r_1 at the root, r_2 instantiated as $\mathit{Reachable}(x) \leftarrow G(x, z) \wedge \mathit{Reachable}(z)$ as the only child of the root, and r_3 instantiated as $\mathit{Reachable}(z) \leftarrow R(z)$ as a leaf child of the instantiation of r_2.

So far, we have not used the fact that Q is monadic. We do so now, by first arguing that we can choose t such that there are no two nodes n and n' in t, with n a strict ancestor of n', sharing the same head atom $R(x)$ with the same variable x (or having the same head atom Goal()). If there were such nodes, the subtree rooted at n could be replaced by the subtree rooted at n', resulting in a new expansion tree whose set of atoms is a subset of that of t, and that I_0 thus still models. This remark is similar to Proposition 3.15 of [18], which is also about the monadic case.

Moreover, we show how to turn t into a DAG structure G such that no two nodes n and n' have the same head atom $R(x)$ with the same variable x, or the same atom Goal(). For this purpose, an equivalence relation $\equiv$ is associated with the nodes of t such that $n \equiv n'$ iff n and n' have the same head atom. We fix an arbitrary representative of the equivalence class of every node n, denoted $\varphi(n)$. We then transform t into a graph G where each edge (n, n') is turned into an edge $(\varphi(n), \varphi(n'))$. Since we assumed no two equivalent nodes could be strict descendants of each other, G is acyclic.

We can define $\Pi_{\text{ext}}(G)$ similarly to the way we defined $\Pi_{\text{ext}}(G)$, as the conjunctive query formed of all extensional atoms of the rules annotating nodes of the graph. Note that $\Pi_{\text{ext}}(t)$ is contained in $\Pi_{\text{ext}}(G)$ since the set of atoms of G is a subset of that of t. This means our original I_0 is still a model of $\Pi_{\text{ext}}(G)$.

We now build a monadic expansion tree $(I, \mathrm{T}(I))$ from a "canonical model" of G. Let ν be a one-to-one mapping from the variables of G to constants. We let $I \coloneqq \nu(\Pi_{\text{ext}}(G))$ and define the tree $\mathrm{T}(I)$ as follows. Consider a value $c = \nu(x)$ of I for some variable x. We construct a new node n_c with c as output element, and let $\mathrm{bag}(n_c)$ contain all extensional facts associated by ν to the atoms of the nodes of G having x in their head atom. Similarly, we construct one node r whose bag contains all extensional facts associated to atoms of nodes of G having Goal() as the head atom. For each value d in a fact within the bag of a node n distinct from its output element if any exists, we put an edge from n to n_d. We will show that this structure is indeed a monadic expansion tree.

We first argue that the underlying structure is a DAG rooted at r. By definition, there cannot be an edge from a node n to itself, and there cannot be an edge to r. If there were a path from a node $n_{\nu(x)}$ to itself through a node $n_{\nu(y)}$ with $x \neq y$, it would mean that, in G, and thus in t, there would be a disconnected set of nodes with the same variable x in their head atom; this is in contradiction with the definition of expansion trees.

Second, we show that $\mathrm{T}(I)$ is not only a DAG, but a tree. We claim that for any variable x, there is only one node of $\mathrm{T}(I)$ whose bag involves a fact with $\nu(x)$ apart from the bag whose output element is $\nu(x)$. It will be the node corresponding to the parent of the topmost node of G whose head atom refers to x; it is easy to see that, in G a node can only have several parents if parents and child share the same head atom variable. We deduce that $\mathrm{T}(I)$ is a tree.

We now verify the properties required of monadic expansion trees:

(i) Every extensional atom in G is in some node n of G; it will thus be present in $\mathrm{T}(I)$, either in the root bag r if the head atom of n is Goal(), or in $n_{\nu(x)}$ if the head of atom of n is of the form $R(x)$.
(ii) If a value d appears in a bag n_c of $\mathrm{T}(I)$ then, by construction either $d = c$, or n_d is a child of n_c.
(iii) A node n_d cannot be a child of a node n in $\mathrm{T}(I)$ unless d appears in $\mathrm{bag}(n)$;
(iv) Nodes of $\mathrm{T}(I)$ are either the root or indexed by their output element, so that output elements of nodes are all distinct.

Let us now verify that the maximum number of facts per bag is less than or equal to $p \times k$. The root bag r contains, for each intensional atom Goal() corresponding to some node n of G, a fact for every extensional atom within n. Note that there are at most p intensional atoms and the number of extensional atoms n is at most the number of atoms in the body of the rule, which is at most k. Similarly, a bag $n_{\nu(x)}$ contains, for each intensional atom of the form $R(x)$ corresponding to some node n of G, a fact for every extensional atom within n. Here we note that there are at most p atoms of the form $R(x)$, and the number of extensional atoms in the rule is again at most k. We obtain the desired bound.

It only remains to show that $I \models Q$ and $I \not\models Q'$. The former is immediate: I is the extensional part of $\nu(G)$, which is a canonical model of G, when G is seen as a conjunctive query – and G can be expanded into an expansion tree of Q. The latter comes from the fact that we showed $I_0 \models \Pi_{\text{ext}}(G)$, which means that there exists a homomorphism $\rho : \Pi_{\text{ext}}(G) \to I_0$. If we had $I \models Q'$, witnessed by homomorphism μ, then $\rho \circ \nu^{-1} \circ \mu$ would be a homomorphism from Q' to I_0, contradicting $I_0 \not\models Q'$. □

Example 2.8. Consider, again, the monadic expansion tree of Example 2.5. It is actually a witness of non-containment of the program P of Example 2.2 into the UCQ's Q_1 from Example 2.2.

2.3 Trees and Tree Validity

Our results for datalog containment will make use of trees in several ways. First, our upper bounds will make use of techniques from tree automata, so we will need to review the definitions of several flavors of automata. Second we will show a tight connection between datalog containment and "universality" or "validity" problems for queries over trees: given a schema describing a set of trees and a Boolean query over trees, does every tree satisfy the query? There are many variants of the problem, depending on the exact signature of trees used. We will thus define several signatures below.

Let Λ be a finite non-empty set of labels. We will consider the settings of both binary and unranked trees. Many of our lower bounds will work in the restricted setting of binary trees. For binary trees with labels from Λ, the following signature is natural.

The *relational signature of ordered, labeled, binary trees*, denoted $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, contains the binary predicates FirstChild, SecondChild, unary Root, Leaf predicates, and Label_α predicates for all $\alpha \in \Lambda$.

A tree T over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ is a relational instance such that:

(i) the non-empty Label^T_α's for $\alpha \in \Lambda$ form a partition of $\text{dom}(T)$ (one can thus talk about the *label* of a node n, which is the $\alpha \in \Lambda$ such that $n \in \text{Label}^T_\alpha$);
(ii) FirstChild^T and SecondChild^T are one-to-one partial mappings with the same domain (the set of *internal nodes*), whose complement is Leaf^T (the set of *leaves*), and with disjoint ranges;
(iii) the inverses of FirstChild^T and SecondChild^T are one-to-one partial mappings;
(iv) $\exists x\, \text{FirstChild}(x, x) \vee \text{SecondChild}(x, x)$ does not hold;
(v) Root^T contains exactly one element (the *root* r of T), and the following formula does not hold for r: $\exists x\, \text{FirstChild}(x, r) \vee \text{SecondChild}(x, r)$.

We denote as $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ (resp., $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$) the relational signature containing all the relations of $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ together with a binary Child relation (resp., binary Child and $\text{Child}^?$ relations). A *tree* T over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ is a relational instance that verifies the same axioms as a tree over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, where Child^T is the disjoint union of FirstChild^T and SecondChild^T. A tree over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ has the additional requirement that $\forall x \forall y\, \text{Child}^?(x, y) \leftrightarrow (\text{Child}(x, y) \vee x = y)$ holds.

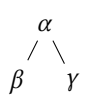
Note that we omit the label alphabet Λ from notation such as $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ for readability. Our upper bound results concerning $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ and $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ will hold for any label set Λ, while in our lower bounds we will usually show hardness for any label set of size at least 2.

The *relational signature of unordered, labeled, unranked trees*, denoted $\mathcal{S}^{\text{unranked}}_{\text{Child}}$, is made out of the binary predicate Child together with the unary Root, Leaf, and Label_α. A tree over $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ is a relational instance such that:

(i) the non-empty Label^T_α's for $\alpha \in \Lambda$ form a partition of $\text{dom}(T)$;
(ii) Child^T is a tree in the usual sense, whose root is the only element of Root^T and whose leaves are exactly the elements of Leaf^T.

We sometimes consider as special cases trees formed of a single node (i.e., trees such that $|\text{dom}(T)| = 1$); we call them *root-only* trees.

Example 2.9. Consider the simple abstract tree with a root labeled α and two children labeled β and γ respectively represented here with the root at the top:



In the four signatures introduced, this tree can be represented as the following collection of facts:

all four signatures	$\mathsf{Root}(r), \mathsf{Label}_\alpha(r), \mathsf{Label}_\beta(f), \mathsf{Leaf}(f), \mathsf{Label}_\gamma(s), \mathsf{Leaf}(s)$
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$	$\mathsf{FirstChild}(r, f), \mathsf{SecondChild}(r, s)$
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$	$\mathsf{FirstChild}(r, f), \mathsf{SecondChild}(r, s), \mathsf{Child}(r, f), \mathsf{Child}(r, s)$
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$	$\mathsf{FirstChild}(r, f), \mathsf{SecondChild}(r, s), \mathsf{Child}(r, f), \mathsf{Child}(r, s), \mathsf{Child}^?(r, r), \mathsf{Child}^?(f, f), \mathsf{Child}^?(s, s), \mathsf{Child}^?(r, f), \mathsf{Child}^?(r, s)$
$\mathcal{S}^{\text{unranked}}_{\text{Child}}$	$\mathsf{Child}(r, f), \mathsf{Child}(r, s)$

We will consider several methods for defining families of trees, in particular tree automata and document type definitions (DTDs). We define them formally in the binary case.

Definition 2.10. A *nondeterministic tree automaton on binary trees* (or BNTA) over finite alphabet Λ is of the form $(\Omega, \Delta_0, \Delta, F)$, where Ω (the control states) is a finite set, $\Delta_0 \subset \Lambda \times \Omega$, $\Delta \subset \Omega^2 \times \Lambda \times \Omega$, and $F \subset \Omega$. A run ρ of a BNTA over a Λ-labeled binary tree is an assignment of states to nodes. A run is accepting if for all leaves l labeled with $\alpha \in \Lambda$, $(\alpha, \rho(l)) \in \Delta_0$; the root is assigned a state in F; and if n has left and right children n_1 and n_2 respectively and label α, then $(\rho(n_1), \rho(n_2), \alpha, \rho(n)) \in \Delta$.

A *deterministic tree automaton on binary trees* (BDTA) over Λ is a BNTA in which for every $(q_1, q_2, a) \in \Omega^2 \times \Lambda$, there is at most one q such that $(q_1, q_2, a, q) \in \Delta$.

The set of all binary trees having an accepting run of BNTA A is the *language* of A, noted $L(A)$. Such a language is then said to be *regular*.

A nondeterministic tree automaton over ranked trees ($\mathsf{NTA}_{\mathsf{Rk}}$) is defined similarly, but with $\Delta \subset \bigcup_{i \leqslant r} \Omega^i \times \Lambda \times \Omega$ for some r. Such an automaton expects trees in which the outdegree of each vertex is at most r. The notion of deterministic tree automaton over ranked trees ($\mathsf{DTA}_{\mathsf{Rk}}$), the language of such an automaton, and regularity of a language of ranked trees is defined analogously to above.

We will also make use of the corresponding notion of nondeterministic tree automaton over unranked trees, $\mathsf{NTA}_{\mathsf{Unr}}$ and of a regular language for unranked trees, see [25]. We will not need to know the definition of a $\mathsf{NTA}_{\mathsf{Unr}}$, since most of the results involving $\mathsf{NTA}_{\mathsf{Unr}}$ will come from prior work. We will use the following simple facts relating $\mathsf{NTA}_{\mathsf{Unr}}$ to their ranked counterparts:

- A BNTA, and more generally an $\mathsf{NTA}_{\mathsf{Rk}}$, is a special case of a $\mathsf{NTA}_{\mathsf{Unr}}$, since we can enforce a restriction on the rank with an automaton.
- A witness for the non-emptiness of an $\mathsf{NTA}_{\mathsf{Unr}}$ A can always be taken to have rank polynomial in the size of A. This can be shown by just "trimming" a witness.

A *DTD for binary trees over* Λ (BDTD) is a pair (d, l_0) where d is a function from Λ to $2^{(\Lambda\times\Lambda)\cup\{\varepsilon\}}$ giving the constraints over the labels of the children of a node; l_0, an element of Λ, is the root label. A binary tree t is *accepted by a* BDTD (d, l_0) if (i) for any node n labeled a, if n is a leaf then $\varepsilon \in d(a)$ and, otherwise, if b and c are the labels of the first and second children of n then $(b, c) \in d(a)$; (ii) the root of t is labeled by l_0. The set of all trees accepted by a BDTD D is the *language* of D, noted $L(D)$. The standard notion of a DTD [28] is for unranked trees. For clarity and to keep a uniform notation we refer to these as UDTDs. For these, d is a function from Λ to regular expressions over Λ. The notion of acceptance of an unranked tree by a UDTD is standard, and we will not have need of it here. We will need the well-known and simple fact that BDTDs can be turned into BNTAs accepting the same language in linear time, and similarly for the unranked case.

Definition 2.11. A query on one of the signatures is *valid* over an automaton or DTD appropriate for that signature (e.g., BNTA or BDTD for a signature for binary trees) if for all trees that satisfy the schema, the query returns true. A query is *valid* with respect to a set of node labels if the query returns true on all trees over that set of node labels.

2.4 Limited Access Patterns

An important area where monadic datalog programs arise is that of *querying under access constraints* [6, 13, 24, 26, 29], also known as *querying under limited access patterns*, or, simply *limited access querying*. We recall basic notions from these works.

Access methods. Given a relational schema, we consider a set of *access methods*

$$\mathsf{ACS} = \{\, \mathsf{AcM}_1 \ldots \mathsf{AcM}_m \,\}$$

with each AcM_i consisting of a source relation $\mathsf{Rel}(\mathsf{AcM}_i)$ and a set $\mathsf{InputAtt}(\mathsf{AcM}_i)$ of *input positions* from the set of positions of $\mathsf{Rel}(\mathsf{AcM}_i)$; intuitively, each access method allows one to put in a tuple of values for $\mathsf{InputAtt}(\mathsf{AcM}_i)$ and get as a result a set of matching tuples. A position of R that is not an input position of a method AcM is an *output position* of AcM. As mentioned in the introduction, we assume that all attributes have the same domain; we do not consider separate abstract domains per attribute.

A combination of an access method and a binding to the input places of the accessed relation will be referred to as an *access*. We will often write an access by adding "?" to the non-input places, omitting the exact method: e.g., $R(3, ?)$ is an access (via some method) to R with the first place bound to 3. If R does not have any output positions, we say that it is a *Boolean* access, and we write for instance $R(3, 4)?$ for an access that checks whether $(3, 4) \in R$. If R does not have any input positions, we say that it is a *free* access. In particular, free accesses can serve to model some initial facts (or some initial data values) that are known before making any access.

We also assume, following the literature,[1] that *each relation has exactly one access method* – allowing relations with no access would not make any difference in our setting, as they would make queries using them unanswerable.

Given a set of access methods and an instance I over signature σ, we define a sequence of values $\mathsf{AccValues}_k(I)$ and a sequence of facts $\mathsf{AccFacts}_k(I)$ by mutual induction as follows:

- $\mathsf{AccValues}_0(I) = \emptyset$;
- For $k \geqslant 0$:

$$\mathsf{AccFacts}_k(I) = \bigcup_{\substack{\mathsf{AcM} \in \mathsf{ACS} \\ \{j_1 \ldots j_m\} = \mathsf{InputAtt}(\mathsf{AcM})}} \; \bigcup_{v_1, \ldots, v_m \in \mathsf{AccValues}_k(I)} \left[\sigma_{\#j_1 = v_1 \wedge \cdots \wedge \#j_m = v_m}(\mathsf{Rel}(\mathsf{AcM}))\right](I);$$

- For $k \geqslant 0$, $\mathsf{AccValues}_{k+1}(I) = \mathsf{dom}(\mathsf{AccFacts}_k(I))$.

We denote $\mathsf{AccValues}(I)$ and $\mathsf{AccFacts}(I)$, respectively, the fixpoint of these two sequences.

Containment under access constraints. We now give the formal definition of containment under limited access patterns.

Definition 2.12. Let Q and Q' be two queries over the relational schema. Q *is contained in* Q' *under the access methods of* ACS if, for every instance, I $Q(\mathsf{AccFacts}(I)) \subseteq Q'(\mathsf{AccFacts}(I))$.

Example 2.13. We consider again the same signature as in Example 2.1: a binary relation G and two unary relations S and T. Assume that the access methods consist on a free access to relation R, an access with input on the first position for relation G, and a Boolean access on relation T.

Then, under these access methods, the CQ $\exists x\, T(x)$ is contained in the UCQ :

$$(\exists x\, T(x) \wedge R(x)) \vee (\exists x' \exists y'\, T(y') \wedge G(x', y')).$$

[1] To the best of our knowledge, with the exception of [6], work on containment under limited access patterns has assumed (at most) a single access per relation. Indeed, access patterns are usually defined to be attached to positions of a relation. See Section 7 for a discussion of the importance of this assumption.

Indeed, the only way to make an access to relation T is for the value bound to be accessible, which is only possible if it has been produced by the free access to relation R or by an access to relation G, with first position bound to an accessible value.

As mentioned in the introduction, containment under limited accesses is strongly related to containment of monadic datalog queries [26], in a manner that we now explain. Assume that Q and Q' are Boolean UCQs. Then containment of Q in Q' under ACS can be reduced to the containment of a monadic datalog query P in Q'. We write $Q = \bigvee_{i=1}^{k} \varphi_i(\mathbf{x}_i)$ with the $\varphi_i(\mathbf{x}_i)$ conjunctions of atoms. The monadic datalog query P makes use of an intensional predicate AccValues and is formed of the following rules:

$$\mathsf{Goal}() \leftarrow \varphi_i(\mathbf{x}_i) \wedge \bigwedge_j \mathsf{AccValues}(x_{i,j}) \qquad \text{for } 1 \leqslant i \leqslant k \tag{1}$$

$$\mathsf{AccValues}(x_j) \leftarrow \mathsf{Rel}(\mathsf{AcM}_\ell)(\mathbf{x}) \wedge \bigwedge_{i \in \mathsf{InputAtt}(\mathsf{AcM}_k)} \mathsf{AccValues}(x_i) \tag{2}$$

for every $1 \leqslant \ell \leqslant m$, and output position j of AcM_ℓ.

Note that if we were to deal with abstract domains, these rules would change to include a predicate $\mathsf{AccValues}_\tau$ for each abstract domain τ.

Example 2.14. The monadic datalog program resulting of the rewriting of query $\exists x\, T(x)$ for the access methods of Example 2.13 is:

$$\begin{aligned} \mathsf{Goal}() &\leftarrow T(x) \wedge \mathsf{AccValues}(x) \\ \mathsf{AccValues}(y) &\leftarrow G(x, y) \wedge \mathsf{AccValues}(x) \\ \mathsf{AccValues}(x) &\leftarrow R(x) \end{aligned}$$

Note that this is exactly the program of Example 2.2.

From the previous discussion and Proposition 2.6, we derive immediately:

Corollary 2.15. *If a UCQ Q is not contained in a UCQ Q' under a set of access constraints, then there exists a monadic expansion tree I that is a model of the MDL query P associated to Q under the access constraints, such that I satisfies $Q \wedge \neg Q'$.*

We now have all the necessary elements to state the main results proved in this work.

3 STATEMENT OF THE MAIN RESULTS

In this paper we study three problems of interest, with strong connections: containment of monadic datalog, validity problems on trees, and containment of UCQs under access constraints.

Though the connection between monadic datalog and querying under limited accesses is well-known [26], one major contribution of this work is to highlight the connection to tree validity problems.

In Section 4, we present upper bounds, by providing a technique for giving EXPTIME upper bounds on certain variations of both the tree validity problem and the query containment problems.

In our lower bound section (Section 5), we prove 2EXPTIME and EXPTIME lower bounds for different variants of the tree validity problem. The technique here is to adapt ideas from [9]. We then use a reduction from tree validity to MDL containment (given in the proof of Theorem 5.10) that allows us to push the 2EXPTIME lower bound to the MDL containment problem, and another reduction (given in the proof of Theorem 5.11) that allows us to push EXPTIME hardness from tree validity to containment under access constraints.

Let us now briefly review the currently known bounds, as well as the results proved here, on tree validity and query containment.

3.1 Results on Tree Validity Problems

We begin by overviewing results on tree validity that are either explicitly in prior work, can be derived with little effort from prior work, or are easy to derive directly. The discussion here will be quite abbreviated, but it does not concern the main results of the paper.

We first note that validity over all trees is tractable for CQs:

Proposition 3.1. *Determining if a CQ is valid over all trees for a given label set can be done in* PTIME *over* $\mathcal{S}^{\text{unranked}}_{\text{Child}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$, *and* $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$.

Proof. We claim a CQ Q is valid if and only if the following conditions are all satisfied:

(i) it does not contain any FirstChild, SecondChild, or Child atom;
(ii) either it does not contain any Label_α atom or $|\Lambda| = 1$;
(iii) there are no (possibly identical) variables x_1, x_2 such that (x_1, x_2) is in the reflective transitive symmetric closure of the set of pairs of variables appearing in a $\text{Child}^?$ atom, and such that $\text{Root}(x_1)$ and $\text{Leaf}(x_2)$ appear in Q.

Indeed, if any of these conditions is not satisfied, Q is not valid:

(i) any root-only tree is a counterexample;
(ii) any tree with no node of that label is a counterexample;
(iii) any tree formed of a single chain of nodes of length greater than the number of $\text{Child}^?$ atoms plus 1 is a counterexample.

Conversely, if T is an arbitrary tree and Q satisfies (i)–(iii), mapping every query variable of Q to the root of the tree if the connected component of that variable does not include any Leaf atom, and to an arbitrary leaf of the tree otherwise, yields a homomorphism. Indeed, we know that Q contains only Root, Leaf, and $\text{Child}^?$ atoms, and that in each connected component of Q there cannot be both a Root and Leaf atom. □

Apart from this very special case, the best upper bound known for the tree validity problems we consider is 2EXPTIME. Indeed, in [8, Theorem 11], validity of a query over $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ with respect to an NTA_{Unr} was shown to be in 2EXPTIME for CQs. This 2EXPTIME upper bound actually holds for all considered problems on tree validity. For most of our signatures, such a bound can be obtained as follows. Convert the UCQ Q to an exponential-sized tree automaton (e.g., BNTA for signatures appropriate to binary trees) A_Q in exponential time. See, for example, [2, Proposition B.1] for this conversion. Then using standard automata techniques [21] we can determinize A_Q in exponential time, complement it, and intersect it with the automaton representing the schema in polynomial time. Finally, we can test the resulting automaton for emptiness in polynomial time. In the case of $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$, we first convert a UCQ Q over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ to a positive existential query Q' over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$. The query Q' can be converted in exponential time to an alternating automaton over trees. The construction is a standard induction: the atoms are converted to automata that work over trees with the free variables annotated on the tree. Conjunction and disjunction are done using the closure properties of alternating automata, which allow positive Boolean combinations in the transition function. Existential quantification can be assumed to be outermost, and requires projecting out the annotations. This can be done by converting the alternating automaton to a nondeterministic automaton in exponential time; for nondeterministic automata the projection step is straightforward. Emptiness of alternating automata can be checked in exponential time [21], which gives the 2EXPTIME bound.

Let us now discuss existing lower bounds. The validity problem with respect to DTDs over $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ has been studied in [9]. [9, Theorem 12] shows that the validity problem for $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ is EXPTIME-hard for *child-only tree-pattern queries*. Given that one can convert these straightforwardly

Table 1. Summary of results on the complexity of tree validity of CQs and UCQs, over various tree signatures and with respect to DTDs, tree automata, or all trees

Previously known results [8, 9] and straightforward arguments from Section 3.1
(in particular, Proposition 3.1 for PTIME results)

Signature	CQ (DTD or tree automaton)	UCQ (DTD or tree automaton)	CQ (all trees)	UCQ (all trees)
$\mathcal{S}^{\text{unranked}}_{\text{Child}}$	EXPTIME-hard, in 2EXPTIME	EXPTIME-hard, in 2EXPTIME	PTIME	in 2EXPTIME
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$	in 2EXPTIME	in 2EXPTIME	PTIME	in 2EXPTIME
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$	EXPTIME-hard, in 2EXPTIME	EXPTIME-hard, in 2EXPTIME	PTIME	in 2EXPTIME
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$	EXPTIME-hard, in 2EXPTIME	EXPTIME-hard, in 2EXPTIME	PTIME	in 2EXPTIME

Results proved in Sections 4 (upper bounds) and 5 (lower bounds).
We give references to statements proving the bounds;
lower bounds w.r.t. tree automata are transferred to lower bounds w.r.t. DTDs using Corollary 5.2.

Signature	CQ (DTD or tree automaton)	UCQ (DTD or tree automaton)	CQ (all trees)	UCQ (all trees)
$\mathcal{S}^{\text{unranked}}_{\text{Child}}$	EXPTIME-complete (Cor. 4.21)	EXPTIME-complete (Cor. 4.21)	PTIME	EXPTIME-complete (Cor. 4.21, Prop. 5.8)
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$	in EXPTIME (Cor. 4.21)	EXPTIME-complete (Cor. 4.21, Thm 5.6)	PTIME	EXPTIME-complete (Cor. 4.21, Cor. 5.7)
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$	EXPTIME-complete (Cor. 4.21)	EXPTIME-complete (Cor. 4.21)	PTIME	EXPTIME-complete (Cor. 4.21, Cor. 5.7)
$\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$	2EXPTIME-complete (Thm 5.3)	2EXPTIME-complete (Thm 5.3)	PTIME	2EXPTIME-complete (Cor. 5.5)

to CQs, we obtain EXPTIME-hardness of the validity problem for CQs (and thus UCQs) with respect to UDTD and NTA_{Unr}.

Inspection of prior work easily shows the lower bound carries over to the $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ signature and, consequently, to the $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ signature, as we now explain. Theorem 12 of [9] relies on Theorem 11 in the same paper, whose proof involves a reduction from finding a winning strategy in a game on tiling systems [20, Rectangle Tiling Game]. Critically, the number of possible moves in this strategy is bounded, by the number of different tiles, which is fixed. Thus the trees involved in the hardness proof are actually ranked. Now, we use a standard encoding of b-ranked trees as binary trees where every node n with at most b children is replaced with a binary subtree of height exactly $\lceil \log_2(b) \rceil$ whose leaves are the children of n. This means that, in the CQ, we replace every Child atom with a chain of $\lceil \log_2(b) \rceil$ child atoms. In the DTD, we enumerate the bounded number of possible words for the labels of children of every node label, and choose fresh node labels for every such possible word and every position in the binary tree encoding the unranked Child relation. It then becomes easy to transform the UDTD on unranked trees into a BDTD on the encoded binary trees.

This concludes our discussion of the state of the art prior to our work. Table 1 (upper) summarizes the results that can be derived from [9] and from the other arguments given so far in this section. In the table, we have abbreviated "all trees in a given label set" by "all trees".

In this work, we establish tight complexity bounds for the validity of CQs and UCQs over all four tree signatures introduced ($\mathcal{S}^{\text{unranked}}_{\text{Child}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$, and $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$) with respect to DTDs, tree automata, and over all trees. The results are summarized in Table 1 (lower). The

Table 2. Summary of results on the complexity of query containment. We give references to statements proving the bounds, from Sections 4 (upper bounds) and 5 (lower bounds) or the literature.

Query containment setting	Containment	Upper bound	Lower bound
MDL in MDL	2EXPTIME-complete	[22, Thm 7.2]	Thm 5.9
MDL in UCQ	2EXPTIME-complete	[22, Thm 7.2]	Thm 5.9
MDL in CQ	2EXPTIME-complete	[22, Thm 7.2]	Thm 5.9
UCQ in UCQ + access constraints	EXPTIME-complete	Cor. 4.28	Thm 5.11
AGEMDL in UCQ	EXPTIME-complete	Cor. 4.28	Cor. 5.12
3-GEMDL in UCQ	2EXPTIME-complete	[22, Thm 7.2]	Thm 5.10
Datalog in UCQ	2EXPTIME-complete	[19, Thm 5.12]	[19, Thm 5.15]

results on validity over all trees refer to the combined complexity of the problem that takes as input both the query and label set, determining if the query is valid for that label set. There is one exception where a tight bound is still open: the case of CQs over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ with respect to BDTDs or BNTAs. In all other cases (beyond the trivial PTIME case of CQs over all trees), we establish 2EXPTIME-completeness (for $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$) and EXPTIME-completeness (for the other three signatures).

3.2 Results on Containment for MDL and Access Constraints

Recall that containment of MDL queries in UCQs is in 2EXPTIME by [22] (indeed, this holds also for containment of two MDL queries [22] or for Datalog in UCQs [17, 19]). In this paper we show that this problem is 2EXPTIME-hard, thus obtaining a tight characterization of its complexity.

In contrast, we show that containment of UCQs under access constraints is EXPTIME-complete. Both the upper and lower bound here are non-trivial. We will also display a subset of MDL, AGEMDL, that exhibits the same behavior. In an AGEMDL query, 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. We will see that this restriction of MDL to obtain EXPTIME-completeness is somewhat robust: if we allow a bounded number of occurrences of every extensional predicate except of just one (we call k-GEMDL the corresponding fragment of MDL, where k is the bound), the complexity of containment jumps back to 2EXPTIME-complete.

Table 2 summarizes these results and provides references to the corresponding theorems.

4 UPPER BOUNDS

In a first step, this section will introduce the main upper bound technique of the paper (Sections 4.1 to 4.3). It consists of machinery for showing that certain containment problems have exponential-sized counterexamples, based on analysis of models that are monadic expansion trees.

We will then show (Section 4.4) that the machinery gives new bounds for tree validity problems. We will also show (Section 4.5) that it can be used to obtain new bounds for containment under access constraints, by reduction from a bound on the AGEMDL fragment of MDL, introduced there.

Our machinery revisits the argument showing the 2EXPTIME upper bound of containment of arbitrary datalog queries in UCQs. This result is proved in [19], as Theorem 5.15 of that paper.

By Proposition 2.6, since we are dealing with *monadic* datalog queries here, we can restrict ourselves to monadic expansion trees. We introduce in Section 4.1 a notion of "interface queries" used to define a type over the nodes of a monadic expansion tree, called *IQ-type* (Definition 4.5). We show that IQ-types are enough to characterize monadic expansion trees that are counterexamples

to containment, with the use of automata techniques (Theorem 4.10). This allows us to bound the size of a minimal counterexample by the number of IQ-types (see Section 4.2). We show how this yields a generic recipe for getting bounds on the containment problem, by highlighting a property of some classes of instances (the *unique mapping condition*, see Section 4.3) that guarantees a bound on the number of IQ-types.

4.1 IQ-types

How big does a monadic expansion tree that is a counterexample to containment of Q into Q' need to be? Clearly it needs to be big enough to witness satisfaction of Q, but the main issue is how big it needs to be to witness non-satisfaction of Q'. Intuitively, one only needs nodes that represent the different kinds of behavior with respect to Q'. A crude notion of "same behavior" w.r.t. Q' would be to identify a node with the collection of subqueries 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 in this section is a finer notion of similarity that takes advantage of the restricted structure of monadic expansion trees. 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. Given a subinstance corresponding to a subtree of the monadic expansion tree, we can capture that interaction by the restriction of the mapping of query variables of Q' to the root of this subtree. We can thus think of fixing a "root interface". Naively, 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 element of the root; (2) for each set of variables at a time, for each connected component of Q' disregarding this set of variables, information about whether the corresponding query is satisfied in the subinstance. We formalize this idea of satisfied "interface queries" through the notion of IQ-types.

4.1.1 Definitions. We recall that the tree structure associated with a monadic expansion tree is denoted by $\mathrm{T}(I)$.

Definition 4.1. Let I be a monadic expansion tree and n a node of $\mathrm{T}(I)$. The *subinstance of I rooted at n* is the set of facts contained in the bags of n and its descendants.

The queries defining IQ-types have to map the facts belonging to bags of descendants of a node in a particular way, given by the notion of relative homomorphism below.

Definition 4.2. Let q' be a conjunctive query and X a subset of the variables of q'. Let I be a monadic expansion tree and n a node of $\mathrm{T}(I)$. A *relative homomorphism h* from the pair (X, q') to the subinstance rooted at n is a function from the variables of q' to the active domain of the subinstance rooted at n such that:

(i) the set of variables mapped to the output element of n by h is equal to X;
(ii) $h(q')$ is included in the subinstance rooted at n;
(iii) at least one of the facts of $h(q')$ is included in the bag of n.

Note that if X is non-empty, the third condition is trivially satisfied: a fact that includes a variable in X is necessarily in the bag of n.

For connected queries, the existence of a homomorphism in the usual sense implies the existence of a relative homomorphism:

Lemma 4.3. *Let q' be a connected conjunctive query. A monadic expansion tree I satisfies q' if and only if there exist a node $n \in \mathrm{T}(I)$, a subset X of the variables of q', and a relative homomorphism from (X, q') to the subinstance rooted at n.*

Proof. The "if" part is obvious: a relative homomorphism from (X, q') to the subinstance rooted at n is in particular a homomorphism from q' to I.

Let h be a homomorphism from q' to I. Let n be the least common ancestor of all nodes of $\mathrm{T}(I)$ having in their bag a fact of $h(q')$. We let X be the (possibly empty) set of variables mapped to the output element of n by h. Observe that one fact of $h(q')$ is in the bag of n. Indeed, q' being connected, the bags containing facts of $h(q')$ form a connected subtree of $\mathrm{T}(I)$; n is the root of this subtree. Therefore, h is a relative homomorphism from (X, q') to the subinstance rooted at n. □

In order to define IQ-types, we need a few further definitions. As explained above, IQ-types define some root interface using a set of queries. The definition of these queries is given below.

Definition 4.4. Given a UCQ Q', and a subset X of the variables of Q', a subquery q' of Q' *covers* X if for each variable x in X, there exists an atom of q' containing x.

A subquery q' of Q' is *closed relative to* X if, whenever q' contains one atom with a variable x not in X then q' contains all atoms of Q' containing x. (Recall that since we only consider Boolean queries, we assume that a variable is not reused across CQs of a UCQ.)

A subquery q' of Q' is *connected relative to* X iff for each atom α_1 and α_2 in q', there exist two variables x_1 in α_1 and x_2 in α_2, both not in X, such that there is a path of variables not in X from x_1 to x_2, where the path is in the graph connecting two variables if they co-occur in an atom of q'. Note that we allow degenerate paths that consist of a single variable $x_1 = x_2$.

A *maximal connected component* of a query relative to X is a closed and connected subquery relative to X that covers X.

Note that if the set of variables X is empty, a maximal connected component of Q' relative to X is a maximal connected component in the usual sense; we let $\mathrm{MCC}(Q)$ be the set of maximal connected components of a query Q.

We now use relative homomorphisms and maximal connected components to define types of a node in a monadic expansion tree.

Definition 4.5. Let Q' be a UCQ. A pair (X, q'), where q' is a subquery of Q' and X a subset of the variables of Q', is a *type element of* Q' iff q' is a maximal connected component of Q' relative to X.

Let I be a monadic expansion tree and n a node of $\mathrm{T}(I)$. A type element (X, q') is *satisfied by a node* n iff there exists a relative homomorphism from (X, q') to the subinstance rooted at n.

The *IQ-type* of a node n with respect to Q', denoted by $\mathrm{type}_{Q'}(n)$, is the set of all type elements satisfied by n.

This notion of IQ-type will be used to characterize instances that satisfy a given query.

Example 4.6. To illustrate the notion of type element, consider the query

$$Q' = \exists x \exists y \exists z\ C(x) \wedge R(x, y) \wedge R(y, z) \wedge C(z)$$

We consider the following pairs consisting of a set of variables and a set of atoms from Q':

$$T_1 = (\{x, z\}, \{R(x, y), R(y, z)\})$$
$$T_2 = (\{x\}, \{R(x, y), R(y, z), C(z)\})$$

We claim that both T_1 and T_2 are type elements of Q'. We argue that the set of atoms of T_1 is closed and connected relative to the set of variables in T_1, and similarly for T_2. We give the argument for T_2, with the one for T_1 being similar. T_2 is connected relative to $\{x\}$ since all pairs of atoms are connected by a path of variables different from x: the two R atoms are connected through y, the second R atom and the C atom through z, and the first R atom and the C atom through z and y. To verify relative closedness of T_2 we need only notice that T_2 contains every atom containing z and every atom containing y.

4.1.2 Composition Lemma. We now introduce an important lemma, the composition lemma, critical to the soundness of our technique for finding monadic-expansion-tree counterexamples to containment of bounded size.

Definition 4.7. Let I and I' be two monadic expansion trees and n and n' two nodes of $\mathrm{T}(I)$ and $\mathrm{T}(I')$, respectively. A bijection φ from the values of n to the values of n' is an *output-isomorphism* iff

(i) it is an isomorphism, i.e., for each atom $R(a'_1 \dots a'_k)$ in n' there exists exactly one atom $R(a_1 \dots a_k)$ in n such that $\varphi(a_i) = a'_i$ for all $1 \leqslant i \leqslant k$;
(ii) child ordering is preserved: if v is the output element of the jth child of n, then $\varphi(v)$ is the output element of the jth child of n';
(iii) the output element of n maps to the output element of n'.

Two nodes n and n' are *output-isomorphic* iff there exists an output-isomorphism from the values of n to the values of n'.

We now state our composition lemma: the IQ-type of a node is determined by its bag of facts (up to output-isomorphism) and the IQ-types of its children.

Lemma 4.8 (Composition Lemma). *Let Q' be a UCQ, and I, I' two monadic expansion trees. Let $n \in \mathrm{T}(I)$ and $n' \in \mathrm{T}(I')$ two nodes of these instances such that there is an output-isomorphism φ from n to n'. Assume that for all children n_c of n and n'_c of n' with the output element of n_c mapped by φ to the output element of n'_c, $\mathrm{type}_{Q'}(n_c) = \mathrm{type}_{Q'}(n'_c)$. Then $\mathrm{type}_{Q'}(n) = \mathrm{type}_{Q'}(n')$.*

Proof. We denote $n_1 \dots n_k$ the children of n; $n'_1 \dots n'_k$ the children of n'. We suppose that for each $1 \leqslant i \leqslant k$, the IQ-types of n_i and n'_i with respect to Q' are the same and that if v is the output element of n_i then $\varphi(v)$ is the output element of n'_i.

We demonstrate that $\mathrm{type}_{Q'}(n') \subseteq \mathrm{type}_{Q'}(n)$: for any subset of variables X of Q' and for any maximal connected component q' of Q' relative to X, if there exists a relative homomorphism from (X, q') to the subtree rooted at n then there exists a relative homomorphism from (X, q') to the subtree rooted at n'. The other direction is implied by the symmetry of the roles of n and n'.

Let X be a subset of the variables of Q' and q' be a maximal connected component of Q' relative to X. Let h be a relative homomorphism from (X, q') to the subtree rooted at n. We need to create a homomorphism h' from (X, q') to the subtree rooted at n'.

For $1 \leqslant i \leqslant k$ we denote by q'_i the maximal subquery mapped by h to the subinstance rooted at n_i. Let A be the set of integers i with $1 \leqslant i \leqslant k$ such that q'_i is non-empty. For $i \in A$, let X_i be the subset of variables of q'_i such that the image of X_i by h is the output element of n_i.

We claim that (X_i, q'_i) is a type element of Q', by exploiting the structure of monadic expansion trees:

- q'_i is closed relative to X_i: since every variable x of q'_i not in X_i maps to values in the subinstance rooted at n_i, all atoms of q' containing x are mapped to the subinstance rooted at n_i;
- q'_i is connected relative to X_i, since q' is connected relative to X and every path of atoms between variables of q'_i not in X_i must be entirely in the subinstance rooted at n_i.

Furthermore, q'_i covers X_i by definition of X_i. Hence, (X, q'_i) is a type element of Q'.

Let α be an atom mapped by h to the bag of n (such an α necessarily exists by the definition of a relative homomorphism) and α_i mapped by h to the subtree rooted at n_i. Since q' is connected relative to X, there exists a path between α and α_i, which translates into a path between $h(\alpha)$ and $h(\alpha_i)$ that necessarily involves the output element of n_i due to the monadic expansion tree structure of the instance, and thus a fact within the bag of n_i. We have thus shown that the restriction h_i of h to the variables of q'_i is a relative homomorphism from (X_i, q'_i) to the subtree rooted at n_i. In

other words, $(X_i, q_i') \in \text{type}_{Q'}(n_i)$. Since $\text{type}_{Q'}(n_i) = \text{type}_{Q'}(n_i')$, there exists a homomorphism h_i' from (X_i, q_i') to the subtree rooted at n_i'.

We then define $h'(x)$ as follows: if $h(x)$ is a value present in a subinstance rooted at some n_i for $i \in A$, we set $h'(x) := h_i'(x)$; if $h(x)$ is a value of $\text{bag}(n)$, we set $h'(x) := \varphi(h(x))$. This definition is well formed: the subinstances rooted at n_i and n_j do not share any common value, and any value shared between the bag of n and the subinstance rooted at n_i must be the output element of n_i. The requirement that φ preserves child ordering ensures that the output element of any n_i in the domain of φ is mapped to the output element of n_i'. Thus we know that for any variable $x \in X_i$, $h_i'(x) = \varphi(h(x))$.

Let α be any atom of q'. Since h is a relative homomorphism from (X, q') to the subtree rooted at n, $h(\alpha)$ is either in $\text{bag}(n)$ (and there is at least one such α) or in one of the subinstance rooted at one of the n_i for $i \in A$. In the former case, $h'(\alpha) = \varphi(h(\alpha)) \in \text{bag}(n')$. In the latter case, $h'(\alpha) = h_i'(\alpha)$, which is in the subinstance rooted at n_i'. In addition, since φ is an output-isomorphism, the only variables mapped to the output element of n' by h' are the variables mapped to the output element of n by h, i.e., the set X. In other words, h' is a relative homomorphism from (X, q') to the subtree rooted at n'. □

4.1.3 From IQ-Types to Automata. We now explain how to see IQ-types as states of a tree automaton operating on monadic expansion trees. This will allow us to see the containment problem in terms of tree automata and derive upper bounds from there.

More specifically, IQ-types capture the "state" of a node in a monadic expansion tree with respect to a query: they can be used as states in a deterministic ranked tree automaton that accepts monadic expansion trees *not* satisfying a given query. Then, the number of IQ-types with respect to a given query will have a direct impact on the size of the tree automaton, and from there on the complexity of the containment problem. We emphasize that *the ability to use IQ-types to form such an automaton relies on some general properties such as the Composition Lemma, not on the fine details of IQ-types.* These details will come into play later – in Section 4.4 and Section 4.5, when we want to argue that in certain situations we have a tighter bound on the number of IQ-types, and hence a tighter bound on the size of the automaton.

Let σ be a relational signature and d a positive integer. We denote by $\mathcal{L}(d, \sigma)$ the set of monadic expansion trees over σ with at most d facts in the bag of each node. We denote by $B(d, \sigma)$ the set of output-isomorphism classes of nodes in the instances of $\mathcal{L}(d, \sigma)$. Since the size of bags is bounded, the size of $B(d, \sigma)$ is finite. Let us derive a bound on its size: to choose an element of $B(d, \sigma)$, one can choose at most d relation names among $O(|\sigma|)$ and, for each of the positions in the corresponding atoms, the number of which is in $O(d \cdot |\sigma|)$, choose a constant among $O(d \cdot |\sigma|)$ possible ones; then, one needs to choose at most $(r + 1)$ output elements among these $O(d \cdot |\sigma|)$ values, where r is the rank of the tree. Since r is itself bounded by the number of constants, i.e., $O(d \cdot |\sigma|)$, we obtain a total bound on the size of $B(d, \sigma)$ of:

$$O\left(|\sigma|^d + O(d \cdot |\sigma|)^{O(d \cdot |\sigma|)}\right) = O\left(\chi^{d \cdot |\sigma|}\right)$$

for some constant χ, i.e., simply exponential in d and in the size of σ.

For each output-isomorphism class c of $B(d, \sigma)$, we choose a distinct symbol $\hat{c}$ and we consider the finite set $\Gamma(d, \sigma) = \{ \hat{c} \mid c \in B(d, \sigma) \}$. $\Gamma(d, \sigma)$ is used as an alphabet for the finite trees that will be used to abstract monadic expansion trees out. For $I \in \mathcal{L}(d, \sigma)$, we denote by $\text{Code}_d(I)$ the finite tree labeled by tags in $\Gamma(d, \sigma)$ obtained from $\text{T}(I)$ by labeling each node of $\text{T}(I)$ by the symbol $\hat{c}$ corresponding to its output-isomorphism class c.

Definition 4.9. Let σ be a signature, $\mathcal{I}$ a collection of monadic expansion trees over σ, and d a positive integer. We say $\mathcal{I}$ is *d-regular* if $\mathcal{I} \subseteq \mathcal{L}(d, \sigma)$ and there exists a $\mathsf{DTA}_{\mathsf{Rk}}$ $A_{\mathcal{I}}$ over the alphabet $\Gamma(d, \sigma)$ recognizing exactly the set $\{ \mathrm{Code}_d(I) \mid I \in \mathcal{I} \}$.

A collection of monadic expansion trees is said to be *regular* if it is d-regular for some d.

Let $\mathcal{I}$ be a regular set of monadic expansion trees. Let Q' be a *connected* conjunctive query (we will consider the case of arbitrary UCQs further on). We reduce the problem of validity of Q' over a regular set of monadic expansion trees $\mathcal{I}$ to the emptiness of a tree automaton. More precisely, we build a deterministic ranked tree automaton $A_{Q'}$ over $\Gamma(d, \sigma)$ such that the intersection of $A_{Q'}$ and $A_{\mathcal{I}}$ recognizes exactly the set $\{ \mathrm{Code}_d(I) \mid I \in \mathcal{I} \wedge I \not\models Q' \}$. Intuitively, the states of $A_{Q'}$ are the types of nodes of instances of $\mathcal{I}$ which cover Q'. We will use Lemma 4.8 to calculate the transition function.

The theorem below formalizes this intuition:

Theorem 4.10. *Consider a signature σ, a positive integer d, and let $\mathcal{I}$ be a d-regular set of monadic expansion trees recognized by the ranked deterministic tree automaton $A_{\mathcal{I}}$. Let Q' be a* connected *conjunctive query.*

There exists a $\mathsf{DTA}_{\mathsf{Rk}}$ $A_{Q'}$ over the alphabet $\Gamma(d, \sigma)$, such that $A_{Q'} \cap A_{\mathcal{I}}$ accepts the set of all $\mathrm{Code}_d(I)$ with $I \in \mathcal{I}$ not satisfying Q'. Moreover, the size of $A_{Q'}$ is in $O\left(|\mathcal{Y}|^{\mathrm{poly}(d,\sigma)}\right)$ where $\mathcal{Y}$ is the set of IQ-types with respect to Q' of nodes of some instances of $\mathcal{I}$.

Finally, if one is given a superset $\mathcal{Z}$ of $\mathcal{Y}$, then we can compute $A_{Q'}$ in time $O\left(|\mathcal{Z}|^{\mathrm{poly}(d,|\sigma|,|Q'|)}\right)$.

Before we go into the details of the automaton, we classify the types that correspond to good (accepting) and bad (rejecting) states.

Definition 4.11. Let Q' be a conjunctive query, Q'' a subquery of Q', and τ an IQ-type with respect to Q'. We say that τ *covers* Q'' if τ contains type elements $(X, q_1) \dots (X, q_n)$ for some set of variables X and subqueries $q_1 \dots q_n$ that together contain each atom of Q''.

We note the following simple consequence of Lemma 4.3:

Proposition 4.12. *A monadic expansion tree I satisfies a subquery Q'' of a connected conjunctive query Q' iff it contains a node n whose IQ-type with respect to Q' covers Q''.*

Proof. Clearly if τ covers Q'', then a monadic expansion tree I having a node with type τ must satisfy Q''. Conversely, if I satisfies Q'', by Lemma 4.3 there is a set X, a node n, and a relative homomorphism h from (Q'', X) to n. Let $q_1 \dots q_n$ be the maximal connected components of Q'' relative to X. Then h also serves as a relative homomorphism of each (X, q_i) to n, while each q_i is closed and connected relative to X, hence is a valid type element. Thus $(X, q_1) \dots (X, q_n)$ witness that τ covers Q''. □

Thus types that cover Q' will be the "bad states" of our automaton.

Proof of Theorem 4.10. We denote by $\mathcal{Y}_{\neg Q'} \subseteq \mathcal{Y}$ the set of IQ-types τ with respect to Q' of nodes of some instances of $\mathcal{I}$ such that Q' is not covered by τ. We will construct a deterministic automaton $A_{Q'}$ whose set of states is $\mathcal{Y}_{\neg Q'}$. IQ-types not in $\mathcal{Y}_{\neg Q'}$ can be seen as leading directly to rejection (i.e., are not co-accessible), and thus need not be made explicit in a trimmed automaton. In the variant where we are given $\mathcal{Z} \supseteq \mathcal{Y}$, we will instead use the set $\mathcal{Z}_{\neg Q'} \supseteq \mathcal{Y}_{\neg Q'}$ of types τ in $\mathcal{Z}$ that do not cover Q'.

Since $\mathcal{I} \subseteq \mathcal{L}(d, \sigma)$, each node of an instance in $\mathcal{I}$ has at most d facts, each of which having at most $|\sigma|$ places. The transition function δ takes as input a list l of at most $d \cdot |\sigma|$ types in $\mathcal{Y}$ and a label $\hat{\alpha}$ in $\Gamma(d, \sigma)$. The result of $\delta(l, \hat{\alpha})$ is determined in the following manner. Let n be a node

in an instance I of $\mathcal{I}$ with output-isomorphism class α and having a list of children whose types match those of l. If there exists an X such that (X, Q') is satisfied by n, the transition is not defined. The outcome of the transition is the type τ of n, if τ does not cover Q'; otherwise, the transition is undefined. Lemma 4.8 guarantees that this transition function is well-defined: the type of a node only depends on the types of the children of this node and its output-isomorphism class. We set all states of the automaton to be final.

Let I be an arbitrary instance in $\mathcal{I}$. Let us show by induction on the structure of $\mathrm{Code}_d(I)$ that $A_{Q'}$ assigns a state τ to a node u of $\mathrm{Code}_d(I)$ if and only if Q' is not satisfied in the subinstance rooted at the corresponding node n of $\mathrm{T}(I)$ and that, in the case where a state τ is assigned, $\tau = \mathrm{type}_{Q'}(n)$. Assume this is true for all descendant of a node of u coding a node n in $\mathrm{T}(I)$. Assume now u is assigned state τ (and thus, τ does not cover Q'). It follows from the induction hypothesis that the children have as their state assignment in the coded instance their type, and by definition of the transition function, $\tau = \mathrm{type}_{Q'}(n)$. Since the subinstance rooted at n does not contain any node whose IQ-type covers Q', according to Proposition 4.12, it does not satisfy Q'. Conversely, the only case where a transition is not defined at a node u is when there exists an X such that (X, Q') is satisfied by a node n coded by u.

We prove the equivalence between acceptance by $A'_Q \cap A_{\mathcal{I}}$ and being the code of an instance of $\mathcal{I}$ not satisfying Q'. First, let I be a monadic expansion tree having its code $\mathrm{Code}_d(I)$ accepted by $A_{Q'}$ and $A_{\mathcal{I}}$. Thus I is in $\mathcal{I}$. Suppose by way of contradiction that I satisfies Q'. Due to Proposition 4.12, there exists a node n of $\mathrm{T}(I)$ whose type τ covers Q'. But then $A_{Q'}$ would have to reach a state corresponding to τ, which is impossible, since no such states are in $\mathcal{Y}_{\neg Q'}$, which is the set of states of our automaton. Conversely, let I be a monadic expansion tree in $\mathcal{I}$ not satisfying Q'. The inductive argument above shows that $A_{Q'}$ accepts I, and we know that $A_{\mathcal{I}}$ also does.

Complexity bounds. First, it is easy to check that the size of $A_{Q'}$ is in $O\left(|B(d,\sigma)| \times |\mathcal{Y}|^{O(d\cdot|\sigma|)}\right) = O\left(\chi^{d\cdot|\sigma|} \times |\mathcal{Y}|^{O(d\cdot|\sigma|)}\right) = O\left(|\mathcal{Y}|^{\mathrm{poly}(d,\sigma)}\right)$. Indeed, the size of the automaton is dominated by the size of its transition function. This transition function takes a sequence of states of the automaton and a symbol of the alphabet and returns a state of the automaton. The maximal length of the sequence is equal to the rank of the instance, which is bounded by $O(d \cdot |\sigma|)$. Moreover, the states of the automaton are the possible types of $\mathcal{I}$ which is equal to $\mathcal{Y}$. Finally, each symbol of the alphabet is an element of $\Gamma(d,\sigma)$, by definition. We finally obtain a bound of $O\left(|B(d,\sigma)| \times |\mathcal{Y}|^{O(d\cdot|\sigma|)}\right)$ for the size of the automaton.

We now turn to the claims about computation time. Let $\mathcal{Z}$ be a superset of the IQ-types satisfied in $\mathcal{I}$. The computation of $\mathcal{Z}_{\neg Q'}$ can be done by enumerating the types in $\mathcal{Z}$ and checking coverage of Q' by the type. The coverage check is polynomial in the type and Q'. Thus the total time is polynomial in the size of $\mathcal{Z}$, the maximal size of a type in $\mathcal{Z}$, and Q'. The maximal size of a type in $\mathcal{Z}$ is $O(2^{|Q'|})$. Thus the set of states $\mathcal{Z}_{\neg Q'}$ can be computed in the required time.

We now discuss the transition function. For each symbol $\hat{\alpha} \in \Gamma(d,\sigma)$, for each list of states $(v_1, \cdots, v_k)$, where k is in $O(d \times |\sigma|)$, we have to determine the type τ of a node in an instance I of $\mathcal{I}$ with a bag in the equivalence class $\hat{\alpha}$ and having children with IQ-types $\tau_1, \cdots \tau_k$. For one fixed representative of this output-isomorphism class, we let v be the output element of the node, B the set of facts, and we let v_i be the output element of each child, $1 \leqslant i \leqslant k$. We proceed by constructing a small monadic expansion tree to determine the type τ. This mini-instance has a node n with bag B and output element v (for technical reasons we cannot put n as the root of the monadic expansion tree since the root is assumed not to have an output element, but we can construct a dummy root node that serves no other purpose). We now explain how to construct the k children of node n.

For each $1 \leqslant i \leqslant k$, we build a subinstance I_i as follows: For each type-element (X_{ij}, q_{ij}) of τ_i, we build a mapping v_{ij} from q_{ij} as follows: for variable x appearing in q_{ij}, if x is in X_{ij} then x is mapped to v_i, otherwise the variable is mapped to a fresh value. We define I'_{ij} to be an instance formed by turning v_{ij} into a homomorphism. I_i is the union of I'_{ij} for all j. The subinstance I is the union of the I_i and B. We now define a tree decomposition of I. We let the root node have an empty bag and child n with output element v and bag B. The children of n are n_i for $1 \leqslant i \leqslant k$, where n_i has output element v_i and bag I_i. Clearly, $\hat{\alpha}$ is the code of the output-isomorphism class of n. Determining the type of Q can be done by simply enumerating all type elements of $\mathcal{Z}$ and determining which are satisfied by n, which can be done in time $O(|\mathcal{Z}| \cdot 2^{|Q'|} \cdot |I|^{|Q'|})$. The size of I is the size of B, plus an $O(d \cdot |\sigma| \cdot 2^{|Q'|} \cdot |Q'|)$. Finally, the whole process of constructing a small monadic expansion tree has to be performed for every possible transition, i.e., $O(|B(d,\sigma)| \times |\mathcal{Z}|^{O(d \cdot |\sigma|)})$ times. □

General UCQs. Theorem 4.10 restricted attention to connected CQs. Now consider the case where $Q' = \bigvee_i Q'_i$ is a general UCQ. For each i, Q'_i is a possibly disconnected CQ which we can write as $Q'_i = \bigwedge_{q'_{ij} \in \mathrm{MCC}(Q'_i)} q'_{ij}$. We first construct a deterministic automaton A_{ij} for each query q'_{ij} as above. Q'_i is not satisfied if and only if one of the q'_{ij} is not satisfied. Thus, $\bigcup_j A_{ij}$ is a nondeterministic ranked tree automaton that recognizes monadic expansion trees that do not satisfy Q'_i. Similarly, Q' is not satisfied if and only if none of the Q'_i is satisfied. Therefore if we consider $A_{Q'} = \bigwedge_i \bigvee_j A_{ij}$ and apply a product construction to the automata for each $\bigcup_j A_{ij}$, we obtain a nondeterministic ranked tree automaton recognizing monadic expansion trees that do not satisfy Q'. Thus, for any d-regular set of instances $\mathcal{I}$, applying a product construction to $A_{Q'} \cap A_{\mathcal{I}}$ gives an $\mathsf{NTA}_{\mathsf{Rk}}$ that accepts the set of all $\mathsf{Code}_d(I)$ with $I \in \mathcal{I}$ not satisfying Q'. The construction of the union of the deterministic automata as a nondeterministic one can be done in time polynomial in $|Q'|$ and in each of the automata (just do the union of the transitions). The construction of the intersection amounts to considering all possible combination of states and therefore raises the size of the automaton to an exponent of $|Q'|$. Thus we have a generalization of Theorem 4.10:

Theorem 4.13. *Let σ be a signature, d a positive integer, and $\mathcal{I}$ a d-regular set of monadic expansion trees recognized by the deterministic ranked tree automaton $A_{\mathcal{I}}$. Let Q' be an arbitrary UCQ.*

Then there exists a nondeterministic *ranked tree automaton $A_{Q'}$ over the alphabet $\Gamma(d,\sigma)$, such that $A_{Q'} \cap A_{\mathcal{I}}$ accepts the set of all $\mathsf{Code}_d(I)$ with $I \in \mathcal{I}$ not satisfying Q'. Moreover, the size of $A_{Q'}$ is in $O\left(|\mathcal{Y}|^{\mathrm{poly}(d,\sigma,|Q'|)}\right)$ where $\mathcal{Y}$ is the set of IQ-types with respect to Q' of nodes of some instances of $\mathcal{I}$.*

Finally, if one is given a superset $\mathcal{Z}$ of $\mathcal{Y}$, then we can compute $A_{Q'}$ in time $O\left(|\mathcal{Z}|^{\mathrm{poly}(d,|\sigma|,|Q'|)}\right)$.

4.2 IQ-Types and Containment of MDL in UCQs

How does the machinery of the last subsection help us? Let us first discuss how these arguments can be used to reprove the 2EXPTIME upper bound on the containment of MDL in UCQs [19].

To determine whether an MDL query Q is not contained in a UCQ Q', we can check whether there exists a monadic expansion tree satisfying Q and not satisfying Q' by Lemma 2.6. But we can actually restrict the search of a witness to a subset of the monadic expansion tree satisfying Q.

First, we observe that a monadic datalog program is equivalent to an infinite union of conjunctive queries, denoted by $\mathcal{UQ}(Q)$. Second, we can consider special cases of monadic expansion trees I of Q with the following property: there exists a conjunctive query q_1 in $\mathcal{UQ}(Q)$ and a surjective homomorphism from q_1 onto I. An instance having the previous property is said to *surjectively satisfy* Q. Indeed, a monadic expansion tree I of Q that does not satisfy Q' has to satisfy some CQ q_1 in $\mathcal{UQ}(Q)$, and all facts not necessary for the homomorphism from q_1 to I can be removed from I while keeping the monadic expansion tree structure and the non-satisfaction of Q'.

There exists an integer d polynomial in Q such that the codes of monadic expansion trees surjectively satisfying Q are in $\mathcal{L}(d, \sigma)$. More precisely, the set of monadic expansion tree models surjectively satisfying Q is regular and there exists a tree automaton A_Q of size exponential in d and in the sizes of σ and Q. This is an argument very similar to Proposition 5.9 of [19], except for the difference in coding of the instances (the proof trees of [19] encoding their expansion trees vs our coding of monadic expansion trees), which is not essential for the proof.

We can then apply Theorem 4.13 to compute $A_{Q'}$ and test the emptiness of $A_Q \cap A_{Q'}$ in polynomial time in the size of the automata. In the general case of MDL containment in UCQs, if we take $\mathcal{Z}$ to be all IQ-types, we obtain a doubly exponential bound on the construction of $A_{Q'}$, yielding an overall 2EXPTIME upper bound on the containment of MDL within a UCQ, as in [19].

We now introduce a special property of classes of instances that allow us to take $\mathcal{Z}$ to be considerably smaller, yielding a better upper bound.

4.3 Unique Mapping Condition

As we shall see in Section 5, the 2EXPTIME upper bound obtained from our method for the containment of arbitrary MDL query in UCQs 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 4.14. For any conjunctive query Q', a class of monadic expansion trees $\mathcal{I}$ satisfies the Q'-Unique Mapping Condition (Q'-UMC) if the following holds for any node n in an instance of $\mathcal{I}$:

> ($\star$) for any type elements (X_1, q_1) and (X_2, q_2) of Q' satisfied at n, if there exists an atom A appearing in both q_1 and q_2 referencing a variable x in both X_1 and X_2 then $X_1 = X_2$ and $q_1 = q_2$.

A class of monadic expansion trees has the Universal Unique Mapping Condition ($\forall$-UMC) if it has the Q'-UMC for every conjunctive query Q'.

The idea behind the name Unique Mapping Condition is that type elements represent ways in which Q' can be partially mapped into an expansion tree. The UMC says that such mappings are determined once we know one atom of Q' in the domain. The Q'-UMC should shed more light on the connectedness and maximality requirements imposed on IQ-types. 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.

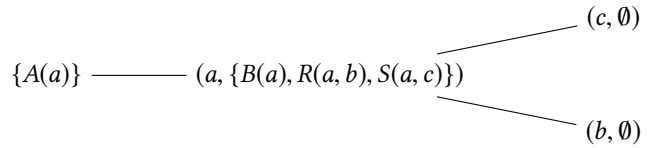
Example 4.15. In the definition of IQ-types (Definition 4.5), we required the subquery to be a relative connected component: relatively closed and relatively connected. This restriction is important for making the UMC non-trivial, and we illustrate this via two examples.

Consider the CQ $Q'_1 = \exists x \exists y \exists z\, R(x, y) \wedge R(y, z)$ and a monadic expansion tree I_1, with $\mathrm{T}(I_1)$ as follows (with the root to the left):

$$\{A(a)\} \text{ ——— } (a, \{R(a, b)\}) \text{ —— } (b, \{R(b, c)\}) \text{ ——— } (c, \emptyset)$$

Consider $X = \{x\}$ and the following subquery of Q'_1: $q_1 = \exists x \exists y\, R(x, y)$. Both (X, Q'_1) and (X, q_1) have a relative homomorphism to the subinstance rooted at the node of $\mathrm{T}(I)$ with bag $\{R(a, b)\}$, and clearly they are different. But this does not contradict the UMC, since while the pair $(X, \{R(x, y), R(y, z)\})$ is a type element, the pair $(X, \{R(x, y)\})$ is not, since $R(x, y)$ is not closed relatively to X: it has one atom containing y, and thus must have the other such atom in order to be relatively closed.

Now consider $Q'_2 = \exists x \exists y \exists z\, B(x) \wedge R(x, y) \wedge S(x, z)$, and a monadic expansion tree I_2, with $\mathrm{T}(I_2)$ as follows:



Let n be the node of $\mathrm{T}(I)$ whose output element is a. We set $X = \{x\}$, and consider two subqueries of Q_2': $q_{21} = \exists x \exists y\, B(x) \wedge R(x, y)$ and $q_{22} = \exists x \exists z\, B(x) \wedge S(x, z)$. Both (X, q_{21}) and (X, q_{22}) have a relative homomorphism to the subinstance rooted at n in I_2, and clearly they are different. But this does not contradict the UMC, since neither $(\{x\}, \{B(x), R(x, y)\})$ nor $(\{x\}, \{B(x), S(x, z)\})$ is a type element. In this case, the reason is that neither subquery is connected relative to X. The definition of relative connectedness would require the existence of variables x_1 in $B(x)$ and x_2 in $R(x, y)$ that are not in $\{x\}$, and that are connected by some path. But clearly such variables cannot exist, since the only variable of $B(x)$ is x.

Above we have given examples that have the UMC, due to the restrictions we have placed on being a type element. We now show that even with these restrictions, it is possible for the UMC to fail.

Example 4.16. Consider a monadic expansion tree of the following form (with the root to the left):

$$(a, \{R(a, b), R(b, a), C(a)\}) \text{ —— } (b, \{R(b, c), R(c, b), B(b)\}) \text{ ———— } (c, \{C(c)\})$$

This is a valid monadic expansion tree for the MDL program:

$$P_1(x) \leftarrow R(x, y) \wedge R(y, x) \wedge C(x) \wedge P_2(y)$$
$$P_2(x) \leftarrow R(x, y) \wedge R(y, x) \wedge B(x) \wedge P_3(y)$$
$$P_3(x) \leftarrow C(x)$$

Consider again the query

$$Q' = \exists x \exists y \exists z\, C(x) \wedge R(x, y) \wedge R(y, z) \wedge C(z)$$

from Example 4.6 along with the two pairs that were shown in Example 4.6 to be type elements of Q':

$$T_1 = (\{x, z\}, \{R(x, y), R(y, z)\})$$
$$T_2 = (\{x\}, \{R(x, y), R(y, z), C(z)\})$$

We claim that these two type elements witness the fact that the expansion tree above does not satisfy the Q'-UMC.

Indeed, note that both T_1 and T_2 are satisfied at the root of the expansion tree. In the case of T_1, the relative homomorphism witnessing this maps both z and x to the value a while mapping y to b. For T_2 the relative homomorphism maps x to a, y to b, and z to c (remember that z cannot be mapped to a in the case of T_2 as $\{x\}$ is by definition the set of variables mapped to a).

Now, observe that T_1 and T_2 have an atom in common, and yet differ in their set of variables and their set of atoms. Thus they witness the failure of the Q'-UMC.

The takeaway here is that the presence of symmetries in both Q' and the expansion tree can cause the UMC to fail.

The UMC is a semantic property which is not straightforward to check. However, in Sections 4.4 and 4.5, we will see two cases where the UMC holds: first, by restricting the classes of instances

we work with (to trees over specific signatures), and second by restricting the MDL program we consider monadic expansion trees of.

We will show that the Unique Mapping Condition suffices to get better upper bounds. The interest of the UMC is that it bounds the number of useful types:

Proposition 4.17. *Let Q' be a UCQ. There exists a set $\mathcal{Z}$ constructible in time $O(2^{\text{poly}(|Q'|)})$, such that for every set $\mathcal{I}$ of monadic expansion trees satisfying the Q'-UMC:*

$$\mathcal{Z} \supseteq \bigcup_{I \in \mathcal{I}} \bigcup_{n \in \mathrm{T}(I)} \text{type}_{Q'}(n).$$

In particular, the size of $|\mathcal{Z}|$ is in $O(2^{\text{poly}(|Q'|)})$.

Proof. Let n be a node of an instance in $\mathcal{I}$ satisfying ($\star$) and $\tau = \text{type}_{Q'}(n)$.

By ($\star$), for every occurrence of a variable x within an atom A of Q', there can be at most one type element of the form (X, q) with A in q and $x \in X$. In other words, the number of type elements of the form (X, q) *with X non-empty* is bounded by the size of Q'. Furthermore, the number of type elements of the form $(\emptyset, q)$ is at most the number of maximal connected components of Q', i.e., also bounded by $|Q'|$.

In other words, choosing an element of $\mathcal{Z}$ requires to choose at most $2|Q'|$ elements among $O(2^{\text{poly}(|Q'|)})$, i.e., there are at most $O(2^{\text{poly}(|Q'|)})$ elements in $\mathcal{Z}$.

$\mathcal{Z}$ can be constructed in $O(2^{\text{poly}(|Q'|)})$ as well, for instance by first enumerating all possible type elements of Q' (there are exponentially many pairs (X, q) with X a subset of the variables of Q' and q a subquery of Q', and one can check in polynomial type that such a pair is a type element), and then constructing all types over these type elements that satisfy the constraints above. □

Together with Theorem 4.13, this proposition implies that one can compute in time exponential in d, $|\sigma|$, and $|Q'|$ a deterministic ranked tree automaton such that the intersection with $A_{\mathcal{I}}$ recognizes the codes of the instances in $\mathcal{I}$ that do not satisfy Q'. Because the intersection of two tree automata is polynomial in their sizes and the emptiness of a tree automaton can be checked in linear time in its size, we conclude:

Corollary 4.18. *Let σ be a signature, d a positive integer, and Q' a UCQ over σ. Let $\mathcal{I}$ be a d-regular set of monadic expansion trees over σ with the Q'-UMC and recognized by the tree automaton $A_{\mathcal{I}}$. Then checking if Q' holds over all $I \in \mathcal{I}$ can be done in $O(|A_{\mathcal{I}}| \times 2^{\text{poly}(|Q'|, d, |\sigma|)})$.*

4.4 Upper Bounds for Tree Validity

We now present our first application of the UMC and IQ-type technology developed over the last few sections. It concerns the problem of tree validity introduced in Section 2.3, over the signatures $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$, $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1,Ch2}}$, and $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1,Ch2,Child}}$.

A tree I over one of these relational signatures can be associated with the bag and output element structure of a monadic expansion tree $\mathrm{T}(I)$ in a canonical way, up to sibling ordering, as follows:

(1) For each vertex v in I, there is a node n_v whose output element is v.
(2) For each node $n_v \in \mathrm{T}(I)$, $\text{bag}(n_v)$ contains all unary facts about v, as well as every fact of the form $\mathsf{Child}(v, x)$, $\mathsf{FirstChild}(v, x)$, $\mathsf{SecondChild}(v, x)$.
(3) Two nodes n_{v_1} and n_{v_2} of $\mathrm{T}(I)$ are in parent-child relation iff $\mathsf{Child}(v_1, v_2)$, $\mathsf{FirstChild}(v_1, v_2)$, or $\mathsf{SecondChild}(v_1, v_2)$ holds in I.
(4) Children of a given node $n_v \in \mathrm{T}(I)$ are ordered in an arbitrary manner.

It is easy to verify that $(I, \mathrm{T}(I))$ is a monadic expansion tree. It has at most 4 facts per node for $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1,Ch2}}$ (the root bag contains one FirstChild, one SecondChild, one Label, and one Root fact), 6 for $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1,Ch2,Child}}$ (same plus two Child facts), and $r + 2$ for $\mathcal{S}^{\text{unranked}}_{\mathsf{Child}}$ where r is the rank of the tree.

We will show that the collection of such trees is an instance class that satisfies the UMC:

Theorem 4.19. *Let σ be any of $\mathcal{S}^{\text{unranked}}_{\text{Child}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$. The collection of monadic expansion trees $(I, \mathrm{T}(I))$, where I is a tree over σ, satisfies the $\forall$-UMC.*

Towards proving the theorem, first note the following:

Lemma 4.20. *Let σ be any of $\mathcal{S}^{\text{unranked}}_{\text{Child}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$. Let q be a connected conjunctive query over σ, and I a tree over σ. Suppose h_1 and h_2 are two homomorphisms from q to I, and x and y are variables of q. Assume $h_1(x) = h_2(x) = r$ for some vertex r of I, and that all variables of q are mapped by h_1 and h_2 to the subtree of I rooted at r. If $h_1(y) = r$ then $h_2(y) = r$.*

Proof. If $y = x$, this is trivial.

Since q is connected and $y \neq x$, there is a non-empty path of binary atoms $A_1 \dots A_n$ and a sequence of variables $x = x_0, \dots x_n = y$ with, for all $1 \leqslant i \leqslant n$, x_{i-1} and x_i distinct variables co-occurring in A_i. Since the A_i's are binary, they have to be FirstChild, SecondChild, or Child atoms; let us denote R_i the corresponding relation name. For $0 \leqslant i \leqslant n$, let d_i be the depth of $h_1(x_i)$ in the subtree of I rooted at r. We know that $d_0 = 0$, and, for all $1 \leqslant i \leqslant n$, we can compute d_i from d_{i-1}:

- if $A_i = R_i(x_{i-1}, x_i)$, $d_i = d_{i-1} + 1$;
- if $A_i = R_i(x_i, x_{i-1})$, $d_i = d_{i-1} - 1$.

But the depth d'_i of $h_2(x_i)$ in the subtree of I rooted at r is computed in exactly the same way: $d'_0 = 0$ and the same recurrence formulas as above hold. This means in particular that $d'_n = d_n = 0$. Since r is the only node at depth 0 in the subtree of I rooted at r, $h_2(y) = r$. □

Note that this lemma does not hold over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$. We are now ready to prove Theorem 4.19:

Proof of Theorem 4.19. Let Q' be an arbitrary conjunctive query, I an instance of σ, and n a node of $\mathrm{T}(I)$, with output element the corresponding tree vertex v_n. Let (X_1, q_1) and (X_2, q_2) be two type elements of Q' in $\text{type}_{Q'}(n)$, with respective relative homomorphisms h_1 and h_2. We assume there exists an atom A in both q_1 and q_2 referencing a variable in both X_1 and X_2. We need to show that $X_1 = X_2$ and $q_1 = q_2$.

We first show $q_1 = q_2$. Let X be the intersection of X_1 and X_2. Let $q_\cap$ be the intersection of q_1 and q_2 (i.e., the atoms in common), and q' the maximal connected component of $q_\cap$ relative to X that contains A.

By way of contradiction, assume $q_1 \neq q_2$. Without loss of generality, since q_1 and q_2 play the same role, we can assume that there exists an atom A' of q_1 that is not in q_2. Since q_1 is connected relative to X_1, we know there is a path consisting of variables not in X_1 such that consecutive variables x_i, x_{i+1} co-occur in an atom A_i of q_1, with the path starting at a variable of A and terminating at a variable of A'. We now consider j to be the least index such that the atom A_{j+1} is not in q_2, and such that all variables $x_1, \dots, x_j$ are not in X_2. Thus, A_j, which contains variable x_{j+1} is in both q_1 and q_2, and it is also in q' since it is connected to A relative to $X_1 \cap X_2$ in $q_\cap$. Note that since $x_{j+1} \notin X_1$, $h_1(x_{j+1}) \neq v_n$, by the definition of a relative homomorphism.

Suppose first that $h_2(x_{j+1}) = v_n$, which means $x_{j+1} \in X_2$. Let x be any variable in $X_1 \cap X_2$ (which is non-empty, since q_1 and q_2 share a non-empty atom). Thus $h_1(x) = h_2(x) = v_n$. Note that both h_1 and h_2 restrict to homomorphisms of the connected query q', and these restrictions contain x and x_{j+1}. Applying Lemma 4.20 above to the query q', we derive that $h_1(x) = h_1(x_{j+1}) = v_n$ which is a contradiction of the fact noted immediately above. Thus we can assume $h_2(x_{j+1})$ is not v_n. Hence

x_{j+1} is not in X_2, by definition of relative homomorphism. But then, by closedness of the connected component q_2, A_{j+1} should be in q_2, which leads to a contradiction.

Thus we completed the argument that q_1 is equal to q_2.

A similar argument shows that X_1 must be equal to X_2, completing the uniqueness argument. As above, let x be a variable of $X_1 \cap X_2$. Let y be a variable of X_1. Then $h_1(y) = v_n = h_1(x)$. Moreover, $h_2(x) = v_n$. Since h_1 witnesses that (X_1, q_1) is a type element satisfied in n, h_1 must map into the subinstance rooted at n. Similarly, h_2 must map into the subinstance rooted at n. By Lemma 4.20 above, applied to $q_1 = q_2$ we infer $h_2(x) = h_2(y)$, and thus $h_2(y) = v_n$. Thus y is in X_1. With the same reasoning, we can demonstrate that $X_2 \subseteq X_1$. Thus X_1 is equal to X_2.

□

From the theorem above and Corollary 4.18, we get the following new bound, which in particular (for $\mathcal{S}^{\text{unranked}}_{\text{Child}}$) answers a question left open from [8].

Corollary 4.21. *Validity of a UCQ over $\mathcal{S}^{\text{unranked}}_{\text{Child}}$, $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$, or $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ with respect to a tree automaton is in* EXPTIME.

Proof. Let us first consider the ranked case, i.e., $\sigma = \mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ or $\sigma = \mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$. We pose $d = 4$ in the former case, $d = 6$ in the latter.

Let Q be a UCQ over σ, and $A = (\Omega, \Delta_0, \Delta, F)$ a binary tree automaton over σ.

From A, we build a binary tree automaton $A' = (\Omega, \Delta'_0, \Delta', F)$ over the alphabet $\Gamma(d, \sigma)$ as follows:

- For every $(\alpha, q) \in \Delta_0$, let $c \in B(d, \sigma)$ be the output-isomorphism class of a bag with output element v and facts $\mathsf{Leaf}(v)$ and $\mathsf{Label}_\alpha(v)$. We then add $(\hat{c}, q)$ to Δ'_0;
- For every $(q_1, q_2, \alpha, q) \in \Delta_0$, let $C \subseteq B(d, \sigma)$ be the output-isomorphism classes of bags with output element v, facts $\mathsf{FirstChild}(v, v_1)$, $\mathsf{FirstChild}(v, v_2)$, $\mathsf{Child}(v, v_1)$ (if $\sigma = \mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$), $\mathsf{Child}(v, v_2)$ (if $\sigma = \mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$), $\mathsf{Label}_\alpha(v)$, and a possible additional fact $\mathsf{Root}(v)$. There are exactly two elements in C. For either of these elements $c \in C$, we add $(q_1, q_2, \hat{c}, q)$ and $(q_2, q_1, \hat{c}, q)$ to Δ'.

By construction, A' recognizes the set $\mathsf{Code}_d(I)$ for I a tree accepted by A. Additionally, A' can be constructed in time linear in A.

We now apply Corollary 4.18 to σ, d, Q', and the class of trees over σ whose monadic expansion tree representation is recognized by A', which has the Q'-UMC by Theorem 4.19. We obtain that checking if Q' holds over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ and $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ can be done in time $O(|A'| \times 2^{\text{poly}(|Q'|, d, |\sigma|)}) = O(|A| \times 2^{\text{poly}(|Q'|)})$ since d and σ are fixed. In other words, it is in EXPTIME.

Finally we deal with the unranked case. We make use of the following easily-verified fact about an automaton A over unranked trees: given a tree t that is accepted by A, there is a subtree t' (obtained by removing subtrees in t) which is a ranked tree of rank at most polynomial in A, such that t' is also accepted by A. Applying this to a counterexample t to validity, we see that t is still a counterexample to validity, since trimming preserves the negation of Q. Thus we have reduced to the ranked case. □

Corollary 4.21 complements the theorem of Björklund et al. [9], which shows that the problem of validity with respect to an NTA is hard for EXPTIME.

In the case of validity over all trees for $\mathcal{S}^{\text{unranked}}_{\text{Child}}$, we can do better:

Proposition 4.22. *Validity of a UCQ over $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ with respect to all trees is in* PSPACE.

Proof. Observe that a counterexample to validity for $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ can always be taken to be a single path with no branching, by keeping one arbitrary branch from the root to a leaf: indeed, removing

other branches amount to simply removing facts from the relational instance and therefore cannot make a CQ true if it did not hold on the original instance (we cannot, on the other hand, shorten a branch, as this amounts to adding new Leaf facts).

We can further assume each CQ in the UCQ is connected, since we can guess a connected component of each CQ. A connected CQ specifies an interval of polynomial size in the path. Thus we need a nondeterministic PSPACE algorithm which determines whether there is a path in which a set S of k-sized intervals are omitted. We can do this by tracking the last k elements observed. □

4.5 Upper bounds for restricted MDL queries

We will apply the UMC to get another exponential decrease in the complexity. This time we deal with general structures, but restrict the queries.

A monadic datalog query 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.

GEMDL and AGEMDL are simple syntactic restrictions of MDL whose properties will allow us to apply the UMC. The main application of these restrictions are the ones that come from limited access querying, in the specific case where access methods have at most one output position. Indeed, consider again the rules in Equation (2) in Section 2.4, an encoding as an MDL program of the unary predicate AccValues representing values that can be discovered via the access restrictions. When there is at most one output position for every access method, there is only one rule for every extensional predicate R, of the form:

$$\mathsf{AccValues}(x_{j_k}) \leftarrow R(\mathbf{x}) \wedge \mathsf{AccValues}(x_{j_1}) \wedge \cdots \wedge \mathsf{AccValues}(x_{j_m})$$

These rules satisfy the GEMDL restriction.

Furthermore, evaluation of UCQ Q over accessible data can be done using a single additional predicate (see Equation (1) in Section 2.4), with rules for every conjunct of Q, also enforcing all variables are accessible.

Combining the GEMDL rules defining AccValues and the rules defining the goal predicate thus results in an AGEMDL query. It follows that the question of containment of unions of conjunctive queries under limited access patterns (recall Definition 2.12), where every access has a single output position, can be expressed as the containment of an AGEMDL query in a UCQ. Formally:

Proposition 4.23. *There is a polynomial-time reduction from containment of UCQs under limited access patterns, where every access has at most one output position, to containment of AGEMDL in UCQ.*

We will get an EXPTIME bound for AGEMDL.

The key feature common to containment of GEMDL queries and the limited access containment problem is that counterexamples have a stronger kind of tree-like instance, compared to simply monadic expansion trees. A *diversified tree-like instance* is a monadic expansion tree in which: (i) for each node n which is not the root, for each relation R, there exists at most one fact in bag(n) having the relation name R; (ii) there is no value v that appears at the same position in two distinct facts with the same relation name.

The first condition is a local one, saying roughly that 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 Fig. 1. The large

ellipses represent bags, with different relations represented by different shapes in a bag. The dark circle represents a common value shared across and within bags.

We now have the following refinement of Proposition 2.6. Let Q be a union of conjunctive queries.

Proposition 4.24. *If an AGEMDL query Q is not contained in a UCQ Q', then there is a* diversified *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.*

Proof. We reuse the construction presented in the proof of Proposition 2.6. We just have to show the additional properties of a diversified tree-like instance.

Recall from Proposition 2.6 that our tree-like instance was derived from a graph $G(Q)$, whose nodes consisted of pairs, a head atom and a rule. Furthermore:

(†) $G(Q)$ has no two nodes n and n' with the same head atom $\alpha(x)$, where x is a variable.

There exists an isomorphism ψ from the facts in the rules of $G(Q)$ to elements of I. For each atom α appearing in a rule within a node m of $G(Q)$ such that y is the variable appearing in the head atom of m, $\psi(\alpha)$ appears in the node of $T(I)$ having $\psi(y)$ as its output element.

(1) Let R be a relation name. Due to Property (†), for each intensional predicate P and for each variable x, there is at most one node of $G(Q)$ having $P(x)$ as head atom. Due to the AGEMDL restriction, a rule associated with a non-root node has at most one atom in the body with relation R. Thus, by construction of $T(I)$, the bag associated with each node has at most one atom with relation R.

(2) Let $R(\bar{a}_1)$ and $R(\bar{a}_2)$ be two facts of I in two distinct non-root nodes n and n' within $T(I)$, which share the value v. By definition of a tree-like instance, and by the fact proved above, we must have that n' and n of $T(I)$ are in a parent-child relationship. We assume that $R(\bar{a}_1)$ is in n' and $R(\bar{a}_2)$ is in n. Let v and v' denote the output elements of n and n' respectively, and let m and m' be the corresponding nodes of $G(Q)$. We prove by contradiction that v does not appear in the same position of $R(\bar{a}_1)$ and $R(\bar{a}_2)$. We denote by x the variable mapped to v by ψ and by x' the variable mapped to v' by ψ. Let ρ be the (necessarily unique) rule of Q having an atom with the relation name R. There exist atoms α_m in the body of the rule of m and $\alpha_{m'}$ in the body of the rule associated to m' such that $\psi(\alpha_m) = R(\bar{a}_2)$ and $\psi(\alpha'_m) = R(\bar{a}_1)$. Because v is the output element of n, by the second property of ψ listed above, x must appear in the head atom of m, and similarly x' must appear in the head atom of m'. Both m and m' have relation R in the associated rule, and thus by the AGEMDL property, the rule must be the same, namely ρ. Because v appears in $R(\bar{a}_1)$ and $R(\bar{a}_2)$ at the same position and $\psi(x) = v$, x appears in the same position in the atoms α_m and $\alpha_{m'}$. Thus $x' = x$, and therefore m and m' have the same head atoms, as well as the same rules, hence they must be the same node of $G(Q)$. This contradicts the assumption that n and n' were distinct.

□

The technique generalizes to the containment problems arising from general access methods, not only ones with a single output:

Proposition 4.25. *If a UCQ Q is not contained in a UCQ Q' under access restrictions ACS then there is a diversified tree-like instance I such that I satisfies $Q \wedge \neg Q'$.*

Proof. The rules that come from limited access patterns with an arbitrary number of output positions satisfy the following weakening of the AGEMDL condition: for any two rules that share an extensional predicate in the body, the bodies are identical. One can easily see that the argument

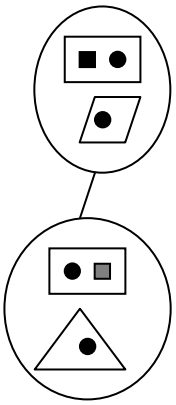


Fig. 1. Diversified tree-like instance

in the proof of Proposition 4.24 applies to show that these rules also admit diversified tree-like instances. □

The following links diversified instances to the IQ-type machinery developed previously:

Theorem 4.26. *The class of diversified tree-like instances satisfies the* $\forall$*-UMC.*

Proof. We need to show that the class of diversified tree-like instances satisfies the Q'-UMC for any conjunctive query Q'. Let Q' be such a query, I a diversified tree-like instance, and n a node of $\mathrm{T}(I)$.

We will show the following:

> (‡) Let X be a subset of variables of Q' and q a subquery of Q'. Take any $x \notin X$ of q and atom α of Q' but not of q that contains x, any relative homomorphism h from (X, q) to the subinstance rooted at n. Then there is at most one way of extending h into a homomorphism h^* from $q^* = q \wedge \alpha$ to the subinstance rooted at n, and of extending X into a superset X^*, such that h^* is a relative homomorphism from (X^*, q^*) to the subinstance rooted at n.

We now prove (‡). We let X, q, x, α, h to be as in the statement of (‡). Let β be an atom of q that contains x; β exists since x is a variable of q. Let n' be the node of the subinstance rooted at n whose bag contains $h(\beta)$. We distinguish two cases:

(1) $h(x)$ is the output element v of n'. Since $x \notin X$, $n' \neq n$. Since α contains x, it is clear $h^*(\alpha)$ can only be mapped to the bag of n' or to that of its parent n''. It cannot be mapped to both, as this would imply that there are two distinct facts with same relation name as α that both contain v in the same position (the position of x in α), which is forbidden in diversified tree-like instances. Within the only possible bag, there cannot be more than one possible way to map α, as otherwise there would be two facts on the same relation within the same bag, which is also forbidden in diversified tree-like instances.
(2) $h(x)$ is not the output element of n'. Then $h(x)$ must be the output element of one of the children n'' of n'. We then proceed exactly as in the previous case, the roles of n' and n'' being exchanged.

This completes the argument for the uniqueness of h^*. We turn to the second statement of (‡), concerning uniqueness of X^*. Here we observe that if such an h^* can be constructed, X^* is

necessarily the set of variables mapped by h^* to the output element of n, by the definition of relative homomorphism. This concludes the proof of (‡).

We now explain how (‡) can be used to prove that the property (★) in the definition of the unique mapping condition holds.

Let (X_1, q_1) and (X_2, q_2) be two type elements of Q' satisfied by n that share an atom α referencing a variable x in $X_1 \cap X_2$. By definition, there exist a relative homomorphism h_1 from (X_1, q_1) to the subinstance rooted at n and a relative homomorphism h_2 from (X_2, q_2) to the subinstance rooted at n. Since α references a variable mapped by h_1 to the output element of n, $h_1(\alpha) \in \text{bag}(n)$; similarly, $h_2(\alpha) \in \text{bag}(n)$. Now, since I is a diversified tree-like instance, there can only be one fact in $\text{bag}(n)$ having same relation name as α, so $h_1(\alpha) = h_2(\alpha)$. In particular, none of the variables of α except those in $X_1 \cap X_2$ can be mapped to the output element by h_1 or h_2.

Now, assume by way of contradiction that $q_1 \neq q_2$ or $X_1 \neq X_2$. Then, without loss of generality since q_1 and q_2 play symmetrical roles, we can assume there exists an atom β in q_1 such that either β is not in q_2 or $h_2(\beta) \neq h_1(\beta)$. Since q_1 is connected relative to X_1, there is a path $x_1 \dots x_k$ of variables not in X_1 from a variable x_1 of α to a variable x_k of β in the graph of co-occurrences of variables in atoms of q_1. Let $\gamma_1 = \alpha, \gamma_2 \dots \gamma_{k+1} = \beta$ be the corresponding atoms on this path and let γ_{i+1} with $1 \leqslant i \leqslant k$ be the first atom on this path such that either γ_{i+1} is not in q_2 or $h_2(\gamma_{i+1}) \neq h_1(\gamma_{i+1})$. Let X be the subset of variables of $q = \gamma_1 \wedge \dots \wedge \gamma_i$ that are mapped to the output element of n both by h_1 and by h_2, i.e., that are in $X_1 \cap X_2$. Since q_2 is closed relative to X_2 and $x_i \notin X_2$, it means that γ_{i+1}, which also contains x_1, is in q_2. We thus must have $h_2(\gamma_{i+1}) \neq h_1(\gamma_{i+1})$. We then apply (‡) to X, q, x_i, γ_{i+1}: indeed, $x_i \notin X$ since $x_i \notin X_1$. But then, the restriction h of h_1 to the variables of q (which is also the restriction of h_2 to the variables of q) is a relative homomorphism from (X, q) to the subinstance rooted at n. Therefore, (‡) tells us that there is at most one way to extend h and X into h^* from $q^* = q \wedge \gamma_{i+1}$ to the subinstance rooted at n and of extending X into a superset X^* such that h^* is a relative homomorphism from (X^*, q^*) to the subinstance rooted at n. But the restriction of h_1 to the variables of q^*, and the restriction of h_2 to the variables of q^* are two relative homomorphisms (for some sets $X_1' \subseteq X_1$, $X_2' \subseteq X_2$) that also extend h in the same sense. They must thus coincide and $h_1(\gamma_{i+1}) = h_2(\gamma_{i+1})$, which is a contradiction. □

Note that an atom α determines a unique set of variables X, so the key is that, in this restricted setting, we have only one way to select which variables map to the output element of the root.

The last thing we need to apply our UMC machinery to diversified instances is to note that this class of tree-like instances can be captured with an exponential-sized automaton:

Lemma 4.27. *The set of diversified tree-like instances satisfying a UCQ Q is d-regular for some d. Further there is a ranked deterministic tree automaton A_Q recognizing the codes of the diversified tree-like instances satisfying Q which has size exponential in Q and can be constructed in time exponential in Q.*

Proof. First, we argue that the set of monadic expansion trees satisfying a UCQ Q is easily recognizable by an exponential sized automaton. The properties of a monadic expansion tree are easily enforced by the transition function of the automaton. The property of satisfying a UCQ is checked by having a state for each subquery Q_0 of Q supplemented with a homomorphism from a subset of the variables in Q_0 to the current node. Updating the state can be done with an exponential sized transition function, and the automaton accepts if the full query Q is covered. The property of being diversified is very simple to check, and the intersection of these two automata can be formed in polynomial time. □

From this lemma, Theorem 4.26, and Proposition 4.17, the number of IQ-types of the set of diversified instances satisfying Q is bounded by an exponential function. Moreover, from Corollary 4.18 we get complexity bounds for AGEMDL and limited access containment:

Corollary 4.28. *The containment of an AGEMDL query in a UCQ can be decided in* EXPTIME. *The containment of two UCQs under limited access constraints can be decided in* EXPTIME.

5 LOWER BOUNDS

We will now prove our lower bound results. Again, there is a tight connection between MDL containment and tree validity problems. We will begin by showing lower bounds for the tree validity problem, and then use these to get results for MDL and limited access containment.

5.1 Lower bounds for tree validity problems

We first prove lower bounds for tree validity problems matching the upper bounds of Section 4.4. For generality, we show that the lower bounds hold for BDTDs (recall that they are more restrictive than BNTAs) by first proving them for BNTAs and then applying the following lemma, which shows that one can reduce NTA problems to DTD problems, by making runs explicit. This result was stated by Björklund et al. (Lemma 18 of [10]) for unranked trees, who mention that it is implicit in Takahashi's work [32]. As in [10], we define the *annotated tree language* of a BNTA A with states Ω over binary trees labeled with Λ as the set of trees in $L(A)$ that are annotated by their accepting runs. More formally, the annotated tree language of A is the set of trees t over $\Lambda \times \Omega$ such that $\Pi_\Lambda(t) \in L(A)$ while $\Pi_\Omega(t)$ is an accepting run of A on $\Pi_\Lambda(t)$.

Lemma 5.1. *Let A be a* BNTA *over binary trees with a single final state, and such that all trees accepted by A have a common root label. Then there exists a* BDTD D_A*, constructible in polynomial time from A, that recognizes the annotated binary tree language of A.*

Proof. Since our notion of DTDs is slightly different from the classical one (in particular, this makes the construction cubic, not quadratic as in [10]), and since the statement of the result in [10] misses the technical condition of imposing a single possible root label, we give the construction explicitly.

Let A be a BNTA with alphabet Λ, states Ω, final state q_f, input states $\Delta_0 \subseteq \Lambda \times \Omega$, and transition relation $\Delta \subseteq \Omega^2 \times \Lambda \times \Omega$. Let a_r be the common root label of all trees accepted by A. We construct the BDTD $D_A = (d, l_0)$ over the alphabet $\Lambda \times \Omega$ as follows (note that this construction is cubic in the size of A):

- $l_0 = (a_\mathrm{r}, q_\mathrm{f})$;
- For $\alpha \in \Lambda$ and $q \in \Omega$,

$$d(\alpha, q) = \{((\beta_1, q_1), (\beta_2, q_2)) \mid \beta_1, \beta_2 \in \Lambda, (q_1, q_2, \alpha, q) \in \Delta\} \cup \{\varepsilon \mid (\alpha, q) \in \Delta_0\}$$

It is clear that a binary tree t is accepted by D_A if and only if $\Pi_\Omega(t)$ is an accepting run of A on $\Pi_\Lambda(t)$. □

Thus, as in [10], if we show that the validity problem is hard for BNTAs (even when all trees have a common root label), we can deduce it is also hard for BDTDs using this reduction:

Corollary 5.2. *Let Q be a UCQ on $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$, $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child}}$, or $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ and A a* BNTA*s such that all trees accepted by A have a common root label. One can construct in polynomial time a UCQ Q' on the same signature and a* BDTD D'_A *such that Q' is valid over D'_A if and only if Q is valid over A. Furthermore, if Q is a CQ, so is Q'.*

Proof. We would like to apply Lemma 5.1 to get the DTD and then rewrite the query appropriately. We have two difficulties: first, we need to deal with the case where A does not have a single final state, since Lemma a 5.1 requires this. Second even in the case where we have a single final state, and hence could apply Lemma 5.1 and take $D'_A := D_A$, we have a difficulty translating from a CQ to another CQ, since we would need disjunction to express the different possible labels of the annotated tree that correspond to one label of the regular tree.

We take care of the final state problem first. We modify $A = (\Omega, \Delta_0, \Delta, F)$ to $A' := (\Omega \cup \{q_f\}, \Delta_0, \Delta \cup \Delta', q_f)$ where q_f is a fresh state and $\Delta' := \{(q_1, q_2, \alpha, q_f) \mid (q_1, q_2, \alpha, q) \in \Delta, \alpha \in \Lambda, q \in F\}$. A' has a single final state, is constructible in time linear in A, and Q is valid over A' if and only if Q is valid over A.

To address the second problem, we make the different annotations structurally adjacent to the labels of the regular tree, rather than being extended labels. This technique is similar to that used in the proof of Theorem 19 of [10] in the case of unranked trees. We construct the BDTD $D_A = (d, (a_r, q_f))$ over

$\Lambda \times (\Omega \cup \{q_f\})$ from A' using Lemma 5.1, then we construct in polynomial time a modified BDTD $D'_A := (d', (a_r, q_f, 0))$ over $(\Lambda \times (\Omega \cup \{q_f\}) \times \{0, 1\}) \cup \Lambda \cup \{\bot\}$ from D_A as follows. For every $\alpha \in \Lambda, q \in \Omega \cup \{q_f\}$:

$$\begin{cases} d'(\alpha, q, 0) & := \{(\alpha, (\alpha, q, 1))\} \\ d'(\alpha, q, 1) & := \{((\alpha_1, q_1, 0), (\alpha_2, q_2, 0)) \mid ((\alpha_1, q_1), (\alpha_2, q_2)) \in d(\alpha, q)\} \cup \{\varepsilon \mid \varepsilon \in d(\alpha, q)\} \\ d'(\alpha) & := \{(\bot, \bot)\} \\ d'(\bot) & := \{\varepsilon\}. \end{cases}$$

Furthermore, we rewrite Q as Q' by doing substitutions of atoms. Let $S(x) := \exists t \exists u\, \mathsf{FirstChild}(x, t) \wedge \mathsf{FirstChild}(t, u) \wedge \mathsf{Label}_\bot(u)$. We construct Q' as follows.

- $R(x, y)$, for $R \in \{\mathsf{FirstChild}, \mathsf{SecondChild}, \mathsf{Child}\}$ is replaced with $\exists z\, \mathsf{SecondChild}(x, z) \wedge R(z, y) \wedge S(x) \wedge S(y)$;
- $\mathsf{Child}^?(x, y)$ is replaced with $\exists z\, \mathsf{Child}^?(x, z) \wedge \mathsf{Child}^?(z, y) \wedge S(x) \wedge S(y)$;
- $\mathsf{Leaf}(x)$ is replaced with $\exists z\, \mathsf{SecondChild}(x, z) \wedge \mathsf{Leaf}(z) \wedge S(x)$;
- $\mathsf{Label}_\alpha(x)$ for $\alpha \in \Lambda$ is replaced with $\exists z\, \mathsf{FirstChild}(x, z) \wedge \mathsf{Label}_\alpha(z)$.

The resulting Q' is a CQ if Q is a CQ, the only atoms added reference relations existing in the current signature, and the construction is polynomial-time. For a UCQ $Q = \bigvee_i Q_i$, we write $Q' = \bigvee_i Q'_i$ where Q'_i is the CQ obtained by applying the substitutions above to Q_i.

We argue that Q' is valid over D'_A if and only if Q is valid over A', i.e., if and only if Q is valid over A.

Assume Q' is valid over D'_A and let $T \in L(A')$. There is therefore an accepting run of A on T; let T' be the annotated tree over $\Lambda \times (\Omega \cup \{q_f\})$ corresponding to this run. We know that T' is accepted by D_A. From T' we construct T'' by replacing every subtree t with root labeled with (α, q) by a subtree formed of a root labeled with $(\alpha, q, 0)$, a first child labeled with q with no children, and a second child labeled with $(\alpha, q, 1)$ with the non-root part of t underneath, similarly transformed. By construction, $T'' \in L(D'_A)$ and we know therefore that $T'' \models Q'$, which means $T'' \models Q'_i$ for some Q'_i. Let ν be a valuation of the variables of Q'_i on T'' that witnesses this. Observe that ν maps variables x of Q_i present in an atom of Q_i to nodes n of T'' with label of the form $(\alpha, q, 0)$:

- If the variable appears in Q_i in an atom $\mathsf{Root}(x)$, n is the root of T'', which has a label $(a_r, q_f, 0)$.
- If the variable appears in Q_i in an atom $\mathsf{Label}_\alpha(x)$, then the first child of n in T'' has α as label, which is only possible if n has $(\alpha, q, 0)$ for some q as label.
- If the variable is of the form $R(x, y)$ or $R(x)$ or $R(y, x)$ for $R \in \{\mathsf{FirstChild}, \mathsf{SecondChild}, \mathsf{Child}, \mathsf{Child}^?, \mathsf{Leaf}\}$

then $S(x)$ holds, which means that n has a first child whose first child is labeled with $\bot$, which is only possible if n has $(\alpha, q, 0)$ for some α, q as label.

We consider the valuation ν' that maps variables x of Q_i to the nodes of T' labeled by (α, q) that were transformed when constructing T'' to the node $\nu(x)$ labeled by $(\alpha, q, 0)$. Let now ν'' be the valuation of variables of Q_i to nodes of T corresponding to ν' on T' (remember that T' is just an annotated version of T). Then ν is a witness that $T \models Q_i$, i.e., $T \models Q$.

Conversely, assume Q is valid over A' and let $T \in L(D'_A)$. Let T' be the tree obtained from T by retaining only the nodes whose label is of the form $(\alpha, q, 0)$, attaching them to the closest retained ancestor, and dropping the 0 in the label. This is a tree in the language of D_A, i.e., an annotated tree of A'. We can thus project the states out and obtain a tree $T'' \in L(A')$, for which we know $T'' \models Q$, meaning $T'' \models Q_i$ for some i. Let ν be the valuation witnessing this, and let ν' the valuation from the variables of Q_i to nodes of T of the form $(\alpha, q, 0)$ corresponding to these nodes of T'' (via the annotation given by T'). Because $T \in L(D'_A)$ and D'_A fully constrains the positions of α and $(\alpha, q, 1)$ nodes, we can extend ν' into a valuation of the variables of Q'_i that did not appear in Q_i, to obtain a witness that $T \models Q'_i$ and thus $T \models Q'$. □

We can now prove the following result, which closely tracks Theorem 6 of [8].

THEOREM 5.3. *Given a CQ Q on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ and a* BDTD *A, it is* 2EXPTIME*-hard to decide whether Q is valid over A.*

PROOF. First, thanks to Corollary 5.2, we prove the result for a BNTA instead of a BDTD.

We adapt the proof of Theorem 6 of [8, 10], which states that validity with respect to an NTA of a CQ with *child and descendant* predicates over *unranked trees* is 2EXPTIME-hard. We adapt it by moving from unranked trees to binary trees (with the changes that it implies in the definition of an NTA), writing the output of the reduction given in [8, 10] using $\text{Child}^?$ instead of the descendant predicate.

We give a self-contained presentation keeping the notation from [8, 10] as much as possible, with notable departures highlighted in **bold font** throughout the proof.

As in [8], we reduce from the termination of an alternating EXPSPACE Turing Machine M, a 2EXPTIME-hard problem [16]. The next few paragraphs are taken in part from [8], with some minor adjustments, as we need to introduce the same concepts.

An alternating Turing machine (ATM) is a tuple $M = (\Omega, \Gamma, \Delta, q_0)$ where $\Omega = \Omega_\forall \uplus \Omega_\exists \uplus \{q_\text{a}\} \uplus \{q_\text{r}\}$ is a finite set of states partitioned into universal states from $\Omega_\forall$, existential states from $\Omega_\exists$, an accepting state q_a, and a rejecting state q_r. The (finite) tape alphabet is Γ and includes a special blank character '#'. The initial state of M is $q_0 \in \Omega$. The transition relation Δ is a subset of $(\Omega \times \Gamma) \times (\Omega \times \Gamma \times \{\text{L}, \text{R}, \text{S}\})$. The letters L, R, and S denote the directions left, right, and stay, according to which the tape head is moved.

A *configuration* of an ATM $M = (\Omega, \Gamma, \Delta, q_0)$ is a triple (τ, k, q) where $\tau : \mathbb{Z} \to \Gamma$ is the *configuration tape*, $k \in \mathbb{Z}$ is the tape head position, and $q \in Q$ is the current state. We assume that $\tau^{-1}(\Gamma \setminus \{\#\})$ is finite (only finitely many symbols on the tape are non-blank). A configuration (τ, k, q) is existential if $q \in \Omega_\exists$, universal if $q \in \Omega_\forall$, accepting if $q = q_\text{a}$, rejecting if $q = q_\text{r}$. The *initial configuration* of M on a word $w \in \Gamma^*$ is the configuration $(\tau, 0, q_0)$ where $\tau(i)$ is the i-th character of w if $1 \leqslant i \leqslant |w|$, and is '#' otherwise. A successor configuration (τ', k', q') of a configuration (τ, k, q) is a configuration such that $\tau'_{|\mathbb{Z}\setminus\{k\}} = \tau_{|\mathbb{Z}\setminus\{k\}}$ and one of the following three properties holds:

- $k' = k - 1$ and $(q, \tau(k), q', \tau'(k), \text{L}) \in \Delta$;
- $k' = k$ and $(q, \tau(k), q', \tau'(k), \text{S}) \in \Delta$;
- $k' = k + 1$ and $(q, \tau(k), q', \tau'(k), \text{R}) \in \Delta$.

An *accepting computation tree* for an ATM M on a word $w \in \Gamma^*$ is a finite *unranked* tree labeled by non-rejecting configurations of M such that: (1) if node v is labeled by an existential configuration C, then v has one child, labeled by one of the successor configurations of C; (2) if v is labeled by a universal configuration C, then v has one child for each successor configuration of C; (3) the root is labeled by the initial configuration on w; and (4) all leaves are labeled by accepting configurations (and accepting configurations only appear as leaves). An ATM M accepts a word $w \in \Gamma^*$ if there exists an accepting computation tree for M on w.

The overall idea of the proof of [8], that we closely adapt, is as follows. Let M be an ATM and w a word of Γ^* of length n. First, for technical reasons, we construct, in polynomial time an ATM M_w which accepts the empty word if and only if M accepts w; this construction is straightforward. Second, from M_w, we construct a BNTA A that checks most important properties of (suitably encoded) computation trees of M_w, except their consistency w.r.t. the transition relation of M_w. The consistency is tested by a query Q that we construct. To be precise, Q is satisfied by a tree T in $L(A)$ if and only if the transition relation of M_w is not respected by T. This means that Q is valid w.r.t. A iff there does not exist a consistent, accepting computation tree for M_w. Since 2EXPTIME is closed under complementation, we conclude that validity of CQs on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ with respect to BNTAs (and thus to BDTDs) is 2EXPTIME-hard. We emphasize that the encoding of the ATM would be straightforward if our query Q was allowed to be in non-recursive datalog, and fairly simple even if Q was a UCQ. It is *the fact that we want to show hardness for containment in the case where Q is a CQ* which motivates the subtleties in the encoding of [8], and we will inherit these in the constructions below.

Without loss of generality, we assume that universal configurations of M_w always have exactly two successor configurations. If they have less, we can just add transition(s) for every universal state and tape symbol to an accepting state; if they have more, we can introduce intermediary states to encode a conjunction as a tree of binary conjunctions, with no change to tape symbol written or tape head move.

We do not give the nondeterministic tree automaton explicitly, but trees in its language will have **the shape represented by Fig. 2.**[2] The bold nodes are the nodes added to the trees of Fig. 3 and 4 of [10]. The labels of dashed edges indicate the number of nodes between a node and its ancestor. This BNTA encodes trees that represent accepting computation trees of M_w.

Each configuration in an accepting computation is encoded by a subtree rooted by a node labeled Conf (it was called CT in [8, 10]); the Conf-node for the initial configuration appears as the unique child of a chain of ℓ nodes with dummy labels from the root for some integer ℓ that we will define further (this chain of ℓ nodes is only needed for technical reasons). A Conf-node has **two** children labeled r and **NextConf**. The subtree rooted at the r-node represents the configuration tape **and the subtree rooted by the NextConf-node has zero, one, or two Conf-children that we will regard as zero, one, or two successor configurations depending upon whether the current state is accepting, existential, or universal.** That is, the links between r, Conf, and **NextConf** nodes represent the "macro structure" of a run, the links between configurations.

A configuration tape can be viewed a complete binary tree of depth n, with leaves of the tree containing information about 2^n cells. A path in the tree encodes the binary address of a cell, with the pattern of left and right children to the leaf encoding the address, and the label of the leaf encoding the tape content. This complete binary tree, that we will refer to as the *abstract tape tree*, is itself encoded for querying purposes underneath a configuration node r. We call this encoding the *physical tape tree*. For $1 \leqslant i \leqslant n$, a node in the physical tape tree with label s represents a node at

[2] **Since our definition of BNTA requires a binary tree to be full, we need to add dummy nodes where needed, with labels distinct from real nodes. This technicality has no impact, and we will ignore these nodes.**

depth i in the abstract tape tree. Each such node representing a node at depth i, with $1 \leqslant i \leqslant n-1$, has for first child a node of label p that serves as a *navigation widget* indicating the sibling type of this node in the tape tree (left or right). It has **as second child a node of label TTCh (for *tape tree children*), which in turn has for children two nodes with label s**, encoding the two children of the current encoded node in the abstract tape tree. The navigation widget is a p-labeled node with a single x-child that has itself a single y-child. If the current encoded node in the tape tree was a left child, $x = 0$ and $y = 1$; otherwise, $x = 1$ and $y = 0$. The root of the physical tree, r, corresponds to the root of the abstract tape tree; since the root of the abstract tape tree is neither a left or right child, r does not need a navigation widget as a child. It therefore has two children, nodes labeled with s encoding the two children of the root in the abstract tape tree. For a node at level n of the abstract tape tree, the corresponding s-labeled node must have a navigation widget child encoding whether it is a left or right child of its parent. But instead of a link to nodes representing its children in the abstract tape tree, it must have a link to a node representing its content. Thus the corresponding node in the physical tape tree has one child that is a p-labeled navigation widget, and second child a c-node that encodes the content of the cell, which we describe immediately below.

In encoding the information about the content of a tape cell in a configuration, we again follow [10]. In the abstract tape tree, this information consists of the symbols on basic cells, symbol and transition followed on the current cell, and current symbol, previous state, previous symbol on previous tape cells. This means each cell would be associated with an element of $\Gamma \cup (\Gamma \times \Delta) \cup (\Gamma \times \Omega \times \Gamma)$: there are polynomially many such annotations, we refer to them in the following as $1, \ldots, k$ fixing an arbitrary order. As in [10], we want to impose a number of horizontal constraints (constraints on the annotations of neighboring cells in a given configuration) and of vertical constraints (constraints on the annotation of the same cell in successive configurations). These constraints can be written as two sets of pairs $H(M_w)$ and $V(M_w)$ of integers $1 \leqslant i, j \leqslant k$, respectively, indicating respectively whether j can appear to the right of i in a configuration, and whether j can appear in the same cell as i in a successive configuration. We refer to [10] for the full set of constraints required.

In our physical tree representation, the content of a cell is not represented by a single label, but by a chain of descendants. For each cell, the c-node has two children, labeled with m (for *me*) and f (for *forbidden*), each having as descendants a chain of k nodes that can have labels either 0 or 1. Only one node has label 1 under m, the one whose depth gives the current content of the cell; other nodes under m have label 0. Under f, for a cell at position i in the tape, node at depth j has label 0 if and only if $(i, j) \in V(M_w)$, and label 1 otherwise.

In Fig. 2 we see a bird's eye view of one of our encodings. In the box in the lower part of the picture near the center of the page, we have highlighted the physical encoding of an abstract tape, focusing on the physical encoding of the address structure while omitting the encoding of the cell content. In the box closer to the left border we have zoomed in on the encoding of a tape cell's content.

As in [10], we can construct in polynomial time a BNTA that enforces that all physical trees have the described form, including respect of horizontal constraints, initial configuration at the root and accepting configuration at the leaves, but *excluding vertical constraints*. Indeed, vertical constraints cannot (at least straightforwardly) be imposed on the tree as they relate nodes of the tree that are very far apart – see [10] for how to encode horizontal constraints and the general structure. Modifications needed because of our binary setting are minor. The language of this BNTA is exactly the codes of accepting computation trees for M_w, except that vertical constraints may be violated.

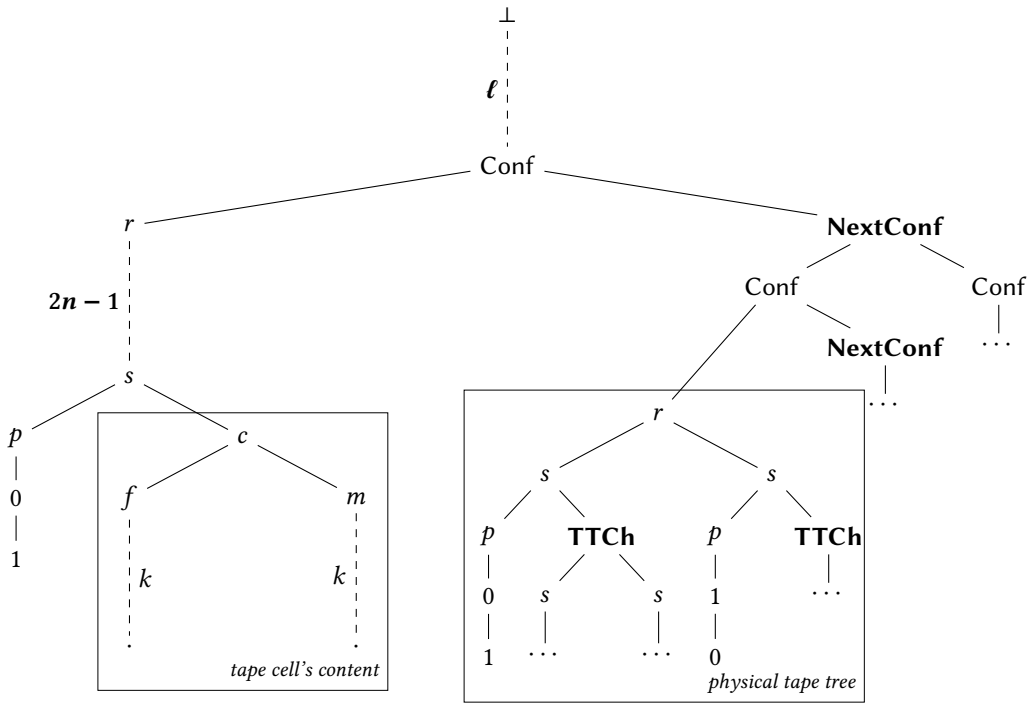


Fig. 2. General structure of trees in proof of Theorem 5.3; bold labels and counters highlight changes from the proof of Theorem 6 of [8]

We will now construct a conjunctive query that holds if vertical constraints are violated. In what follows, we denote by $R^i(x, y)$ the chain $\exists x_1 \dots x_{i-1}\, R(x, x_1) \wedge \dots \wedge R(x_{i-1}, y)$ for R a binary relation and $i \geqslant 1$.

We first need to construct a conjunctive query $\mathsf{SameCell}(s_1, s_2)$ that expresses that two s-nodes encoding a node at depth n in the tape tree (i.e., at the bottom of the tape tree) correspond to the same cell of successive configuration tapes. To do that, we will need the following subformulas:

- A formula $\mathsf{Succ}(r_1, r_2)$ that expresses that r_1 and r_2 are each the root of a tree encoding a tape, with the configuration of r_2 being a successor in the computation tree of that of r_1. Formally:

$$\mathsf{Succ}(r_1, r_2) \coloneqq \exists s_1 s_2\, \mathsf{Label}_r(r_1) \wedge \mathsf{Label}_r(r_2) \\ \wedge \mathsf{Child}(s_1, r_1) \wedge \mathsf{Child}(s_2, r_2) \wedge \mathsf{Child}^2(s_1, s_2).$$

- A formula $\Phi_i(x, y)$ that expresses that x and y are s-nodes encoding a node at the i-th level of two tape trees, such that the configuration of y is a successor in the computation tree of the configuration of x:

$$\Phi_i(x, y) \coloneqq \exists r_1 r_2\, \mathsf{Label}_s(x) \wedge \mathsf{Label}_s(y) \wedge \mathsf{Succ}(r_1, r_2) \\ \wedge \mathsf{Child}^{\mathbf{2i-1}}(r_1, x) \wedge \mathsf{Child}^{\mathbf{2i-1}}(r_2, y).$$

- A formula $\Psi_i(x, y)$ that expresses that $\Phi_i(x, y)$ holds and that, additionally, x and y are both first children or both second children of their parents ($\Psi_i(x, y)$ does not hold if x is a first child and y a second child or vice versa); note that we could not use FirstChild and SecondChild

here as it would require disjunction. We can, however, use the navigation widgets:

$$\begin{aligned}\Psi_i(x,y) \coloneqq \exists p_x p_y t_x t_y \boldsymbol{t'_x t'_y} z\, \Phi_i(x,y) \wedge \mathsf{Label}_p(p_x) \wedge \mathsf{Label}_p(p_y) \wedge \mathsf{Label}_1(t_x) \wedge \mathsf{Label}_1(t_y)\\ \wedge \mathsf{Child}(x,p_x) \wedge \mathsf{Child}(y,p_y) \wedge \mathbf{Child}(\boldsymbol{p_x, t'_x}) \wedge \mathbf{Child}(\boldsymbol{p_y, t'_y})\\ \wedge \mathbf{Child}^?(\boldsymbol{t'_x, t_x}) \wedge \mathbf{Child}^?(\boldsymbol{t'_y, t_y})\\ \wedge \mathsf{Child}^{(2i-1)+4}(z,t_x) \wedge \mathsf{Child}^{(2i-1)+6}(z,t_y)\end{aligned}$$

Observe that when x and y are both first children, the t_x and t_y are grandchildren of the p-node, and therefore **at distance $(2i-1)+3$ of the r-node**, so going up $(2i-1)+4$ times brings us to the Conf-node of the current configuration, and going up $(2i-1)+6$ times brings us to the Conf-node of the preceding configuration. Similarly, if x and y are both second children, the t_x and t_y are children of the p-node, so going up $(2i-1)+4$ times brings us to the parent of the Conf-node of the current configuration, and going up $(2i-1)+6$ times brings us to the parent of the Conf-node of the preceding configuration. **This is one of the two places where we need the chain of ℓ nodes at the root: otherwise, since the initial configuration does not have a preceding configuration, we would not be able to go high enough up in the tree to find the z node. Taking $\ell \geqslant 1$ suffices.**

- Finally, $\mathsf{SameCell}(s_1, s_2)$ is written as:

$$\begin{aligned}\mathsf{SameCell}(s_1,s_2) \coloneqq \exists x_1 \cdots x_{n-1} y_1 \cdots y_{n-1} \bigwedge_{1 \leqslant i < n-1} \left(\mathsf{Child}^2(x_i, x_{i+1}) \wedge \mathsf{Child}^2(y_i, y_{i+1})\right)\\ \wedge \mathsf{Child}^2(x_{n-1}, s_1) \wedge \mathsf{Child}^2(y_{n-1}, s_2)\\ \wedge \Psi_n(s_1,s_2) \wedge \bigwedge_{1 \leqslant i < n} \Psi_i(x_i, y_i).\end{aligned}$$

We can now use these subformulas in the following sentence, that expresses the final conjunctive query Q. It checks whether the two same cells s_1 and s_2 of successive configurations violate vertical constraints. Remember that the value of a cell is encoded under the m-node, while vertical constraints are encoded under the f-node. A vertical constraint occurs when the (unique) position of a 1-node under the m-descendant of s_2 is equal to the position of a 1-node under the f-descendant of s_1.

$$\begin{aligned}Q \coloneqq \exists s_1 s_2 t_1 t_2 f_1 m_2 u_1 u_2 z\, \mathsf{SameCell}(s_1,s_2) \wedge \mathsf{Child}(s_1,t_1) \wedge \mathsf{Child}(s_2,t_2)\\ \wedge \mathsf{Child}(t_1,f_1) \wedge \mathsf{Child}(t_2,m_2)\\ \wedge \mathsf{Label}_f(f_1) \wedge \mathsf{Label}_m(m_2) \wedge \mathsf{Label}_1(u_1) \wedge \mathsf{Label}_1(u_2)\\ \wedge \mathbf{(Child^?)^k}(\boldsymbol{f_1, u_1}) \wedge \mathbf{(Child^?)^k}(\boldsymbol{m_2, u_2})\\ \wedge \mathsf{Child}^{(2n-1+3)+k}(z,u_1) \wedge \mathsf{Child}^{(2n-1+5)+k}(z,u_2).\end{aligned}$$

This is the other place we need the chain of ℓ nodes at the root: otherwise, again, since the initial configuration does not have a preceding configuration, we would not be able to go high enough up in the tree to find the z node. Taking $\ell \geqslant k-1$ suffices.

The query Q can be constructed in polynomial time, and Q is valid over the BNTA previously constructed if and only if the Turing machine M_w has no accepting (EXPSPACE) computation tree. □

This hardness result of CQ validity over trees satisfying a DTD actually implies hardness of UCQ validity over all trees, thanks to the following argument, that shows that BDTD constraints can be encoded by the negation of a union of conjunctive queries.

Lemma 5.4. *Let $\tau = (d, l_0)$ be a* BDTD. *Then one can construct in polynomial time a UCQ q_τ over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ such that for any binary tree t, t is accepted by τ iff t does not satisfy q_τ.*

Proof. We denote by d^c the function from Λ to $2^{\Lambda\times\Lambda}$ such that for any $\alpha \in \Lambda$, $d^c(\alpha) = 2^{\Lambda\times\Lambda} - d(\alpha)$. We denote by φ_α the query equal to $\exists x\, \mathsf{Leaf}(x) \wedge \mathsf{Label}_\alpha(x)$ if $\varepsilon \in d^c(\alpha)$ and equal to false, e.g., $\exists x\, \mathsf{FirstChild}(x, x)$, otherwise.

We define q_τ as follows:

$$\bigvee_{\alpha\in\Lambda} \bigvee_{(\beta,\gamma)\in d^c(\alpha)} (\exists xyz\, \mathsf{Label}_\alpha(x) \wedge \mathsf{FirstChild}(x, y) \wedge \mathsf{Label}_\beta(y) \wedge \mathsf{SecondChild}(x, z) \wedge \mathsf{Label}_\gamma(z))$$
$$\vee \bigvee_{\alpha\in\Lambda} \varphi_\alpha$$
$$\vee \bigvee_{\alpha\in\Lambda-\{l_0\}} (\exists x\, \mathsf{Root}(x) \wedge \mathsf{Label}_\alpha(x)).$$

It is easy to check that t is accepted by the DTD τ iff t does not satisfy q_τ. □

Corollary 5.5. *Deciding whether a UCQ on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ is valid is* 2EXPTIME*-hard.*

Proof. We reduce the problem of Theorem 5.3 to the current problem. Let τ be a DTD and q be a CQ on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$.

Thanks to Lemma 5.4, we can construct q_τ in polynomial time such that t does not satisfy $q \vee q_\tau$ if and only if $t \in L(\tau)$ and $t \not\models q$. □

In the restricted case where Child and Child$^?$ relations cannot be used in the query, we prove an EXPTIME lower bound, which matches the upper bound of Corollary 4.21. This result is proved in [9], in a slightly different setting, as the Child relation is allowed (because of this restriction, we prove the hardness for UCQs instead of CQs).

Theorem 5.6. *[Adapted from Theorem 10 of [9]] Deciding whether a UCQ on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ is valid over a DTD, or whether a CQ on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ or $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ is valid over a DTD, is* EXPTIME*-hard.*

Proof. Theorem 10 of [9] shows the EXPTIME-hardness of CQ validity on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ over an NTA. The CQ used in the proof is of the form:

$$Q \coloneqq \exists x_1 \ldots \exists x_n\, \mathsf{Child}(x_1, x_2) \wedge \cdots \wedge \mathsf{Child}(x_{n-1}, x_n) \wedge \mathsf{Label}_a(x_1) \wedge \mathsf{Label}_b(x_n)$$

The setting of [9] is that of unranked trees, but the proof of Theorem 10 only uses binary trees. They are not full binary trees, but they can easily be rendered full by adding nodes with dummy labels as second children of nodes with a single child.

To move from NTAs to DTDs, we use Corollary 5.2 (in the same way, Theorem 12 of [9] states the EXPTIME-hardness of CQ validity on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ over a DTD).

The only thing that remains to be proven is the EXPTIME-hardness of UCQ validity on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ over an NTA. This is easily done by observing that in the proof of Theorem 10 of [9] all but one of the Child can be replaced by a FirstChild atom (the proof relies on the reduction from a tiling game, and non-branching chains of nodes are used to encodes tiles and constraints, with branching used only to encode choices of one of the player; the query matches nodes within the encoding two successive tiles, thus with at most one branching on the second child).

We consider the query Q'_i, for $1 \leqslant i \leqslant n-1$, obtained by replacing in Q the i-th atom $\mathsf{Child}(x_i, x_{i+1})$ by an atom $\mathsf{SecondChild}(x_i, x_{i+1})$, and all other atoms $\mathsf{Child}(x_i, x_{i+1})$ by an atom $\mathsf{FirstChild}(x_i, x_{i+1})$. We then let $Q' \coloneqq \bigvee_{i=1}^{n-1} Q'_i$, which is a formula with $n(2n-1)$ atoms (and thus of polynomial size in the size of Q).

Then Q' can be used instead of Q in the proof of Theorem 10 of [9], and Q' is on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$. □

Note that this leaves the complexity of CQ validity on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ over a DTD as an open problem.

Applying Lemma 5.4, we conclude from Theorem 5.6 the EXPTIME-hardness of UCQ containment on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ and $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$:

Corollary 5.7. *Deciding whether a UCQ is valid with respect to all trees on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ or $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child}}$ is* EXPTIME*-hard.*

Note, however, that the case of UCQ validity over all trees on $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ is in PSPACE by Proposition 4.22, and thus is not covered by Lemma 5.4. We can show that the problem over $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ is complete for PSPACE.

Proposition 5.8. *Deciding whether a UCQ is valid with respect to all trees on $\mathcal{S}^{\text{unranked}}_{\text{Child}}$ is* PSPACE*-hard. This holds even if the trees are restricted to have no branching (at most one child per node).*

Proof. We consider the problem of tiling a polynomial width grid. A solution to such a problem can be coded as a path, where the label alphabet are the tiles and the path represents the concatenation of the rows of the tiling. Given a tiling problem we can create in polynomial time a UCQ Q such that the tiling problem has a solution if and only if a counterexample to validity of Q codes a solution. The satisfaction of the horizontal constraint will correspond to one CQ of Q, while the satisfaction of the vertical constraint will represent a second CQ. □

5.2 Lower bounds for MDL and limited access containment

We now apply the prior results to get bounds on MDL and limited access containment.

We first show a 2EXPTIME lower bound for the problem of checking the containment of a monadic datalog program in a CQ. This matches the general upper bound for the containment of a datalog query within a union of CQs.

Theorem 5.9. *MDL containment in a CQ is* 2EXPTIME*-hard.*

Proof. We reduce from the problem of validity of a CQ on $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$ over a BNTA, which is 2EXPTIME-hard by Theorem 5.3.

Let $A = (\Omega, \Delta_0, \Delta, F)$ be a BNTA and Q a conjunctive query. We build a monadic datalog program P as follows:

- For every $q \in \Omega$, we have an intensional monadic predicate P_q.
- For every $q \in F$, we have a rule:

$$\text{Goal}() \leftarrow \text{Root}(r) \wedge P_q(r).$$

- For every symbol $\alpha \in \Lambda$, for every $q \in \Delta_0(\alpha)$, we have a rule:

$$P_q(l) \leftarrow \text{Leaf}(l) \wedge \text{Label}_\alpha(l) \wedge \text{Child}^?(l, l).$$

- For every symbol $\alpha \in \Lambda$, for every $q_1, q_2, q' \in \Omega$ such that $q' \in \Delta(\alpha, q_1, q_2)$, we have a rule:

$$\begin{aligned} P_{q'}(n) \leftarrow\ & \text{Label}_\alpha(n) \wedge P_{q_1}(n_1) \wedge P_{q_2}(n_2) \wedge \\ & \text{FirstChild}(n, n_1) \wedge \text{Child}(n, n_1) \wedge \text{Child}^?(n, n_1) \wedge \\ & \text{SecondChild}(n, n_2) \wedge \text{Child}(n, n_2) \wedge \text{Child}^?(n, n_2) \wedge \\ & \text{Child}^?(n, n) \end{aligned}$$

We claim that this is a valid reduction of tree validity to MDL containment. In one direction, given a counterexample to tree validity of Q, we can interpret the relations of the program in the usual way, and this is easily seen to be a counterexample to containment of P in Q.

In the other direction, if there is a counterexample to containment of P in Q, then by the proof of Proposition 2.6 there is a counterexample that is an expansion tree of the program P, with the notion of expansion tree defined in [19]: see the proof of Proposition 2.6 for a review of the notion of expansion tree. But one can see that an expansion tree of P must actually be a tree over $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$. Further, the second rule guarantees that such a tree must satisfy that BNTA A. □

We now show a more elaborate result, which is that even for a restricted subset of MDL generalizing AGEMDL, the problem of containment in a UCQ is 2EXPTIME-hard.

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). Thus, 3-GEMDL is the class of MDL queries such that:

(i) every extensional predicate appears in at most 3 rules;
(ii) every extensional predicate appears at most once in a rule.

We show that in the case of 3-GEMDL, the complexity jumps back up to 2EXPTIME, relying on our 2EXPTIME-hardness result for UCQ validity over trees of $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2,Child,Child}^?}$. Because of space constraints, the proof is deferred to an electronic appendix.

Theorem 5.10. *3-GEMDL containment in a UCQ is* 2EXPTIME*-hard.*

We similarly show, by using a reduction from UCQ validity over trees of $\mathcal{S}^{\text{bin}}_{\text{Ch1,Ch2}}$ to UCQ containment under limited access patterns, EXPTIME-hardness of this latter problem: Again, because of space constraints, the proof is deferred to an electronic appendix.

Theorem 5.11. *The problem of UCQ containment under limited access patterns is* EXPTIME*-hard. The hardness holds even if every access has at most one output position.*

Theorem 5.11 implies, in particular, by Proposition 4.23:

Corollary 5.12. *Containment of an AGEMDL query in a UCQ is* EXPTIME*-hard.*

We have thus shown matching complexity lower bounds for the upper bounds of [33] on monadic datalog containment and of Section 4.5 on AGEMDL containment in a UCQ, using reductions from tree validity problems.

6 RELATED WORK

Monadic datalog containment. Special cases of the containment problem of monadic datalog in UCQs have been studied in the past. As mentioned in the introduction, the Chaudhuri and Vardi article [18] proved a co-NEXPTIME upper bound of containment of unary MDL queries in a union of unary connected conjunctive queries. Their paper extends earlier work by Courcelle [23], who noted the connection with graph decompositions, and by Chaudhuri and Vardi [17] (published in journal form in [19]) that established complexity bounds for containment of general datalog queries in non-recursive datalog queries. The proof technique in [18] does not extend to non-connected unary queries, as our 2EXPTIME-completeness results for the general case show.

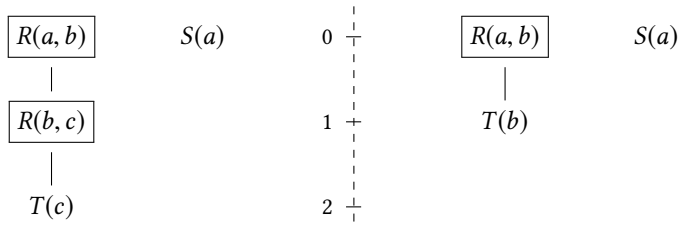


Fig. 3. Left: Example database in the crayfish-chase of query $Q_1 : \exists x \exists y\, R(x, y) \wedge S(x)$. Right: Same database after applying the subtree replacement procedure in the proof of Lemma 5 of [13].

Segoufin and ten Cate [33] 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.

[3, 4] shows a 2EXPTIME upper bound for containment of datalog in positive queries with constants.

Limited access containment. The problem of containment under access restrictions originates from [12] and is examined further in [13]. The model allows constants in the users queries, and also allows relations to be typed from a domain, which could have a fixed set of values. It restricts to conjunctive queries, rather than UCQs as in earlier work.

Both papers claim a co-NEXPTIME bound for the problem, with the proof being sketched in [12] and given in detail in [13]. The argument applies a technique similar to the one applied in our work: one shows that if there is a counterexample instance for containment, it can be taken to be tree-like (a "crayfish-chase instance" in the terminology of [13]). Then it is claimed that if there is a tree-like instance, it can be truncated to have depth polynomially bounded; this is Lemma 5 of [13]. The shrinking is done by repeatedly finding appropriate comparable nodes and replacing the subtree of the upper node with the subtree of the lower node. It is thus analogous to showing that exponential sized paths must have two nodes with the same automaton state.

Our results apply to access method containment for UCQs without constants, and provide an EXPTIME bound, and thus are orthogonal to those in [13]. In addition, our bounds apply to other classes of MDL containment problems where the polynomial depth property does not hold.

We also believe that there is a flaw in the proof of Lemma 5 of [13], already in the case of queries without constants and with a single domain for all attributes. Indeed, consider a schema with one binary relation R, with an access on its second position, and two unary relations S and T, with free accesses on each. On this schema, consider the following queries:

$$Q_1 : \exists x \exists y\, R(x, y) \wedge S(x)$$
$$Q_2 : \exists x \exists y\, R(x, y) \wedge S(x) \wedge T(y)$$

In Fig. 3 (left), we give an example database instance D that is in the crayfish-chase of Q_1 (one can verify that all properties of the crayfish-chase, in Definition 2 of [13], are satisfied). We are in subcase (1b) of the proof of Lemma 5 of [13]: Q_2 is connected, and there is more than one relation (R and S) that can be on the smallest level (0) of the mapped facts of Q_2 in a crayfish-chase of Q_1. Accordingly, we consider all paths in D from a node of level 0 to a leaf, and we consider the first and last occurrences of every relation among R and S. Only the two atoms framed in Fig. 3 (left) are of interest here. The proof then proceeds with shrinking the database, by applying the replacement (see Definition 4 of [13]) of the subtree rooted at the upper node by the subtree rooted rooted at

the lower node, resulting in the database instance in Fig. 3 (right). The proof goes on by applying further replacements to other cases of multiple occurrences of relations in the database, irrelevant here as every relation appears only once. Now, observe that Q_2 is satisfied in the resulting database but not in the original database D, whereas Lemma 5 of [13] claims that the shrunk database after subtree replacements cannot be satisfied by Q_2 if the original database was not satisfied by Q_2.

The problem comes from joins at level 0 of crayfish-chase database forests. We do not see how the proof can be easily fixed, say by imposing that subtree replacement is only carried at levels that are deep enough: it seems critical in the proof that subtree replacements of relations in Q_2 are done at the occurrence of a relation that has the lowest-level possible, to ensure that newly created joins with relations above this level do not change the satisfaction of Q_2. Of course, these comments do not amount to a disproof of the polynomial depth property claimed in the paper. We leave this question for future work.

Our main upper bound technique originates from our work on limited access querying [6]. 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 [6], relating it to tree-like instances. Our lower bounds can be seen as exploring the limits of this method. However [6] also contains some significant errors.

- A co2NEXPTIME lower bound for containment of positive queries (which extend UCQs by allowing $\exists, \vee, \wedge$ to be freely mixed) under access patterns is claimed. The proof in [6] is flawed, and in [4] a 2EXPTIME upper bound on this problem is proven. The latter is at odds (relative to complexity-theoretic hypothesis) with the former.
- A coNEXPTIME upper bound is claimed in [6] for containment of UCQs under general access patterns. The proof given there only works for schemas with a single-access per relation. The multiple access containment problem is open (see the discussion in Section 7).

Validity and containment problems on trees. Björklund et al. [8] 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 a descendant predicate.

Some of the results in this work were announced in the conference paper [5].

7 CONCLUSIONS AND FUTURE 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 (monadic expansion trees) 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 limited access containment come with two main restrictions, and we discuss lifting them here.

First we assume that there are no constants in the queries. This does not make any difference in the 2EXPTIME upper bounds for general MDL containment in UCQs, but the assumption is critical for our EXPTIME upper bounds. We believe that the addition of constants makes all of our EXPTIME-complete problems into co-NEXPTIME-complete problems.

The second restriction is that there is a single access method per relation. Although this is a standard assumption in the literature, it is easily seen to be unimportant for decidability of the

problem. Indeed, one still get a 2EXPTIME bound for limited access containment, either directly via tree-like witnesses or by reduction to MDL containment. As with constants, the issue is whether the EXPTIME bound carries over to this case. Although we conjecture that the EXPTIME bounds carry over to this case, the problem is open at the time of writing.

ACKNOWLEDGMENT

Benedikt was funded by the EPSRC grants PDQ (EP/M005852/1), ED^3 (EP/N014359/1), and DBOnto (EP/L012138/1).

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*.
[2] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. 2015. Provenance Circuits for Trees and Treelike Instances (Extended Version). CoRR abs/1511.08723.
[3] Michael Benedikt, Pierre Bourhis, and Clemens Ley. 2012. Querying Schemas with Access Paths. *PVLDB* (2012).
[4] Michael Benedikt, Pierre Bourhis, and Clemens Ley. 2015. Analysis of Schemas with Access Restrictions. *ACM Trans. Database Syst.* 40, 1 (2015), 5.
[5] Michael Benedikt, Pierre Bourhis, and Pierre Senellart. 2012. Monadic Datalog Containment. In *ICALP*.
[6] Michael Benedikt, Georg Gottlob, and Pierre Senellart. 2011. Determining relevance of accesses at runtime. In *PODS*.
[7] Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. 2015. The Complexity of Boundedness for Guarded Logics. In *LICS*. Extended version available at https://www.cs.ox.ac.uk/people/michael.vandenboom/papers/LICS15-gnfpb-long.pdf.
[8] Henrik Björklund, Wim Martens, and Thomas Schwentick. 2008. Optimizing Conjunctive Queries over Trees Using Schema Information. In *MFCS*.
[9] Henrik Björklund, Wim Martens, and Thomas Schwentick. 2013. Validity of tree pattern queries with respect to schema information. In *MFCS*.
[10] Henrik Björklund, Wim Martens, and Thomas Schwentick. 2016. Conjunctive query containment over trees using schema information. *Acta Informatica* (2016).
[11] Piero A. Bonatti. 2004. On the decidability of containment of recursive datalog queries. In *PODS*.
[12] Andrea Calì and Diego Calvanese. 2006. Containment of Conjunctive Queries under Access Limitations. In *SEBD*.
[13] Andrea Calì and Davide Martinenghi. 2008. Conjunctive Query Containment under Access Limitations. In *ER*.
[14] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. 2005. Decidable containment of recursive queries. *Theoretical Computer Science* 336, 1 (2005).
[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Containment of Conjunctive Regular Path Queries with Inverse. In *KR*.
[16] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. 1981. Alternation. *J. ACM* 28, 1 (1981).
[17] Surajit Chaudhuri and Moshe Y. Vardi. 1992. On the Equivalence of Recursive and Nonrecursive Datalog Programs. In *PODS*.
[18] Surajit Chaudhuri and Moshe Y. Vardi. 1994. On the Complexity of Equivalence between Recursive and Nonrecursive Datalog Programs. In *PODS*.
[19] Surajit Chaudhuri and Moshe Y. Vardi. 1997. On the Equivalence of Recursive and Nonrecursive Datalog Programs. *JCSS* 54, 1 (1997).
[20] Bogdan S. Chlebus. 1986. Domino-Tiling Games. *J. Comput. Syst. Sci.* 32, 3 (1986), 374–392. https://doi.org/10.1016/0022-0000(86)90036-X
[21] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. 2002. Tree Automata Techniques and Applications. Available at http://www.grappa.univ-lille3.fr/tata/.
[22] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. 1988. Decidable Optimization Problems for Database Logic Programs. In *STOC*.
[23] Bruno Courcelle. 1991. Recursive queries and context-free graph grammars. *Theoretical Computer Science* 78, 1 (1991).
[24] Oliver M. Duschka and Alon Y. Levy. 1997. Recursive Plans for Information Gathering. In *IJCAI*.
[25] Ferenc Gécseg and Magnus Steinby. 1997. "Tree Languages". In *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa (Eds.). Vol. 3. Springer Verlag, Chapter 1, 1–68.
[26] Chen Li and Edward Chang. 2001. Answering Queries with Useful Bindings. *TODS* 26, 3 (2001), 313–343.
[27] Todd D. Millstein, Alon Y. Halevy, and Marc Friedman. 2003. Query containment for data integration systems. *J. Comput. System Sci.* 66, 1 (2003), 20–39.

[28] Frank Neven. 2002. Automata Theory for XML Researchers. *SIGMOD Record* 31, 3 (2002), 39–46. https://doi.org/10.1145/601858.601869
[29] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. 1995. Answering Queries Using Templates with Binding Patterns. In *PODS*.
[30] Neil Robertson and Paul D. Seymour. 1986. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms* 7, 3 (1986).
[31] Oded Shmueli. 1993. Equivalence of datalog queries is undecidable. *J. Log. Program.* 15, 3 (1993). https://doi.org/10.1016/0743-1066(93)90040-N
[32] Masako Takahashi. 1975. Generalizations of Regular Sets and Their Application to a Study of Context-Free Languages. *Information and Control* 27, 1 (1975), 1–36.
[33] Balder ten Cate and Luc Segoufin. 2011. Unary negation. In *STACS*.

A PROOF OF THEOREM 5.10

Theorem 5.10. *3-GEMDL containment in a UCQ is* 2EXPTIME*-hard.*

Proof. We reduce from UCQ validity over trees of $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ (see Corollary 5.5).

For technical reasons, it will be more convenient to reduce from hardness of UCQ validity over trees of $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ *that have more than one node* (i.e., whose root is not a leaf). So we start by showing that we can assume this without loss of generality:

Claim A.1. *UCQ validity over trees of* $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ *reduces to UCQ validity over trees of* $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ *with more than one node.*

Proof of the claim. Let Q be a UCQ which is an instance of the former problem. For every $\alpha \in \Lambda$, we evaluate Q over the tree consisting only of a root with label α. This evaluation can be done in time linear in Q: a CQ is true over such a tree iff it does not include any Child or β atoms with $\beta \neq \alpha$.

If one of these evaluations returns false, we take for Q' the query $\exists x\, \mathsf{Root}(x) \wedge \mathsf{Leaf}(x)$; otherwise we take $Q' := Q$. Observe that if Q is valid over all trees, then it is in particular valid over all root-only trees and $Q' = Q$ is valid over all trees with more than one node. Conversely, if Q is not valid over all trees, either there is a root-only tree that does not satisfy Q and then $Q' = \mathsf{Root}(x) \wedge \mathsf{Leaf}(x)$ is not satisfied by any tree with more than one node, or there is a tree with more than one node that does not satisfy Q and then it does not satisfy $Q' = Q$ either. □

Now let Q be a UCQ over $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ which is an instance for the UCQ validity problem over trees with more than one node.

The goal is to construct a 3-GEMDL query Q'_1 and a UCQ Q'_2 over a relational signature such that Q is valid over trees of $\mathcal{S}^{\text{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ if and only if Q'_1 is contained in Q'_2. To achieve this, we build a one-to-one correspondence between binary tree witnesses of non-validity of Q and minimal monadic expansion tree witnesses of non-containment of Q'_1 in Q'_2. The binary tree will be encoded as the expansion tree of Q'_1. The temptation would be to have a unary predicate $U(x)$ and an extensional predicate $\mathsf{Child}(x, y)$ with a rule, such as $U(x) \leftarrow \mathsf{Child}(x, y), \mathsf{Child}(x, z), U(y), U(z)$ which will generate a binary expansion tree. However, this is problematic for two reasons:

(1) The 3-GEMDL restriction prevents us from having two occurrences of the same extensional predicate, such as Child in the rule above.
(2) We need to express Q by UCQ Q'_2 over the encoded structure, and Q uses the $\mathsf{Child}^?$ predicate in addition to Child. Naively inlining $\mathsf{Child}^?(x, y)$ as a disjunction $\mathsf{Child}(x, y) \vee (x = y)$ will lead to an exponential blowup.

To resolve the two sources of difficulty just mentioned, we are going to use the following relational signature:

- a 7-ary relation C;
- unary predicates P_3, P_4, P_5;
- a unary predicate LabInd_α for every label α of Λ;
- a unary predicate Leaf;
- a unary predicate Root;

The relation C serves to connect a parent node p with its two children node q and r, in a way that is pictured in Fig. 4 and that we will detail further. The positions of C have the following intended meaning:

(i) the node identifier;
(ii) a value used to connect a parent with its first child;
(iii) a value used to connect a first child with its sibling;

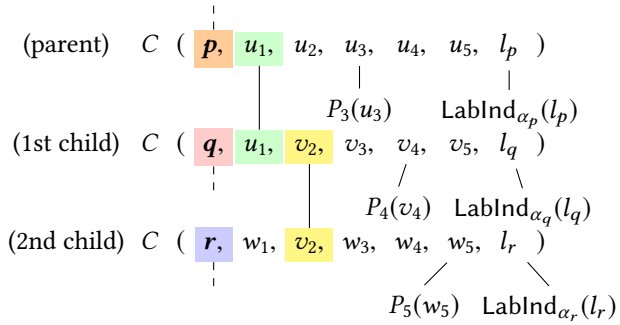


Fig. 4. Example of coding of the Child relation between a parent node p and its children q and r

(iv) a value used to distinguish the parent when connected to predicate P_3;
(v) a value used to distinguish the first child when connected to predicate P_4;
(vi) a value used to distinguish the second child when connected to predicate P_5;
(vii) a value used to connect to a label predicate LabInd_α.

We now define the 3-GEMDL query Q_1' as follows; it uses 7 unary intensional predicates $U_0 \dots U_6$ (intuitively meant to represent each abstract domain of a position of C) as well as a goal predicate:

$$
\begin{aligned}
U_0(x) &\leftarrow \mathsf{Leaf}(x)\\
U_0(x_0) &\leftarrow C(x_0,x_1,x_2,x_3,x_4,x_5,x_6) \wedge U_1(x_1) \wedge U_3(x_3) \wedge U_6(x_6)\\
U_1(x_1) &\leftarrow C(x_0,x_1,x_2,x_3,x_4,x_5,x_6) \wedge U_0(x_0) \wedge U_2(x_2) \wedge U_4(x_4) \wedge U_6(x_6)\\
U_2(x_2) &\leftarrow C(x_0,x_1,x_2,x_3,x_4,x_5,x_6) \wedge U_0(x_0) \wedge U_5(x_5) \wedge U_6(x_6)\\
U_3(x) &\leftarrow P_3(x)\\
U_4(x) &\leftarrow P_4(x)\\
U_5(x) &\leftarrow P_5(x)\\
U_6(x) &\leftarrow \mathsf{LabInd}_\alpha(x) \text{ for every } \alpha \in \Lambda\\
\mathsf{Goal} &\leftarrow \mathsf{Root}(x_0) \wedge U_0(x_0)
\end{aligned}
$$

We define the translation of the relations Child, Child$^?$, FirstChild, SecondChild, and Label_α (for $\alpha \in \Lambda$) over binary trees into conjunctive queries Child′, Child$^{?\prime}$, FirstChild′, SecondChild′, and Label'_α. These will be used as subformulas of the formula Q_2' that we will construct.

$$
\begin{aligned}
\mathsf{Child}'(x,y) &:= \exists u_1 \exists v_2 \exists u_3 \exists v_4 && C(x,u_1,\_,u_3,\_,\_,\_) \wedge P_3(u_3) \wedge\\
&&& C(\_,u_1,v_2,\_,v_4,\_,\_) \wedge P_4(v_4) \wedge\\
&&& C(y,\_,v_2,\_,\_,\_,\_)\\
\mathsf{Child}^{?\prime}(x,y) &:= \exists u_1 \exists v_2 \exists u_3 && C(x,u_1,\_,u_3,\_,\_,\_) \wedge P_3(u_3) \wedge\\
&&& C(\_,u_1,v_2,\_,\_,\_,\_) \wedge\\
&&& C(y,\_,v_2,\_,\_,\_,\_)\\
\mathsf{FirstChild}'(x,y) &:= \exists u_1 \exists u_3 \exists v_4 && C(x,u_1,\_,u_3,\_,\_,\_) \wedge P_3(u_3) \wedge\\
&&& C(y,u_1,\_,\_,v_4,\_,\_) \wedge P_4(v_4)
\end{aligned}
$$

$$\mathsf{SecondChild}'(x,y) \coloneqq \exists u_1 \exists v_2 \exists u_3 \exists v_4 \exists w_5 \quad C(x,u_1,\_,u_3,\_,\_,\_) \wedge P_3(u_3) \wedge C(\_,u_1,v_2,\_,v_4,\_,\_) \wedge P_4(v_4) \wedge C(y,\_,v_2,\_,\_,w_5,\_) \wedge P_5(w_5)$$

$$\mathsf{Label}'_\alpha(x) \coloneqq \exists l \quad C(x,\_,\_,\_,\_,\_,l) \wedge \mathsf{LabInd}_\alpha(l)$$

(The "_" symbols denote anonymous existentially used variables with a single occurrence each.)

We finally construct query Q'_2 by replacing in Q all occurrences of $R(x,y)$ with the $R'(x,y)$ subformula, where R stands for Child, Child$^?$, FirstChild, SecondChild, or Label$_\alpha$. The Root and Leaf atoms are left as is. We also add to Q'_2 the following UCQs. Their role is to ensure that the instance represents a tree structure, by forbidding a node to have two different labels, and forbidding a root node to have a parent or to be a leaf:[3]

- for every $\alpha, \beta \in \Lambda$ with $\alpha \neq \beta$:

$$\psi_{\alpha,\beta} = \exists x \exists l \exists l'\ C(x,\_,\_,\_,\_,\_,l) \wedge \mathsf{LabInd}_\alpha(l) \wedge C(x,\_,\_,\_,\_,\_,l') \wedge \mathsf{LabInd}_\beta(l');$$

- $\psi_{\mathsf{Root}} = \exists r \exists x\ \mathsf{Root}(r) \wedge \mathsf{Child}'(x,r) \quad \vee \quad \exists r\ \mathsf{Root}(r) \wedge \mathsf{Leaf}(r)$.

Q'_2 is a UCQ, and its construction is in polynomial time.

We show that containment of Q'_1 in Q'_2 under the access restrictions of the schema is equivalent to the validity of Q over binary trees of $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$.

To prove one direction, consider a binary tree T over $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$ with more than one node that does not satisfy Q. We define the relational instance I of the target schema made of:

- for each leaf node p of T, a fact $\mathsf{Leaf}(p)$;
- for each node p of T with children q and r the facts shown on Fig. 4 (with all fresh variables except for p, q, r) – in that figure α_s denotes the label of node s;
- for the root r of T, an additional fact $\mathsf{Root}(r)$.

Note that since T is not a root-only tree, every node is either a child or a parent node, and therefore there will be at least one C atom for every node of T. By construction, the instance I satisfies Q'_1 and is actually a minimal monadic expansion tree witness for Q'_1: this is due to the fact that for each node x in the tree T, one can show by induction on the tree structure that the MDL program Q'_1 produces the intensional fact $U_0(x)$. Now, observe that for X in Child, Child$^?$, FirstChild, SecondChild, $X'(x,y)$ is true in I if and only if $X(x,y)$ is true in T (in Child$'$, the second and third C atoms can unify; in Child$^{?\prime}$ all three C atoms can unify). Similarly, the translation of a Label$_\alpha$, Root or Leaf atom holds in I if and only if the original atom holds in T. Q'_2 is therefore false in I, while Q'_1 holds in I. Thus I is a witness to non-containment.

Conversely, assume we have a witness I of non-containment of Q'_1 in Q'_2. By Proposition 2.6, we can assume, without loss of generality, I to be a monadic expansion tree. We also assume I to be minimal, in the sense that no subinstance of I remains a valid monadic expansion tree instance of Q'_1 not satisfying Q'_2. We construct a binary tree T from I as follows:

- The set of nodes of T is the projection of C onto its first position in I; leaves are the content of the Leaf predicate in I, which is a subset of the nodes thanks to the minimality of I. A $\mathsf{Leaf}(x)$ fact not connected to the first position of a C fact can be safely removed from I; since ψ_{Root} is not satisfied we know that there is no $\mathsf{Leaf}(x)$ fact directly connected to a $\mathsf{Root}(x)$ fact.
- A node y is the first (resp., second) child of a node x if and only if $\mathsf{FirstChild}'(x,y)$ (resp., $\mathsf{SecondChild}'(x,y)$) holds in I.

[3]Forbidding a root node to be a leaf is a technical requirement, that ensures we have a way to assign a label to this root node.

- The root of T is the unique value in Root in I (the existence is given by the satisfiability of Q_1'. The uniqueness comes from the minimality of I: if I contains two $\mathsf{Root}(x)$ facts with distinct x's, one of them can be removed still resulting in a valid monadic expansion tree).
- The label of a given node x is given by the unique α such that

$$\exists u_1 \exists u_2 \exists u_3 \exists u_4 \exists u_5 \exists l \; C(x, u_1, u_2, u_3, u_4, u_5, l) \wedge \mathsf{LabInd}_\alpha(l)$$

  holds in I. The existence of α is guaranteed by the fact that, since no $\mathsf{Leaf}(x)$ fact is directly connected to a $\mathsf{Root}(x)$ fact, the only way to produce an intensional $U_0(x)$ fact is to have a C fact, which requires the existence of a LabInd fact. The uniqueness comes from the fact I does not satisfy any of the $\psi_{\alpha,\beta}$'s.

Since we want to view T as a tree over $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2},\mathsf{Child},\mathsf{Child}^?}$, we add to T all facts for Child and $\mathsf{Child}^?$ implied by FirstChild and SecondChild. We argue that, equivalently, $\mathsf{Child}(x, y)$ and $\mathsf{Child}^?(x, y)$ are added to T whenever $\mathsf{Child}'(x, y)$ or $\mathsf{Child}^{?\prime}(x, y)$ hold in I. Indeed, if for example $\mathsf{Child}'(x, y)$ holds, either the second and third C predicate in the definition of Child' unify (which means $\mathsf{FirstChild}'(x, y)$ holds) or they do not, in which case the $U_2(v_2)$ intensional fact required must have been created by a rule that implies that a $P_5(w_5)$ fact also exists, and this means $\mathsf{SecondChild}'(x, y)$ holds. The reasoning is similar for $\mathsf{Child}^{?\prime}(x, y)$.

Observe that by the fact that I is a monadic expansion tree and by minimality, the instance I contains, for each node x of T, one or two C facts with x as first position. If x is not a leaft, there is one fact which would be the parent of an x; this would be connected to a P_3 atom. If x is not the root, another fact would be either a first child of x, connected to a P_4 atom, or a second child, connected to a P_5 atom in the schema of Fig. 4. Keep in mind that the root cannot be a leaf. Furthermore, because of the structure of the MDL program Q_1', the tree $\mathrm{T}(I)$ which witnesses that I is a monadic expansion tree will have the following property: if $\mathsf{Child}'(x, y)$ holds, then x is an output element of a bag that is an ancestor of the bag whose output element is y.

Following the definition of trees in Section 2.3, we check that T is indeed an ordered, labeled, binary tree.

(i) We have already shown that each node has exactly one label.
(ii) We have established that leaf nodes could not have a $C(\_, \_, \_, u_3, \_, \_, \_)$ fact such that $P_3(u_3)$ holds in I, which means they cannot be a parent in T. On the other hand, all internal nodes have exactly one first child and one second child in T. Similarly, a first child cannot be a second child.
(iii) Every non-root child has a C fact in either first or second child position in I and has thus exactly one parent in T.
(iv) A node cannot be a child of itself: if $\mathsf{Child}'(x, y)$ holds, $x \neq y$ since they are output elements of different bags.
(v) We have already shown that the root r is unique. The formula ψ_{Root} guarantees that $\mathsf{Child}'(x, r)$ does not hold, which exactly means that r does not have a parent in T.

Finally, since I does not satisfy Q_2', T does not satisfy Q, which concludes the proof. □

B PROOF OF THEOREM 5.11

Theorem 5.11. *The problem of UCQ containment under limited access patterns is* EXPTIME*-hard. The hardness holds even if every access has at most one output position.*

Proof. Let Q be a union of conjunctive queries over trees of $\mathcal{S}^{\mathrm{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$. We construct in polynomial-time two UCQs over some schema with access methods, such that containment under limited access holds if and only if Q is valid over all trees. We then conclude using Corollary 5.7.

We build a schema S such that any monadic expansion tree encodes a binary tree. Let R be a relation of arity 4 such that the first position indicates the parent relation, the second and third positions indicate the FirstChild and SecondChild relations, and the last position encodes the label of the node. The access method associated with R has as input the last three positions and for output the first position. Let R_α for $\alpha \in \Lambda$ be a set of unary relations, with each of these relations having a free access method. Let R_{Root} and R_{Leaf} be two unary relations. The first relation has a Boolean access method, while the second relation has a free access method. Note that every access has at most one output position.

The relation $\mathsf{FirstChild}(x, y)$ is simulated by the formula $\varphi_{\mathsf{FirstChild}}(x, y) = \exists z \exists w\, R(x, y, z, w)$. The relation $\mathsf{SecondChild}(x, y)$ is simulated by the formula $\varphi_{\mathsf{SecondChild}}(x, y) = \exists z \exists w\, R(x, z, y, w)$. The relation $P_\alpha(x)$ is simulated by the formula $\varphi_\alpha(x) = \exists y \exists z \exists w\, R(x, y, z, w) \wedge R_\alpha(w)$. The relation $\mathsf{Root}(x)$ is simulated by the formula $\varphi_{\mathsf{Root}}(x) = R_{\mathsf{Root}}(x)$. The relation $\mathsf{Leaf}(x)$ is simulated by the formula $\varphi_{\mathsf{Leaf}}(x) = \exists y \exists z \exists w\, R(x, y, z, w) \wedge R_{\mathsf{Leaf}}(y) \wedge R_{\mathsf{Leaf}}(z)$.

We denote by q_2 the query obtained by replacing in Q the relations of $\mathcal{S}^{\mathsf{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ by the associated formulas. The query q_1 is the disjunction of the following forbidden patterns of the monadic expansion tree:

(1) A value in the second or third position of R appears in the relations R_α:

$$\bigvee_{\alpha \in \Lambda} [\exists x \exists y \exists z \exists w \quad (R(x, y, z, w) \wedge R_\alpha(y)) \quad \vee \quad (R(x, y, z, w) \wedge R_\alpha(z))] .$$

(2) A value in the fourth position of R appears in the relation R_{Leaf} or in the first position of an R-fact:

$$\exists x \exists y \exists z \exists w \quad R(x, y, z, w) \wedge (R_{\mathsf{Leaf}}(w) \vee \exists y' \exists z' \exists w'\, R(w, y', z', w')).$$

(3) A fact of R has a value in its second position appearing in R_{Leaf} and a value in its third position appearing as the first position in another R fact, and conversely with second and third reversed:

$$\exists x \exists y \exists z \exists w \exists y' \exists z' \exists w' \quad R(x, y, z, w) \wedge R_{\mathsf{Leaf}}(y) \wedge R(z, y', z', w') \quad \vee \quad R(x, y, z, w) \wedge R_{\mathsf{Leaf}}(z) \wedge R(y, y', z', w').$$

We claim that $q_0 = \exists x\, \varphi_{\mathsf{Root}}(x)$ is contained in $q_1 \wedge q_2$ under the access constraints iff Q is valid over trees in $\mathcal{S}^{\mathsf{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$.

Let us first assume T is a tree on $\mathcal{S}^{\mathsf{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ that does not satisfy query Q. We define I the relational instance on S formed of:

- for each leaf p of T with label α, facts $R(p, q, r, s)$, $R_{\mathsf{Leaf}}(q)$, $R_{\mathsf{Leaf}}(r)$, and $R_\alpha(s)$ for some fresh constants q, r, s;
- for each internal node p of T with label α and children q and r, facts $R(p, q, r, s)$ and $R_\alpha(s)$ for some fresh constant s;
- for the root r of T a fact $R_{\mathsf{Root}}(r)$.

First, observe that $\mathsf{AccFacts}(I) = I$. Indeed, all facts of the R_{Leaf} and R_α are accessible since these relations have a free access method. Furthermore, the R fact associated to each leaf is accessible using the R_{Leaf} and R_α facts as inputs of the access method on R, and the R fact associated to an internal node is accessible as long as the R facts associated with the children of that node are accessible. We can thus show by induction on the depth of the tree that every R fact is accessible. Similarly, the R_{Root} fact is accessible once the R fact corresponding to the root is accessible.

We then show that $I \models q_0$ while $I \not\models q_1 \wedge q_2$, which will witness that q_0 is not contained in $q_1 \wedge q_2$ under the access constraints. It is clear that $I \models q_0$. Now, observe that none of the forbidden

patterns of q_1 is present in I, which means $I \not\models q_1$. Now, I is simply a relational encoding of T and, by construction, $I \models q_2$ if and only if $T \models Q$, which we now not to hold. This means $I \not\models q_1 \wedge q_2$.

Conversely, assume some instance I of S is a witness of non-containment of q_0 in $q_1 \wedge q_2$ under access constraints. We can assume, w.l.o.g., I to be a monadic expansion tree (by Corollary 2.15), and to be minimal among all monadic expansion tree instances; in particular, the minimality implies that $I = \mathsf{AccFacts}(I)$. We construct a tree T on $\mathcal{S}^{\mathsf{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$ in the following manner:

- The nodes of T are the values in the first position of the relation R in I.
- A node q is first (resp., second) child of a node p if and only if $\varphi_{\mathsf{FirstChild}}(p, q)$ (resp, $\varphi_{\mathsf{SecondChild}}(p, q)$) holds.
- A node p is a leaf if $\varphi_{\mathsf{Leaf}}(p)$ holds.
- A node p is the root if $R_{\mathsf{Root}}(p)$ holds.
- The label of a node p is the α such that $\varphi_\alpha(p)$ holds.

All that remains to be shown is that T is indeed well-defined. Once this is shown, since I is a relational encoding of T and $I \models q_2$ if and only if $T \models Q$, we conclude that $T \not\models Q$ and thus that Q is not valid over all trees on $\mathcal{S}^{\mathsf{bin}}_{\mathsf{Ch1},\mathsf{Ch2}}$.

So, following the definition of trees in Section 2.2, we check that T is indeed an ordered, labeled, binary tree.

(i) Since the fourth position of R is an input position, for every R fact there should be a fact that provides the value in fourth position of that fact in an output position. The pattern (2) of q_1 prevents it to be an R_{Leaf} or R fact. The only remaining possibility is that of an R_α fact, which means for every node p, some $\varphi_\alpha(p)$ holds. Since I is a monadic expansion tree and minimal, there can only be one such R_α fact for every R fact, and no two R facts can have the same value in first position.

(ii) For every R node, by pattern (3) of R, the values in second and third position either both come from an R_{Leaf} fact, or neither comes from an R_{Leaf} fact. In the latter case, by pattern (1) of R, they cannot come from an R_α fact either, so they must both come from the output position of an R fact, which means all non-leaf nodes indeed have two child nodes.

(iii) Since I is a monadic expansion tree, no two R facts can have the same value in first position, and every non-root node indeed has a unique parent.

(iv) Again, since I is a monadic expansion tree, no node can be a child of itself.

(v) Since q_0 holds, there is an R_{Root} fact in I. By minimality of I, is it is unique. □