Optimal Nonrecursive Datalog Rewritings of Linear TGDs and Bounded (Hyper)Tree-Width Queries

M. Bienvenu[1], S. Kikot[2], R. Kontchakov[2], V. Ryzhikov[3], and M. Zakharyaschev[2]

[1] CNRS & University of Montpellier, France (`meghyn@lirmm.fr`)
[2] Birkbeck, University of London, UK (`{kikot, roman, michael}@dcs.bbk.ac.uk`)
[3] Free University of Bozen-Bolzano, Italy (`ryzhikov@inf.unibz.it`)

Abstract. Our concern is answering ontology-mediated queries $(\mathcal{O}, \boldsymbol{q})$, where $\mathcal{O}$ is a set of linear tgds and $\boldsymbol{q}$ a conjunctive query (CQ) of bounded hypertree width. Assuming that the arity of predicates is bounded, we show that polynomial-size nonrecursive Datalog rewritings can be constructed and executed in (*i*) LOGCFL for OMQs with ontologies of bounded existential depth; (*ii*) NL for OMQs with ontologies of bounded depth and CQs whose hypertree decompositions have a bounded number of leaves; (*iii*) LOGCFL for OMQs with acyclic CQs whose join trees have a bounded number of leaves.

1 Introduction

As shown in [3, 4, 13], the optimal combined complexity (LOGCFL and NL) of answering ontology-mediated queries (OMQs) with *OWL 2 QL* ontologies of bounded depth and conjunctive queries (CQs) of bounded treewidth can be achieved by means of rewriting them into nonrecursive datalog (NDL) queries, though not via positive-existential rewritings. (Note that in these cases the complexity of OMQs matches the complexity of evaluating the underlying CQs.) Our recent experiments have demonstrated that such NDL rewritings, reformulated as Spark SQL queries with views, are efficiently executed by Apache Spark taking advantage of their parallelisable structure.

The aim of this paper is to extend the above mentioned results to ontologies and CQs with predicates of *arbitrary fixed arity*. We consider ontologies that consist of linear TGDs (linear existential rules or atomic-hypothesis rules) [2, 6, 11, 12], which are instances of finite unification sets [18, 15, 16]. Our interest in this problem is also motivated by the system ETAP [5] designed to answer natural language questions by translating them into SPARQL and executing—along with background knowledge—over RDF data extracted from texts. To illustrate, suppose that the data contains the atoms $\mathit{Purchased}(j, c)$ and $\mathit{Car}(c)$ representing the sentence 'John purchased a car'. To answer the question 'has a car been sold?' ETAP utilises the ontology rules (with omitted universal quantifiers)

$$\begin{aligned}&\mathit{Purchased}(x, y) \to \exists v z\, \big(\mathit{Purchase}(v) \land \mathit{hasAgent1}(v, x) \land \\ &\hspace{14em}\mathit{hasObject}(v, y) \land \mathit{hasAgent2}(v, z)\big),\\ &\mathit{Purchase}(v) \land \mathit{hasAgent1}(v, x) \land \mathit{hasObject}(v, y) \land \mathit{hasAgent2}(v, z) \to \\ &\qquad \exists v' \,\big(\mathit{Sale}(v') \land \mathit{hasAgent1}(v', z) \land \mathit{hasObject}(v', x) \land \mathit{hasAgent2}(v', y)\big),\end{aligned}$$

where v and v' represent the acts of purchase and sale, respectively. The rules are clearly beyond the limitations of *OWL 2 QL*; however, the knowledge they represent can also be captured by means of linear TGDs with ternary predicates:

$$\mathit{Purchased}(x, y) \to \exists z\, \mathit{Purchase}(x, y, z), \quad \mathit{Purchase}(x, y, z) \to \mathit{Sale}(z, y, x),$$

which are enough to answer the query $\exists xyz\, (\mathit{Car}(y) \land \mathit{Sale}(x, y, z))$.

We classify OMQs $\boldsymbol{Q} = (\mathcal{O}, \boldsymbol{q})$ with linear TGDs and predicates of any *fixed arity* $\boldsymbol{n} < \omega$ along three axes: (1) the existential depth $\boldsymbol{d}$ of $\mathcal{O}$, that is, the maximal depth of Skolem terms in the chases of $\mathcal{O}$ over arbitrary data (cf. [8]), (2) the hypertree width $\boldsymbol{t}$ of $\boldsymbol{q}$, and (3) the number $\boldsymbol{\ell}$ of leaves in the tree underlying a hypertree decomposition of $\boldsymbol{q}$. Thus, $\mathsf{OMQ}(p_1, p_2, p_3)$ denotes the class of OMQs in which parameter (i) is bounded by $p_i \in \mathbb{N} \cup \{\infty\}$. We show that, for any fixed $\boldsymbol{d}, \boldsymbol{t}, \boldsymbol{\ell} < \omega$, answering OMQs in the classes $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$ and $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ can be done in LOGCFL (for combined complexity) by means of NDL-rewritings, and even in NL for $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \boldsymbol{\ell})$. On the other hand, one can show that answering OMQs in $\mathsf{OMQ}(\infty, \boldsymbol{t}, \boldsymbol{\ell})$, for $\boldsymbol{t}, \boldsymbol{\ell} \geq 2$, is NP-hard by observing that the sequence of tree-shaped CQs from the proof of [3, Theorem 20] is of path width 2. Thus, we obtain a full classification of the classes $\mathsf{OMQ}(p_1, p_2, p_3)$, for $p_i \in \mathbb{N} \cup \{\infty\}$, with respect to combined complexity.

2 Preliminaries

Ontology-mediated queries. Let Σ be a *relational schema* with the maximum arity $\mathit{ar}(\Sigma)$ of its predicates bounded by $\boldsymbol{n}$. By writing $P(\boldsymbol{x})$, for a predicate name P and an n-tuple $\boldsymbol{x}$ of variables (with possible repetitions), we mean that P is n-ary. By writing $\gamma(\boldsymbol{x})$, we mean that the free variables of formula γ are $\boldsymbol{x}$, where $\boldsymbol{x}$ contains no repetitions. If the meaning is clear from the context, we use set-theoretic notation for lists.

A *data instance*, $\mathcal{D}$, over Σ is any finite set of ground atoms $P(\boldsymbol{a})$ with predicate symbols P from Σ. We denote by $\mathsf{ind}(\mathcal{D})$ the set of individual constants in $\mathcal{D}$. An *ontology* is any finite set, $\mathcal{O}$, of sentences of the form

$$\forall \boldsymbol{x}\, (\gamma_0(\boldsymbol{x}) \to \exists \boldsymbol{y}\, \gamma_1(\boldsymbol{x}', \boldsymbol{y})) \qquad \text{and} \qquad \forall \boldsymbol{x}\, (\gamma_0(\boldsymbol{x}) \to \gamma_2(\boldsymbol{x}')),$$

where γ_0, γ_1 and γ_2 are atoms with predicate symbols from Σ and $\boldsymbol{x}' \subseteq \boldsymbol{x}$, for disjoint sets $\boldsymbol{x}$ and $\boldsymbol{y}$ of variables. When writing rules, we omit the universal quantifiers.

An *ontology-mediated query* (OMQ) $\boldsymbol{Q}(\boldsymbol{x})$ is a pair $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$, in which $\mathcal{O}$ is an ontology and $\boldsymbol{q}(\boldsymbol{x})$ a *conjunctive query* (*CQ*), that is, a formula of the form $\exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$, where φ is a conjunction of atoms $P(\boldsymbol{z})$ over Σ with $\boldsymbol{z} \subseteq \boldsymbol{x} \cup \boldsymbol{y}$. A tuple $\boldsymbol{a} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{x}|}$ is a *certain answer* to $\boldsymbol{Q}(\boldsymbol{x})$ over $\mathcal{D}$ if $\mathfrak{M} \models \boldsymbol{q}(\boldsymbol{a})$, for every model $\mathfrak{M}$ of $\mathcal{O} \cup \mathcal{D}$; in this case we write $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}(\boldsymbol{a})$. If the list $\boldsymbol{x}$ of *answer variables* is empty, a *certain answer* to $\boldsymbol{Q}$ over $\mathcal{D}$ is 'yes' if $\mathfrak{M} \models \boldsymbol{q}$, for every model $\mathfrak{M}$ of $\mathcal{O} \cup \mathcal{D}$, and 'no' otherwise. OMQs and CQs without answer variables are called *Boolean*. We often regard CQs as *sets* of their atoms. We abuse notation and use sets of variables in place of sequences assuming that they are ordered in some (fixed) way. Also, given $\boldsymbol{c} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{z}|}$ and $z \in \boldsymbol{z}$, we write $\boldsymbol{c}(z)$ to refer to the component of $\boldsymbol{c}$ that corresponds to z.

Canonical models. An important property of tgds is the fact [1] that, for any $\mathcal{O}$ and $\mathcal{D}$, there is a (possibly infinite) *canonical* (or *universal*) *model* $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that, for every

CQ $\boldsymbol{q}(\boldsymbol{x})$ and $\boldsymbol{a} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{x}|}$, we have $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathfrak{C}_{\mathcal{O},\mathcal{D}} \models \boldsymbol{q}(\boldsymbol{a})$. Such a canonical model can be constructed by the following (*oblivious*) *chase procedure* that, intuitively, 'repairs' $\mathcal{D}$ with respect to $\mathcal{O}$ (though not in the most economical way). With each rule ϱ of the form $\gamma_0(\boldsymbol{x}) \to \exists \boldsymbol{y}\, \gamma_1(\boldsymbol{x}', \boldsymbol{y})$, where $\boldsymbol{x} = (x_1, \dots, x_n)$, $\boldsymbol{y} = (y_1, \dots, y_k)$ and $k > 0$, we associate the k-tuple $\boldsymbol{s}_\varrho = (s^1_\varrho, \dots, s^k_\varrho)$ of distinct n-ary *Skolem function* symbols. An *application of* ϱ *to* $\mathcal{D}$ under a map $h\colon \boldsymbol{x} \to \mathsf{ind}(\mathcal{D})$ such that $h(\gamma_0) \in \mathcal{D}$ adds $h'(\gamma_1)$ to $\mathcal{D}$, where h' is defined by taking $h'(x_i) = h(x_i)$, for $1 \le i \le n$, and $h'(y_j) = s^j_\varrho(h(\boldsymbol{x}))$, for $1 \le j \le k$. An application of a rule $\gamma_0(\boldsymbol{x}) \to \gamma_2(\boldsymbol{x}')$ to $\mathcal{D}$ under such an h adds $h(\gamma_2)$ to $\mathcal{D}$. The *chase algorithm* applies these two rules exhaustively to $\mathcal{O}$ and $\mathcal{D}$ in a breadth-first manner. More precisely, we set $\mathfrak{C}^0_{\mathcal{O},\mathcal{D}} = \mathcal{D}$ and say that the atoms in $\mathfrak{C}^0_{\mathcal{O},\mathcal{D}}$ are of (*derivation*) *level* 0. Assuming that $\mathfrak{C}^{n-1}_{\mathcal{O},\mathcal{D}}$ has already been constructed, we define $\mathfrak{C}^n_{\mathcal{O},\mathcal{D}}$ as follows. Take some enumeration of all distinct pairs (ϱ_i, h_i) such that $\varrho_i \in \mathcal{O}$ with γ_i on the left-hand side is applicable to $\mathfrak{C}^{n-1}_{\mathcal{O},\mathcal{D}}$ under h_i. If none of the atoms in the $h_i(\gamma_i)$ is of level $n-1$, then we set $\mathfrak{C}^n_{\mathcal{O},\mathcal{D}} = \mathfrak{C}^{n-1}_{\mathcal{O},\mathcal{D}}$. Otherwise, we apply the ϱ_i under h_i to $\mathfrak{C}^{n-1}_{\mathcal{O},\mathcal{D}}$ one after the other and say that the newly added atoms are of (*derivation*) *level* n; the resulting extension of $\mathfrak{C}^{n-1}_{\mathcal{O},\mathcal{D}}$ is denoted by $\mathfrak{C}^n_{\mathcal{O},\mathcal{D}}$. The *canonical model* $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$ is then the union of all $\mathfrak{C}^n_{\mathcal{O},\mathcal{D}}$, for $n < \omega$.

The domain $\Delta^{\mathfrak{C}_{\mathcal{O},\mathcal{D}}}$ of $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$ consists of terms built from the constants in $\mathcal{D}$ using Skolem functions s^j_ϱ, for $\varrho \in \mathcal{O}$. The *depth* of such a term is the maximal number of nested occurrences of function symbols in it. We say that $\mathcal{O}$ is of *depth* $k \le \omega$ if k is the minimal ordinal such that $\Delta^{\mathfrak{C}_{\mathcal{O},\mathcal{D}}}$ contains no terms of depth $> k$, for any data $\mathcal{D}$. For an ontology $\mathcal{O}$ and a ground atom $P(\boldsymbol{a})$, we set $\mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})) = \Delta^{\mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}} \setminus \boldsymbol{a}$. We denote by $\mathsf{term}_{\mathcal{O}}$ the union of $\mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$, for all possible (up to renaming the constants) atoms $P(\boldsymbol{a})$ with predicates in $\mathcal{O}$ (assuming that distinct $P(\boldsymbol{a})$ do not share constants). It should be clear that $\mathcal{O}$ is of finite depth iff $\mathsf{term}_{\mathcal{O}}$ is finite. By counting the number of possible linear derivations of Skolem terms, we see that, for $\mathcal{O}$ of depth k, $|\,\mathsf{term}_{\mathcal{O}}\,| \le (ar(\Sigma)^{ar(\Sigma)}|\mathcal{O}|)^k$. We assume that any constant a occurring in $\mathsf{term}_{\mathcal{O}}$ has a *twin variable* $\widetilde{a}$ and denote by $\widetilde{\boldsymbol{a}}$ the result of replacing all constants in $\boldsymbol{a}$ with their twin variables. Given a tuple $\boldsymbol{b} \subseteq \mathsf{ind}(\mathcal{D})$, we denote by $\boldsymbol{a}/\boldsymbol{b}(\widetilde{a})$ the substitution that maps each a in $\boldsymbol{a}$ to the corresponding $\boldsymbol{b}(\widetilde{a})$ in $\boldsymbol{b}$.

NDL-rewritings. A *datalog program*, Π, is a finite set of Horn clauses of the form $\forall \boldsymbol{z}\, (\gamma_0 \leftarrow \gamma_1 \land \dots \land \gamma_m)$, where each γ_i is an atom $Q(\boldsymbol{y})$ with $\boldsymbol{y} \subseteq \boldsymbol{z}$ or an equality $(z = z')$ with $z, z' \in \boldsymbol{z}$. (As usual, we omit $\forall \boldsymbol{z}$ from clauses.) The atom γ_0 is the *head* of the clause, and $\gamma_1, \dots, \gamma_m$ its *body*. All variables in the head must occur in the body, and $=$ can only occur in the body. The predicates in the heads of clauses in Π are *IDB predicates*, the rest (including $=$) *EDB predicates*. A predicate Q *depends* on P in Π if Π has a clause with Q in the head and P in the body. Π is a *nonrecursive datalog* (NDL) *program* if the (directed) *dependence graph* of the dependence relation is acyclic. The size $|\Pi|$ of Π is the number of symbols in it. An *NDL query* is a pair $(\Pi, G(\boldsymbol{x}))$, where Π is an NDL program and G a predicate. A tuple $\boldsymbol{a} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{x}|}$ is an *answer to* $(\Pi, G(\boldsymbol{x}))$ *over* a data instance $\mathcal{D}$ if $G(\boldsymbol{a})$ holds in the first-order structure with domain $\mathsf{ind}(\mathcal{D})$ obtained by closing $\mathcal{D}$ under the clauses in Π; in this case we write $\Pi, \mathcal{D} \models G(\boldsymbol{a})$. The problem of checking whether $\boldsymbol{a}$ is an answer to $(\Pi, G(\boldsymbol{x}))$ over $\mathcal{D}$ is called the *query evaluation problem*. The *depth* of $(\Pi, G(\boldsymbol{x}))$ is the length, $\mathsf{d}(\Pi, G)$, of the longest directed path in the dependence graph for Π starting from G.

An NDL query $(\Pi, G(\boldsymbol{x}))$ is an *NDL-rewriting of an OMQ* $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ in case $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\Pi, \mathcal{D} \models G(\boldsymbol{a})$, for any $\mathcal{D}$ and any $\boldsymbol{a} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{x}|}$. Every OMQ is known to have an NDL-rewriting [2, 6].

Tree decomposition. A *tree decomposition* of a CQ $\boldsymbol{q}$ with variables $\mathit{var}(\boldsymbol{q})$ is a pair (T, λ) of an (undirected) tree $T = (V, E)$ and $\lambda \colon V \to 2^{\mathit{var}(\boldsymbol{q})}$ such that

- for any atom $P(\boldsymbol{z}) \in \boldsymbol{q}$, there exists $v \in V$ with $\boldsymbol{z} \subseteq \lambda(v)$;
- for any variable z in $\boldsymbol{q}$, the set of vertices $\{v \in V \mid z \in \lambda(v)\}$ is connected in T.

We call $\lambda(v)$ the *bag for* v. The *width* of (T, λ) is $\max_{v \in V} |\lambda(v)| - 1$. The *treewidth of* $\boldsymbol{q}$ is the minimum width over all tree decompositions of $\boldsymbol{q}$. It is known [9] that, for CQs of bounded arity $\boldsymbol{n}$, the notions of bounded treewidth and *bounded hypertree width* [10] are interchangeable. Indeed, in this case, every hypertree decomposition of width $\boldsymbol{t}$ induces a tree decomposition of width $\boldsymbol{t} \cdot \boldsymbol{n}$. A CQ $\boldsymbol{q}$ is called *acyclic* if it has a *join tree* whose nodes are the atoms of $\boldsymbol{q}$ and, whenever atoms γ_1 and γ_2 share a variable, this variable occurs in all atoms along the (unique) path in the tree linking γ_1 and γ_2. It is known [10] that a CQ $\boldsymbol{q}$ is acyclic iff $\boldsymbol{q}$ is of hypertree width 1.

3 NL and LogCFL Fragments of NDL

In this section we present two classes of NDL queries that enjoy NL- and LOGCFL-complete evaluation. First, observe that if the number of variables in each clause of an NDL query is bounded, then the size of its grounding (obtained by replacing variables by all possible combinations of constants) is polynomial. So, evaluation of such NDL queries is tractable. However, if we bound the number of variables in clauses of NDL rewritings of OMQs, then we will also effectively impose a bound on the number of answer variables in their CQs. To avoid this limitation, we treat answer variables of the CQs (and the predicate positions they occur in) differently from all other variables in the NDL-rewritings. Intuitively, answer variables get their values fixed by a candidate certain answer and thus do not cause an exponential blowup of groundings.

Formally, an NDL query $(\Pi, G(\boldsymbol{x}))$ is called *ordered* if each of its IDB predicates Q has a fixed list of variables $\boldsymbol{x}_Q \subseteq \boldsymbol{x}$, the *parameters of* Q, such that

- the parameters of G are $\boldsymbol{x}$ and, in every clause, the parameters of the head include all the parameters of the predicates in the body;
- the parameters $\boldsymbol{x}_Q$ of each Q occupy the last $|\boldsymbol{x}_Q|$ positions in every occurrence of Q in Π; they can, however, occur in other positions too.

The *width* $\mathsf{w}(\Pi, G)$ of an ordered $(\Pi, G(\boldsymbol{x}))$ is the maximum number of non-parameter variables in a clause of Π. Observe that Boolean NDL queries are trivially ordered (their IDB predicates have no parameters), and the width of such queries is simply the maximum number of variables in a clause of Π. As all the NDL-rewritings we construct are ordered, with their parameters being the answer variables, in the sequel we will consider only ordered NDL queries. We say that a class of NDL queries is of *bounded width* if there is $\mathsf{w} > 0$ such that $\mathsf{w}(\Pi, G) \leq \mathsf{w}$, for all $(\Pi, G(\boldsymbol{x}))$ in the class. As we observed above, evaluation of NDL queries of bounded width is P-complete.

Our first subclass of NDL queries is based on linear rules. An NDL program is *linear* [1] if the body of its every clause contains at most one IDB predicate.

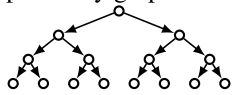
Theorem 1. *Evaluation of linear NDL queries of bounded width is* NL*-complete for combined complexity.*

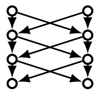
Our second subclass was inspired by semi-unbounded fan-in circuits. Recall that the class LOGCFL of problems reducible in logarithmic space to context-free languages can equivalently be defined in terms of L-uniform families of semi-unbounded fan-in circuits (where OR-gates have arbitrarily many inputs, and AND-gates two inputs) of polynomial size and logarithmic depth. Alternatively, LOGCFL can be defined using *nondeterministic auxiliary pushdown automata* (NAuxPDAs) [7], which are nondeterministic Turing machines with an additional work tape constrained to operate as a pushdown store. Sudborough [17] proved that LOGCFL coincides with the class of problems that are solved by NAuxPDAs in logarithmic space and polynomial time (the space on the pushdown tape is not subject to the logarithmic bound). Moreover, there is an algorithm that, given a semi-unbounded fan-in circuit $\boldsymbol{C}$ and an input, computes the output using an NAuxPDA in logarithmic space in the size of $\boldsymbol{C}$ and exponential time in the depth of $\boldsymbol{C}$ [19, pp. 392–397]. Using these results, it can be shown that any $(\Pi, G(\boldsymbol{x}))$ with *at most two atoms* in the body of any clause can be evaluated on a data instance $\mathcal{D}$ by an NAuxPDA in space $\log|\Pi| + \mathsf{w}(\Pi, G) \cdot \log|\mathcal{D}|$ and time $2^{O(\mathsf{d}(\Pi,G))}$ (thus, in LOGCFL provided the query width is bounded and its depth is logarithmic).

In the rewritings we propose in Sections 5 and 7, however, the number of atoms in the clauses is not bounded. We require the following to generalise the idea. A function ν from the predicate names in Π to non-negative integers $\mathbb{N}$ is called a *weight function* for an NDL query $(\Pi, G(\boldsymbol{x}))$ if, for any clause $Q(\boldsymbol{z}) \leftarrow P_1(\boldsymbol{z}_1) \land \cdots \land P_k(\boldsymbol{z}_k)$ in Π, we have

$$\nu(Q) > 0 \quad \text{and} \quad \nu(Q) \geq \nu(P_1) + \cdots + \nu(P_k),$$

Note that $\nu(P)$ can be 0 for an EDB predicate P. To illustrate, we consider NDL queries with the following dependency graphs:





The one on the left has a weight function bounded by the number of predicates (i.e., linear in the size of the query); intuitively, this function corresponds to the number of directed paths from a vertex to the leaves. In contrast, any NDL query with the dependency graph on the right can only have a weight function whose values (numbers of paths) are exponential. Linear NDL queries have weight functions bounded by 1.

Let e_Π be the maximum number of EDB predicates in a clause of Π. The *skinny depth* $\mathsf{sd}(\Pi, G)$ of $(\Pi, G(\boldsymbol{x}))$ is the minimum value of

$$2\mathsf{d}(\Pi, G) + \log \nu(G) + \log \mathsf{e}_\Pi$$

over possible weight functions ν. One can show, using Huffman coding, that any NDL query $(\Pi, G(\boldsymbol{x}))$ can be transformed into an equivalent skinny NDL query $(\Pi', G(\boldsymbol{x}))$ of depth not exceeding $\mathsf{sd}(\Pi, G)$ and such that $|\Pi'| = O(|\Pi|^2)$ and $\mathsf{w}(\Pi', G) \leq \mathsf{w}(\Pi, G)$. We say that a class of NDL queries has *logarithmic skinny depth* if there is $c > 0$ such that $\mathsf{sd}(\Pi, G) \leq c \log|\Pi|$, for all $(\Pi, G(\boldsymbol{x}))$ in the class. We now obtain:

Theorem 2. *Evaluation of NDL queries of logarithmic skinny depth and bounded width is* LOGCFL*-complete for combined complexity.*

3.1 NDL Rewritings over (Complete) Data

We say that a data instance $\mathcal{D}$ is *complete for an ontology* $\mathcal{O}$ if $\mathcal{O}, \mathcal{D} \models P(\boldsymbol{a})$ implies $P(\boldsymbol{a}) \in \mathcal{D}$, for any ground atom $P(\boldsymbol{a})$, where P in Σ and $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{D})$. An NDL query $(\Pi, G(\boldsymbol{x}))$ is an *NDL-rewriting of an OMQ* $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ *over complete data* in case $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\Pi, \mathcal{D} \models G(\boldsymbol{a})$, for any $\mathcal{D}$ complete for $\mathcal{O}$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{D})$.

Given an NDL-rewriting $(\Pi, G(\boldsymbol{x}))$ of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data, we denote by Π^* the result of replacing each EDB predicate P in Π with a fresh IDB predicate P^* of the same arity and adding the clauses $P^*(\boldsymbol{z}) \leftarrow \gamma$ for every atom γ with a predicate symbol from $\mathcal{O}$ such that $\mathcal{O} \models \gamma \to P(\boldsymbol{z})$, where $\boldsymbol{z}$ is a tuple of variables (with possible repetitions). Clearly, $(\Pi^*, G(\boldsymbol{x}))$ is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances and $|\Pi^*| \leq |\Pi| + ar(\Sigma)^{ar(\Sigma)} \cdot |\mathcal{O}|^2$.

We say that a class of OMQs is *skinny-reducible* if there are $c > 0$ and $\mathsf{w} > 0$ and an $\mathrm{L}^{\mathrm{LOGCFL}}$-transducer that, given any OMQ $\boldsymbol{Q}(\boldsymbol{x})$ in the class, computes its NDL-rewriting $(\Pi, G(\boldsymbol{x}))$ over complete data with $\mathsf{sd}(\Pi, G) \leq c \log |\Pi|$ and $\mathsf{w}(\Pi, G) \leq \mathsf{w}$. Theorem 2 and the transformation * give the following:

Corollary 1. *Answering OMQs is in* LOGCFL *for combined complexity for any skinny-reducible class.*

The transformation *, however, does not preserve linearity because it replaces occurrences of EDB predicates P by IDB predicates P^*. A more involved 'linear' construction is given in the proof of the following, where a possible increase of the width is due to the 'replacement' of atoms $P(\boldsymbol{z})$ by atoms γ whenever $\mathcal{O} \models \gamma \to P(\boldsymbol{z})$:

Lemma 1. *Fix any* $\mathsf{w} > 0$. *There is an* L^{NL}*-transducer that, for a linear NDL-rewriting* $(\Pi, G(\boldsymbol{x}))$ *of an OMQ* $\boldsymbol{Q}(\boldsymbol{x})$ *over complete data with* $\mathsf{w}(\Pi, G) \leq \mathsf{w}$*, computes its linear NDL-rewriting* $(\Pi', G(\boldsymbol{x}))$ *over arbitrary data with* $\mathsf{w}(\Pi', G) \leq \mathsf{w} + ar(\Sigma)$.

4 Conditional Rewritings

Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ with an ontology of finite depth. Intuitively, we recursively split $\boldsymbol{q}(\boldsymbol{x})$ into subqueries $\boldsymbol{q}_D$ based on subtrees D of a tree decomposition of $\boldsymbol{q}$ and combine the rewritings of $\boldsymbol{q}_D$ into a rewriting of $\boldsymbol{q}$. To guarantee 'compatibility' of the rewritings of the subqueries, we take account of the *types* of points on the boundaries of the $\boldsymbol{q}_D$. So, for each D and each type $\boldsymbol{w}$, we take a fresh IDB predicate $G_D^{\boldsymbol{w}}$ to represent the *conditional rewriting* of $\boldsymbol{q}_D$ provided that its boundary satisfies the type. We now give formal definitions.

A *type* is a partial map $\boldsymbol{s}$ from the variables of $\boldsymbol{q}$ to $\mathsf{term}_{\mathcal{O}} \cup \{\varepsilon\}$; its domain is denoted by $\mathsf{dom}(\boldsymbol{s})$. The unique partial type with $\mathsf{dom}(\boldsymbol{\varepsilon}) = \emptyset$ is denoted by $\boldsymbol{\varepsilon}$. We use types to represent how variables are mapped into the canonical model: $\boldsymbol{s}(z) = \varepsilon$ means that z is mapped to an individual constant and $\boldsymbol{s}(z) = f(\boldsymbol{a})$, for a Skolem term $f(\boldsymbol{a})$, means that z is mapped to an element of the form $f(\boldsymbol{c})$, for some $\boldsymbol{c} \subseteq \mathsf{ind}(\mathcal{D})$. Given a type $\boldsymbol{s}$ and a tuple $\boldsymbol{z} = (z_1, \dots, z_n) \subseteq \mathsf{dom}(\boldsymbol{s})$, we denote the tuple $(\boldsymbol{s}(z_1), \dots, \boldsymbol{s}(z_n))$ by $\boldsymbol{s}(\boldsymbol{z})$. A type $\boldsymbol{s}$ is *compatible* with a bag $\lambda(t)$ if $\boldsymbol{s}(x) = \varepsilon$, for all $x \in \boldsymbol{x} \cap \mathsf{dom}(\boldsymbol{s})$, and, for every $S(\boldsymbol{z}) \in \boldsymbol{q}$ with $\boldsymbol{z} \subseteq \lambda(t) \cap \mathsf{dom}(\boldsymbol{s})$, one of the following applies:

(d) $\boldsymbol{s}(\boldsymbol{z}) \subseteq \{\varepsilon\}$;

(b) there is $P(\boldsymbol{a})$ such that $\boldsymbol{s}(\boldsymbol{z}) \subseteq \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})) \cup \{\varepsilon\}$ but neither $\boldsymbol{s}(\boldsymbol{z}) \subseteq \{\varepsilon\}$ nor $\boldsymbol{s}(\boldsymbol{z}) \subseteq \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$;
(i) there is $P(\boldsymbol{a})$ such that $\boldsymbol{s}(\boldsymbol{z}) \subseteq \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$ and $S(\boldsymbol{s}(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$.

Given a type $\boldsymbol{s}$, we take a tuple of variables *var*$(\boldsymbol{s})$ that contains, for $z \in \mathsf{dom}(\boldsymbol{s}) \setminus \boldsymbol{x}$,

$$\text{variable } z, \text{ if } \boldsymbol{s}(z) = \varepsilon, \qquad \text{and} \qquad \text{variables } \widetilde{\boldsymbol{a}}, \text{ if } \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})).$$

Denote the answer variables that occur in $\mathsf{dom}(\boldsymbol{s})$ by $\boldsymbol{x}_{\boldsymbol{s}}$. Our rewritings use conjunctions $\mathsf{At}^{\boldsymbol{s}}(\mathit{var}(\boldsymbol{s}), \boldsymbol{x}_{\boldsymbol{s}})$ of the following formulas, for all $S(\boldsymbol{z}) \in \boldsymbol{q}$ with $\boldsymbol{z} \subseteq \mathsf{dom}(\boldsymbol{s})$:

(d′) $S(\boldsymbol{z})$ if $\boldsymbol{s}(\boldsymbol{z}) \subseteq \{\varepsilon\}$;
(b′) the disjunction $$\bigvee_{g\colon \boldsymbol{z}' \to \boldsymbol{a}} \Big[P(\widetilde{\boldsymbol{a}}) \ \wedge \bigwedge_{z \in \boldsymbol{z}' \text{ and } g(z) = a} (z = \widetilde{a})\Big]$$

over *grounding functions* $g\colon \boldsymbol{z}' \to \boldsymbol{a}$ such that $\boldsymbol{z}' = \{z \in \boldsymbol{z} \mid \boldsymbol{s}(z) = \varepsilon\} \neq \emptyset$, $\boldsymbol{z}'' = \{z \in \boldsymbol{z} \mid \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))\} \neq \emptyset$ and $\mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$ contains the result of replacing $\boldsymbol{z}'$ and $\boldsymbol{z}''$ in $S(\boldsymbol{z})$ by $g(\boldsymbol{z}')$ and $\boldsymbol{s}(\boldsymbol{z}'')$, respectively;
(i′) $P(\widetilde{\boldsymbol{a}})$ if $\boldsymbol{s}(\boldsymbol{z}) \subseteq \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$.

Strictly speaking, the resulting rewritings will not be NDL programs because of disjunctions in **(b′)**, but we can get rid of them using an extra predicate and (if needed) the construction from the proof of Lemma 1 keeping the size and the execution time polynomial.

5 LOGCFL Rewritings for $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$

We now construct skinny-reducible NDL rewritings for the CQs of bounded treewidth.

Theorem 3. *For any $\boldsymbol{d} \geq 0$ and $\boldsymbol{t} \geq 1$, the class $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$ is skinny-reducible.*

Fix a connected CQ $\boldsymbol{q}(\boldsymbol{x})$ and a tree decomposition (T, λ) of its Gaifman graph. Let D be a subtree of T. The *size* of D is the number of nodes in it. A node t of D is called *boundary* if T has an edge $\{t, t'\}$ with $t' \notin D$. We denote by ∂D the union of all $\lambda(t) \cap \lambda(t')$ for boundary nodes t of D and its neighbours t' in T *outside* D. The *degree* $\mathsf{deg}(D)$ of D is the number of its boundary nodes (so, the only subtree of T of degree 0 is T itself). We say that a node t *splits* D into subtrees $D_1, \dots, D_k$ if the D_i partition D without t: each node of D except t belongs to exactly one D_i.

Lemma 2. *Let D be a subtree of T of size $n > 1$.*
If $\mathsf{deg}(D) = 2$, then there is a node t splitting D into subtrees of size $\leq n/2$ and degree ≤ 2 and, possibly, one subtree of size $< n - 1$ and degree 1.
If $\mathsf{deg}(D) \leq 1$, then there is t splitting D into subtrees of size $\leq n/2$ and degree ≤ 2.

We define recursively a set $\mathfrak{R}$ of subtrees of T, a binary 'predecessor' relation $\prec$ on $\mathfrak{R}$, and a function β on $\mathfrak{R}$ indicating the bag of the splitting node. We begin by adding T to $\mathfrak{R}$. Take any $D \in \mathfrak{R}$ that has not been split yet. If D is of size 1, then $\beta(D) = \lambda(t)$ for the only node t of D. Otherwise, by Lemma 2, we find a node t in D

that splits it into $D_1, \dots, D_k$. We set $\beta(D) = \lambda(t)$ and, for $1 \leq i \leq k$, add D_i to $\mathfrak{R}$ and set $D_i \prec D$; then, we apply the procedure to each of $D_1, \dots, D_k$. For each $D \in \mathfrak{R}$, we recursively define a set of atoms

$$\boldsymbol{q}_D \;=\; \bigl\{S(\boldsymbol{z}) \in \boldsymbol{q} \mid \boldsymbol{z} \subseteq \beta(D)\bigr\} \;\cup\; \bigcup\nolimits_{D' \prec D} \boldsymbol{q}_{D'}.$$

Let $\boldsymbol{x}_D$ be the set of variables from $\boldsymbol{x}$ that occur in $\boldsymbol{q}_D$. By the definition of tree decomposition, $\boldsymbol{q}_T = \boldsymbol{q}$ and $\boldsymbol{x}_T = \boldsymbol{x}$.

We now define an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$. Fix $D \in \mathfrak{R}$ and a type $\boldsymbol{w}$ with $\mathsf{dom}(\boldsymbol{w}) = \partial D$. Let $G^{\boldsymbol{w}}_D(\mathit{var}(\boldsymbol{w}), \boldsymbol{x}_D)$ be a fresh IDB predicate with parameters $\boldsymbol{x}_D$. As we described above, a node is selected in D to split it into smaller trees (provided that it contains more than one node). We extend the type $\boldsymbol{w}$ to cover the variables $\beta(D)$ of the selected bag: more precisely, we consider types $\boldsymbol{s}$ with $\mathsf{dom}(\boldsymbol{s}) = \beta(D)$ such that they are compatible with bag $\beta(D)$ and agree with $\boldsymbol{w}$ on their common domain. Observe that, if D' is a subtree resulting from splitting D, then the domain of the extended type, $\boldsymbol{s} \cup \boldsymbol{w}$, includes $\partial D'$, and thus $\partial D'$ coincides with the domain of the restriction of $\boldsymbol{s} \cup \boldsymbol{w}$ to $\partial D'$, denoted $(\boldsymbol{s} \cup \boldsymbol{w}) \restriction_{\partial D'}$. Now, for each type $\boldsymbol{s}$ with $\mathsf{dom}(\boldsymbol{s}) = \beta(D)$ such that $\boldsymbol{s}$ is compatible with bag $\beta(D)$ and agrees with $\boldsymbol{w}$ on their common domain, the NDL program $\Pi^{\rm LOG}_{\boldsymbol{Q}}$ contains

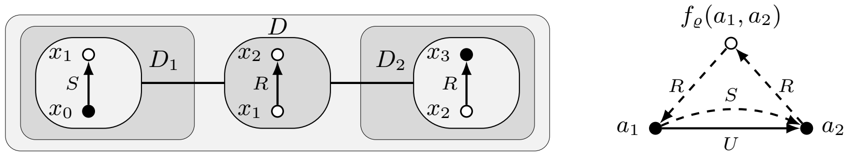
$$G^{\boldsymbol{w}}_D(\mathit{var}(\boldsymbol{w}), \boldsymbol{x}_D) \leftarrow \mathsf{At}^{\boldsymbol{s}}(\mathit{var}(\boldsymbol{s}), \boldsymbol{x}_{\boldsymbol{s}}) \wedge \bigwedge\nolimits_{D' \prec D} G^{(\boldsymbol{s}\cup\boldsymbol{w})\restriction_{\partial D'}}_{D'}(\mathit{var}((\boldsymbol{s} \cup \boldsymbol{w})\restriction_{\partial D'}), \boldsymbol{x}_{D'}).$$

By induction on $\prec$, one can now show that $(\Pi^{\rm LOG}_{\boldsymbol{Q}}, G^{\boldsymbol{\varepsilon}}_T)$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$.

Example 1. Let $\boldsymbol{q}(x_0, x_3) = \exists x_1 x_2 \,\bigl(S(x_0, x_1) \wedge R(x_1, x_2) \wedge R(x_2, x_3)\bigr)$ and $\mathcal{O}$ consist of the following linear rules:

$$\begin{aligned} \varrho\colon\quad U(x,y) &\to \exists v\, T(x,v,y), & T(x,v,y) &\to R(v,x),\\ T(x,v,y) &\to R(y,v), & T(x,v,y) &\to S(x,y). \end{aligned}$$

The subtree structure of the tree decomposition of $\boldsymbol{q}(x_0, x_3)$ and the canonical model are as follows:



The goal predicate for the rewriting of $\boldsymbol{q}(x_0, x_3)$ is $G^{\boldsymbol{\varepsilon}}_D(x_0, x_3)$ with parameters x_0 and x_3. For the type $\boldsymbol{s}$ for the middle bag sending x_1 to ε and x_2 to $f_\varrho(a_1, a_2)$, we have

$$G^{\boldsymbol{\varepsilon}}_D(x_0, x_3) \leftarrow G^{x_1 \mapsto \varepsilon}_{D_1}(x_1, x_0) \wedge U(\widetilde{a}_1, \widetilde{a}_2) \wedge (x_1 = \widetilde{a}_2) \wedge G^{x_2 \mapsto f_\varrho(a_1,a_2)}_{D_2}(\widetilde{a}_1, \widetilde{a}_2, x_3),$$

where and $\widetilde{a}_1$ and $\widetilde{a}_2$ are the twin variables in $\mathit{var}(\boldsymbol{s})$. Note that the type for $G^{x_2 \mapsto f_\varrho(a_1,a_2)}_{D_2}$ has no non-twin variables, and we have the following rule for this predicate

$$G^{x_2 \mapsto f_\varrho(a_1,a_2)}_{D_2}(\widetilde{a}_1, \widetilde{a}_2, x_3) \leftarrow U(\widetilde{a}_1, \widetilde{a}_2) \wedge (x_3 = \widetilde{a}_1).$$

Lemma 3. *For any $\mathcal{D}$ complete for $\mathcal{O}$, any predicate $G_D^{\boldsymbol{w}}$ and any $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{w})|+|\boldsymbol{x}_D|}$, we have $\Pi_{\boldsymbol{Q}}^{\mathrm{LOG}}, \mathcal{D} \models G_D^{\boldsymbol{w}}(\boldsymbol{b})$ iff there is a homomorphism $h\colon \boldsymbol{q}_D \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that*

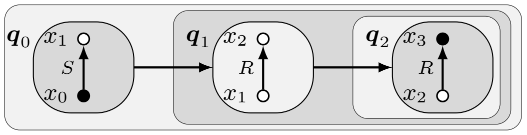
$$h(z) = \begin{cases} \boldsymbol{b}(z), & \textit{for all } z \in \boldsymbol{x}_D \textit{ and all } z \in \partial D \textit{ with } \boldsymbol{w}(z) = \varepsilon, \\ \boldsymbol{w}(z)[\boldsymbol{a}/\boldsymbol{b}(\widetilde{\boldsymbol{a}})], & \textit{for all } z \in \partial D \textit{ with } \boldsymbol{w}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases}$$

6 NL Rewritings for OMQ($\boldsymbol{d}$, $\boldsymbol{t}$, $\boldsymbol{\ell}$)

For OMQs based upon bounded leaf queries and bounded depth ontologies, we establish the following theorem:

Theorem 4. *Let $\boldsymbol{d} \geq 0$, $\boldsymbol{t} \geq 1$ and $\boldsymbol{\ell} \geq 2$ be fixed. There is an L^{NL}-transducer that, given any OMQ in $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \boldsymbol{\ell})$, constructs its polynomial-size linear NDL-rewriting of width $\leq \boldsymbol{\ell}(\boldsymbol{t}+1)$.*

Let $\mathcal{O}$ be an ontology of finite depth $\boldsymbol{d}$ and $\boldsymbol{q}(\boldsymbol{x})$ a CQ with a tree decomposition (T, λ) of width $\leq \boldsymbol{t}$ having $\leq \boldsymbol{\ell}$ leaves. Fix one of the nodes of T as root, and let M be the maximum distance to a leaf from the root. For $0 \leq n \leq M$, by an *n-slice* we mean the set of all nodes of T located at distance n from the root. Denote by $\boldsymbol{y}^n$ the union of all bags $\lambda(t)$ for a node t in the n-slice. For $1 \leq n \leq M$, let $\boldsymbol{z}^n$ be the union of all $\lambda(t) \cap \lambda(t')$ for a node t in the n-slice and its predecessor t' in T (which is in $(n-1)$-slice), and let $\boldsymbol{z}^0 = \emptyset$. By definition, $\boldsymbol{z}^{n+1} \subseteq \boldsymbol{y}^{n+1} \cap \boldsymbol{y}^n$ and, clearly, $|\boldsymbol{z}^n| \leq |\boldsymbol{y}^n| \leq \boldsymbol{\ell}(\boldsymbol{t}+1)$. Denote by $\boldsymbol{q}_n(\boldsymbol{z}^n_{\exists}, \boldsymbol{x}^{\geq n})$ the query consisting of all atoms $S(\boldsymbol{z})$ of $\boldsymbol{q}$ with $\boldsymbol{z} \subseteq \bigcup_{k \geq n} \boldsymbol{y}^k$, where $\boldsymbol{z}^n_{\exists} = \boldsymbol{z}^n \setminus \boldsymbol{x}$ and $\boldsymbol{x}^{\geq n} = \boldsymbol{x} \cap \bigcup_{k \geq n} \boldsymbol{y}^k$. These queries and sets of variables for the CQ from Example 1 are shown below:



$\boldsymbol{y}^2 = \{x_2, x_3\}$ $\boldsymbol{z}^2 = \{x_2\}$ $\boldsymbol{x}^2 = \{x_3\}$
$\boldsymbol{y}^1 = \{x_1, x_2\}$ $\boldsymbol{z}^1 = \{x_1\}$ $\boldsymbol{x}^1 = \emptyset$
$\boldsymbol{y}^0 = \{x_0, x_1\}$ $\boldsymbol{z}^0 = \emptyset$ $\boldsymbol{x}^0 = \{x_0\}$

A *type for $\boldsymbol{z}^n$* is a total map $\boldsymbol{w}$ from $\boldsymbol{z}^n$ to $\mathsf{term}_{\mathcal{O}} \cup \{\varepsilon\}$. Likewise, a *type for $\boldsymbol{y}^n$* is a total map $\boldsymbol{s}$ from $\boldsymbol{y}^n$ to $\mathsf{term}_{\mathcal{O}} \cup \{\varepsilon\}$. We say $\boldsymbol{s}$ *compatible* with $\boldsymbol{y}^n$ if it is compatible with every bag $\lambda(t)$ in the n-slice.

Consider the NDL program $\Pi_{\boldsymbol{Q}}^{\mathrm{LIN}}$ defined as follows. For every $0 \leq n < M$ and every type $\boldsymbol{w}$ for $\boldsymbol{z}^n$, we introduce a new IDB predicate $G_n^{\boldsymbol{w}}(var(\boldsymbol{w}), \boldsymbol{x}^{\geq n})$ with parameters $\boldsymbol{x}^{\geq n}$. For each type $\boldsymbol{s}$ for $\boldsymbol{y}^n$ such that $\boldsymbol{s}$ is compatible with $\boldsymbol{y}^n$ and agrees with $\boldsymbol{w}$ on $\boldsymbol{z}^n$, the program $\Pi_{\boldsymbol{Q}}^{\mathrm{LIN}}$ contains the clause

$$G_n^{\boldsymbol{w}}(var(\boldsymbol{w}), \boldsymbol{x}^{\geq n}) \leftarrow \mathsf{At}^{\boldsymbol{s}}(var(\boldsymbol{s}), \boldsymbol{x}_{\boldsymbol{s}}) \wedge G_{n+1}^{\boldsymbol{s}\restriction \boldsymbol{z}^{n+1}}(var(\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}), \boldsymbol{x}^{\geq n+1}).$$

For every type $\boldsymbol{w}$ for $\boldsymbol{z}^M$ and every type $\boldsymbol{s}$ for $\boldsymbol{y}^M$ such that $\boldsymbol{s}$ is compatible with $\boldsymbol{y}^M$ and agrees with $\boldsymbol{w}$ on $\boldsymbol{z}^M$, we include the clause

$$G_M^{\boldsymbol{w}}(var(\boldsymbol{w}), \boldsymbol{x}^{\geq M}) \leftarrow \mathsf{At}^{\boldsymbol{s}}(var(\boldsymbol{s}), \boldsymbol{x}_{\boldsymbol{s}}).$$

Finally, we use $G_0^{\boldsymbol{\varepsilon}}$ with parameters $\boldsymbol{x}$ as the goal predicate (note that $\boldsymbol{z}^0 = \emptyset$, and so the domain of any type for $\boldsymbol{z}^0$ is empty).

Lemma 4. *For any $\mathcal{D}$ complete for $\mathcal{O}$, any predicate $G_n^{\boldsymbol{w}}$, any $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{w})|+|\boldsymbol{x}^{\geq n}|}$, we have $\Pi_{\boldsymbol{Q}}^{\text{LIN}}, \mathcal{D} \models G_n^{\boldsymbol{w}}(\boldsymbol{b})$ iff there is a homomorphism $h\colon \boldsymbol{q}_n \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that*

$$h(z) = \begin{cases} \boldsymbol{b}(z), & \text{for all } z \in \boldsymbol{x}^{\geq n} \text{ and all } z \in \boldsymbol{z}_{\exists}^{n} \text{ with } \boldsymbol{w}(z) = \varepsilon, \\ \boldsymbol{w}(z)[\boldsymbol{a}/\boldsymbol{b}(\widetilde{\boldsymbol{a}})], & \text{for all } z \in \boldsymbol{z}_{\exists}^{n} \text{ with } \boldsymbol{w}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases}$$

It should be clear that $\Pi_{\boldsymbol{Q}}^{\text{LIN}}$ is a linear NDL program of width $\leq \boldsymbol{\ell}(\boldsymbol{t}+1)$ and containing $\leq |\boldsymbol{q}| \cdot |\,\mathsf{term}_{\mathcal{O}}\,|^{\boldsymbol{\ell}(\boldsymbol{t}+1)}$ predicates. Moreover, it takes only logarithmic space to store a type $\boldsymbol{w}$, which allows us to show that $\Pi_{\boldsymbol{Q}}^{\text{LIN}}$ can be computed by an L^{NL}-transducer. We apply Lemma 1 to obtain an NDL-rewriting for arbitrary data instances, and then use Theorem 1 to conclude that the resulting program can be evaluated in NL.

7 LOGCFL Rewritings for $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$

Unlike the previous two classes, answering OMQs from the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ can be harder—LOGCFL-complete—than evaluating their CQs, which can be done in NL.

Theorem 5. *For any fixed $\boldsymbol{\ell} \geq 2$, the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ is skinny-reducible.*

For OMQs with ontologies of unbounded depth and acyclic CQs whose join trees have a bounded number of leaves, our rewriting uses the notion of Skolem witness that generalises tree witnesses [14].

Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ, let $\mathfrak{s} = (\mathfrak{s}_{\mathsf{r}}^1, \dots, \mathfrak{s}_{\mathsf{r}}^n, \mathfrak{s}_{\mathsf{i}})$ be a tuple of disjoint sets of variables in $\boldsymbol{q}(\boldsymbol{x})$ such that $\mathfrak{s}_{\mathsf{i}} \neq \emptyset$ and $\mathfrak{s}_{\mathsf{i}} \cap \boldsymbol{x} = \emptyset$, and let $\mathfrak{s}_{\mathsf{r}} = \mathfrak{s}_{\mathsf{r}}^1 \cup \dots \cup \mathfrak{s}_{\mathsf{r}}^n$,

$$\boldsymbol{q}_{\mathfrak{s}} \;=\; \big\{\, S(\boldsymbol{z}) \in \boldsymbol{q} \mid \boldsymbol{z} \subseteq \mathfrak{s}_{\mathsf{r}} \cup \mathfrak{s}_{\mathsf{i}} \text{ and } \boldsymbol{z} \not\subseteq \mathfrak{s}_{\mathsf{r}} \,\big\}.$$

If $\boldsymbol{q}_{\mathfrak{s}}$ is a minimal subset of $\boldsymbol{q}$ containing every atom of $\boldsymbol{q}$ with at least one variable from $\mathfrak{s}_{\mathsf{i}}$ and such that there is a homomorphism $h\colon \boldsymbol{q}_{\mathfrak{s}} \to \mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$ with $\boldsymbol{a} = (a_1, \dots, a_n)$ and $h^{-1}(a_j) = \mathfrak{s}_{\mathsf{r}}^j$ for $1 \leq j \leq n$, then we call $\mathfrak{s}$ a *Skolem witness for $\boldsymbol{Q}(\boldsymbol{x})$ generated by $P(\boldsymbol{a})$*. Intuitively, $\mathfrak{s}$ identifies a minimal subset of $\boldsymbol{q}$ that can be mapped to the *Skolem part* of the canonical model $\mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$ consisting of Skolem terms: the variables in $\mathfrak{s}_{\mathsf{r}}$ are mapped to constants from $\boldsymbol{a}$ and the variables in $\mathfrak{s}_{\mathsf{i}}$ to Skolem terms in $\mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$.

The logarithmic-depth NDL-rewriting for $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ is based on the following:

Lemma 5. *Every tree T of size n has a node splitting it into subtrees of size $\leq \lceil n/2 \rceil$.*

Let $\boldsymbol{Q}(\boldsymbol{x}_0) = (\mathcal{O}, \boldsymbol{q}_0(\boldsymbol{x}_0))$ be an OMQ with an acyclic CQ having a join tree T_0. We repeatedly apply Lemma 5 to decompose the CQ into smaller and smaller subqueries. Formally, for an acyclic CQ $\boldsymbol{q}$, we denote by $\gamma_{\boldsymbol{q}}$ a vertex in the join tree T for $\boldsymbol{q}$ that satisfies the condition of Lemma 5. Let $\mathfrak{Q}$ be the smallest set containing $\boldsymbol{q}_0(\boldsymbol{x}_0)$ and the following CQs, for every $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ with at least one existentially quantified variable:

(1) the CQs $\boldsymbol{q}_i(\boldsymbol{x}_i)$ corresponding to the connected components T_i with root $\gamma_{\boldsymbol{q}_i}$ adjacent to $\gamma_{\boldsymbol{q}}$ of the result of removing $\gamma_{\boldsymbol{q}}$ from T, where $\boldsymbol{x}_i$ consists of the restriction of $\boldsymbol{x}$ to the variables in $\boldsymbol{q}_i$ together with the common variables of $\gamma_{\boldsymbol{q}_i}$ and $\gamma_{\boldsymbol{q}}$;

(2) for each Skolem witness $\mathfrak{s}$ for $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathfrak{s}_\mathsf{r} \neq \emptyset$ and $\gamma_{\boldsymbol{q}} \in \boldsymbol{q}_\mathfrak{s}$, the CQs $\boldsymbol{q}_1^\mathfrak{s}(\boldsymbol{x}_1^\mathfrak{s}), \dots, \boldsymbol{q}_k^\mathfrak{s}(\boldsymbol{x}_k^\mathfrak{s})$ that correspond to the connected components $T_i^\mathfrak{s}$ of the results of removing $\boldsymbol{q}_\mathfrak{s}$ from T (note that $\boldsymbol{q}_\mathfrak{s}$ is connected in T), where each $\boldsymbol{x}_i^\mathfrak{s}$ is the set of variables in $\boldsymbol{x} \cup \mathfrak{s}_\mathsf{r}$ that occur in $\boldsymbol{q}_i^\mathfrak{s}$.

The NDL program $\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}$ uses IDB predicates $G_{\boldsymbol{q}}(\boldsymbol{x})$, for $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$, whose parameters are the variables in $\boldsymbol{x}_0$ that occur in $\boldsymbol{q}(\boldsymbol{x})$. For each $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ that has no existentially quantified variables, we include the clause $G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$. For any $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ with existential variables, we include

$$G_{\boldsymbol{q}}(\boldsymbol{x}) \quad \leftarrow \gamma_{\boldsymbol{q}} \;\wedge \bigwedge\nolimits_{1 \leq i \leq n} G_{\boldsymbol{q}_i}(\boldsymbol{x}_i),$$

where $\boldsymbol{q}_1(\boldsymbol{x}_1), \dots, \boldsymbol{q}_n(\boldsymbol{x}_n)$ are the subqueries obtained by splitting $\boldsymbol{q}$ by $\gamma_{\boldsymbol{q}}$ in (1), and, for any Skolem witness $\mathfrak{s}$ of $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathfrak{s}_\mathsf{r} \neq \emptyset$ and $\gamma_{\boldsymbol{q}} \in \boldsymbol{q}_\mathfrak{s}$ and any $P(\boldsymbol{a})$ generating $\mathfrak{s}$, the clause

$$G_{\boldsymbol{q}}(\boldsymbol{x}) \quad \leftarrow P(\widetilde{\boldsymbol{a}}) \;\wedge \bigwedge\nolimits_{z \in \mathfrak{s}_\mathsf{r}^j} (z = \widetilde{a}_j) \;\; \wedge \bigwedge\nolimits_{1 \leq i \leq k} G_{\boldsymbol{q}_i^\mathfrak{s}}(\boldsymbol{x}_i^\mathfrak{s}),$$

where $\boldsymbol{q}_1^\mathfrak{s}, \dots, \boldsymbol{q}_k^\mathfrak{s}$ are the connected components of $\boldsymbol{q}$ without $\boldsymbol{q}_\mathfrak{s}$. Finally, if $\boldsymbol{q}_0$ is Boolean, then we include $G_{\boldsymbol{q}_0} \leftarrow P(\widetilde{\boldsymbol{a}})$ for all atoms $P(\boldsymbol{a})$ such that $\mathcal{O}, \{P(\boldsymbol{a})\} \models \boldsymbol{q}_0$.

Lemma 6. *For any OMQ with an acyclic CQ, any data $\mathcal{D}$ complete for $\mathcal{O}$, any query $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ and any $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{x}|}$, we have $\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}, \mathcal{D} \models G_{\boldsymbol{q}}(\boldsymbol{b})$ iff there is a homomorphism $h\colon \boldsymbol{q} \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ with $h(\boldsymbol{x}) = \boldsymbol{b}$.*

Now fix $\boldsymbol{\ell} > 1$ and consider $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}_0(\boldsymbol{x}))$ from the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ (remember that we have fixed arity $\boldsymbol{n}$). The size of the program $\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}$ is polynomially bounded in $|\boldsymbol{Q}|$ since $\boldsymbol{q}_0$ has polynomially-many subtrees of $T_{\boldsymbol{q}_0}$ and $O(|\boldsymbol{q}_0|^{\boldsymbol{\ell}})$ Skolem witnesses (there are at most $O(|\boldsymbol{q}_0|^{\boldsymbol{\ell}} \cdot |\Sigma| \cdot \boldsymbol{n}^{\boldsymbol{n}})$ pairs of a Skolem witness $\mathfrak{s}$ and its generating atom $P(\boldsymbol{a})$). It is readily seen that the function ν defined by $\nu(G_{\boldsymbol{q}}) = |\boldsymbol{q}|$, for each $\boldsymbol{q} \in \mathfrak{Q}$, is a weight function for $(\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}, G_{\boldsymbol{q}_0}(\boldsymbol{x}))$ with $\nu(G_{\boldsymbol{q}_0}) \leq |\boldsymbol{Q}|$. Moreover, by Lemma 5, $\mathsf{d}(\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}, G_{\boldsymbol{q}_0}) \leq \log \nu(G_{\boldsymbol{q}_0}) + 1$; also, $\mathsf{w}(\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}, G_{\boldsymbol{q}_0}) \leq \boldsymbol{\ell} + 1$. Finally, we note that, since the number of leaves is bounded, it is in NL to decide whether a vertex satisfies the conditions of Lemma 5, and in LOGCFL to decide whether $\mathcal{O}, \{P(\boldsymbol{a})\} \models \boldsymbol{q}(\boldsymbol{a})$, for bounded-leaf acyclic CQs $\boldsymbol{q}(\boldsymbol{x})$ (see the full version[1]), or whether a (logspace) representation of a possible Skolem witness is indeed a Skolem witness. This allows us to show that $(\Pi_{\boldsymbol{Q}}^{\mathsf{Sw}}, G_{\boldsymbol{q}_0}(\boldsymbol{x}))$ can be generated by an $\mathrm{L}^{\mathrm{LOGCFL}}$-transducer. By Corollary 1, the obtained NDL-rewritings can be evaluated in LOGCFL.

8 Conclusion

We presented NDL rewritings for three classes of OMQs with CQs of bounded (hyper)-tree width and ontologies given as linear TGDs. These NDL rewritings can be constructed and evaluated in LOGCFL, NL and LOGCFL, respectively (provided that the arity of predicates is bounded). Since the three upper bounds match the lower bounds inherited from the OWL 2 QL setting [4], the proposed rewritings are theoretically optimal.

[1] `http://www.dcs.bbk.ac.uk/~kikot/DL17-1-full.pdf`

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Baget, J.-F., Leclère, M., Mugnier, M.-L., Salvat, E.: On rules with existential variables: Walking the decidability line. Artificial Intelligence 175(9–10), 1620–1654 (2011)
3. Bienvenu, M., Kikot, S., Kontchakov, R., Podolskii, V.V., Ryzhikov, V., Zakharyaschev, M.: The complexity of ontology-based data access with OWL 2 QL and bounded treewidth queries. In: Proc. of the 26th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS (2017)
4. Bienvenu, M., Kikot, S., Podolskii, V.V.: Tree-like queries in OWL 2 QL: succinctness and complexity results. In: Proc. of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015. pp. 317–328. IEEE Computer Society (2015)
5. Boguslavsky, I., Dikonov, V., Iomdin, L., Lazursky, A., Sizov, V., Timoshenko, S.: Semantic analysis and question answering: a system under development. In: Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference Dialogue. p. 21. No. 14 (2015)
6. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics 14, 57–83 (2012)
7. Cook, S.A.: Characterizations of pushdown machines in terms of time-bounded computers. Journal of the ACM 18(1), 4–18 (1971)
8. Cuenca Grau, B., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity notions for existential rules and their application to query answering in ontologies. Journal of Artificial Intelligence Research 47, 741–808 (2013)
9. Gottlob, G., Greco, G., Leone, N., Scarcello, F.: Hypertree decompositions: Questions and answers. In: Proc. of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS (2016)
10. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)
11. Gottlob, G., Manna, M., Pieris, A.: Polynomial Rewritings for Linear Existential Rules, pp. 2992–2998. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence, IJCAI (2015)
12. Gottlob, G., Orsi, G., Pieris, A.: Query rewriting and optimization for ontological databases. ACM Transactions on Database Systems 39(3), 25:1–25:46 (2014)
13. Kikot, S., Kontchakov, R., Podolskii, V., Zakharyaschev, M.: On the succinctness of query rewriting over shallow ontologies. In: Proc. of the Joint Meeting of the 23rd EACSL Annual Conf. on Computer Science Logic (CSL 2014) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2014). pp. 57:1–57:10. ACM (2014)
14. Kikot, S., Kontchakov, R., Zakharyaschev, M.: Conjunctive query answering with OWL 2 QL. In: Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012). pp. 275–285. AAAI (2012)
15. König, M., Leclère, M., Mugnier, M.-L., Thomazo, M.: Sound, complete and minimal UCQ-rewriting for existential rules. Semantic Web 6(5), 451–475 (2015)
16. König, M., Leclere, M., Mugnier, M.-L.: Query rewriting for existential rules with compiled preorder. In: Proc. of the 24th Int. Joint Conf. on Artificial Intelligence, IJCAI (2015)
17. Sudborough, I.H.: On the tape complexity of deterministic context-free languages. Journal of the ACM 25(3), 405–414 (1978)
18. Thomazo, M.: Compact rewriting for existential rules. In: Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence, IJCAI (2013)
19. Venkateswaran, H.: Properties that characterize LOGCFL. Journal of Computer and System Sciences 43(2), 380–404 (1991)

A Proof of Lemma 1

Lemma 1. *Fix any* $\mathsf{w} > 0$. *There is an* L^{NL}*-transducer that, for a linear NDL-rewriting* $(\Pi, G(\boldsymbol{x}))$ *of an OMQ* $\boldsymbol{Q}(\boldsymbol{x})$ *over complete data with* $\mathsf{w}(\Pi, G) \leq \mathsf{w}$*, computes its linear NDL-rewriting* $(\Pi', G(\boldsymbol{x}))$ *over arbitrary data with* $\mathsf{w}(\Pi', G) \leq \mathsf{w} + ar(\Sigma)$*.*

Proof. Let $(\Pi, G(\boldsymbol{x}))$ be a linear NDL query such that $\mathsf{w}(\Pi, G) \leq \mathsf{w}$ and such that it is a rewriting of OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$ over complete data instances. We replace every clause λ in Π by a set of clauses Λ' defined as follows. Let λ be of the form

$$Q(\boldsymbol{z}) \leftarrow P(\boldsymbol{z}_0) \land S_1(\boldsymbol{x}_1) \land \dots \land S_n(\boldsymbol{x}_n) \land \psi, \tag{1}$$

where $P(\boldsymbol{x}_0)$ is the only IDB body atom in λ, $S_1(\boldsymbol{x}_1), \dots, S_n(\boldsymbol{x}_n)$ are the EDB body atoms not involving equality and ψ contains all equality body atoms. For every $S_i(\boldsymbol{x}_i)$, we fix a set $\boldsymbol{y}_i$ of fresh variables, $|\boldsymbol{y}_i| \leq ar(\Sigma)$, and define a set Γ_i of atoms that imply $S_i(\boldsymbol{x}_i)$ with respect to $\mathcal{O}$:

$$\Gamma_i \;=\; \big\{\, \gamma(\boldsymbol{x}_i, \boldsymbol{y}_\gamma) \mid \mathcal{O} \models \gamma(\boldsymbol{x}_i, \boldsymbol{y}_\gamma) \to S_i(\boldsymbol{x}_i) \text{ with } \boldsymbol{y}_\gamma \subseteq \boldsymbol{y}_i \,\big\}.$$

Note that the Γ_i share the same additional variables $\boldsymbol{y}_i$ (but the $\boldsymbol{y}_i$ are pairwise disjoint). Then Λ' comprises the following clauses:

$$\begin{aligned} Q_0(\boldsymbol{z}_0) &\leftarrow P(\boldsymbol{z}_0),\\ Q_i(\boldsymbol{z}_i) &\leftarrow Q_{i-1}(\boldsymbol{z}_{i-1}) \land \gamma(\boldsymbol{x}_i, \boldsymbol{y}_\gamma), \quad \text{for } 1 \leq i \leq n \text{ and } \gamma(\boldsymbol{x}_i, \boldsymbol{y}_\gamma) \in \Gamma_i,\\ Q(\boldsymbol{z}) &\leftarrow Q_n(\boldsymbol{z}_n) \land \psi, \end{aligned}$$

where $\boldsymbol{z}_i$, for $i > 0$, is the union of $\boldsymbol{z}_{i-1}$ and $\boldsymbol{x}_i$ (note that $\boldsymbol{z}_n = \boldsymbol{z}$). Let Π' be the program obtained from Π by replacing each clause λ by the set of clauses Λ'. By construction, Π' is a linear NDL program and its width does not exceed $\mathsf{w}(\Pi, G) + ar(\Sigma)$, where the possible increase is due to the $\boldsymbol{y}_\gamma$.

We now argue that $(\Pi', G(\boldsymbol{x}))$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances. It can be easily verified that $(\Pi', G(\boldsymbol{x}))$ is equivalent to $(\Pi'', G(\boldsymbol{x}))$, where NDL program Π'' is obtained from Π by replacing each clause (1) by the (possibly exponentially larger) set of clauses of the form

$$Q(\boldsymbol{z}) \leftarrow P_0(\boldsymbol{z}_0) \land \gamma_1(\boldsymbol{x}_1, \boldsymbol{y}_{\gamma_1}) \land \dots \land \gamma_n(\boldsymbol{x}_n, \boldsymbol{y}_{\gamma_n}) \land \psi, \tag{2}$$

for all $\gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_{\gamma_i}) \in \Gamma_i$ and $1 \leq i \leq n$. It thus suffices to show that $(\Pi'', G(\boldsymbol{x}))$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances.

First suppose that $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}(\boldsymbol{a})$, where $\mathcal{D}$ is an arbitrary data instance. Let $\mathcal{D}'$ be the complete data instance obtained from $\mathcal{D}$ by adding the ground atoms $S(\boldsymbol{c})$, for S from Σ and $\boldsymbol{c} \subseteq \mathsf{ind}(\mathcal{D})$, if $\mathcal{O} \models \gamma(\boldsymbol{x}, \boldsymbol{y}) \to S(\boldsymbol{x})$ and $\gamma(\boldsymbol{c}, \boldsymbol{b}) \in \mathcal{D}$, for some atom $\gamma(\boldsymbol{x}, \boldsymbol{y})$ and some $\boldsymbol{b} \subseteq \mathsf{ind}(\mathcal{D})$. Clearly, $\mathcal{O}, \mathcal{D}' \models \boldsymbol{q}(\boldsymbol{a})$, so we must have $\Pi, \mathcal{D}' \models G(\boldsymbol{a})$. A simple inductive argument (on the order of derivation of ground atoms) shows that whenever a clause (1) is applied using a substitution $\boldsymbol{c}$ for the variables in the body to derive $Q(\boldsymbol{c}(\boldsymbol{z}))$ using Π, we can find a corresponding clause (2) and a substitution $\boldsymbol{c}'$ extending $\boldsymbol{c}$ (on the fresh variables $\boldsymbol{y}_{\gamma_i}$) that allows us to derive $Q(\boldsymbol{c}'(\boldsymbol{z}))$ using Π''.

Indeed, for each $1 \leq i \leq n$, we have $S_i(\boldsymbol{c}(\boldsymbol{x}_i)) \in \mathcal{D}'$, so, there must exist some $\gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_{\gamma_i})$ such that $\mathcal{O} \models \gamma_i(\boldsymbol{x}_i, \boldsymbol{y}_{\gamma_i}) \to S(\boldsymbol{x})$ and $\gamma_i(\boldsymbol{c}(\boldsymbol{x}_i), \boldsymbol{b}_i) \in \mathcal{D}$, for some $\boldsymbol{b}_i \subseteq \mathsf{ind}(\mathcal{D})$; so, we set $\boldsymbol{c}'(\boldsymbol{y}_{\gamma_i}) = \boldsymbol{b}_i$. It then suffices to choose the clause (2) whose atoms match the ground atoms $\gamma_i(\boldsymbol{c}(\boldsymbol{x}_i), \boldsymbol{b}_i) \in \mathcal{D}$.

For the converse direction, observe that $\Pi \subseteq \Pi''$.

To complete the proof, we note that it is in NL to decide whether an atom belongs to Γ_i, and thus we can construct the program Π' by means of an L^{NL}-transducer.

B Correctness of $\Pi_{\boldsymbol{Q}}^{\mathrm{LOG}}$

Lemma 3. *Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$. For any data instance $\mathcal{D}$ complete for $\mathcal{O}$, any $D \in \mathfrak{R}$, any type $\boldsymbol{w}$ with $\mathsf{dom}(\boldsymbol{w}) = \partial D$ and any $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|\mathit{var}(\boldsymbol{w})| + |\boldsymbol{x}_D|}$, we have $\Pi_{\boldsymbol{Q}}^{\mathrm{LOG}}, \mathcal{D} \models G_D^{\boldsymbol{w}}(\boldsymbol{b})$ iff there is a homomorphism $h\colon \boldsymbol{q}_D \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that*

$$h(z) = \begin{cases} \boldsymbol{b}(z), & \text{for all } z \in \boldsymbol{x}_D \text{ and all } z \in \partial D \text{ with } \boldsymbol{w}(z) = \varepsilon, \\ \boldsymbol{w}(z)[\boldsymbol{a}/\boldsymbol{b}(\widetilde{\boldsymbol{a}})], & \text{for all } z \in \partial D \text{ with } \boldsymbol{w}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases} \tag{3}$$

Proof. ($\Rightarrow$) The proof is by induction on $\prec$. For the basis of induction, let D be of size 1, that is, $D = \{t\}$. Then $\sigma(D) = t$. Suppose that $\Pi_{\boldsymbol{Q}}^{\mathrm{LOG}}, \mathcal{D} \models G_D^{\boldsymbol{w}}(\boldsymbol{b})$. Then there must exist a type $\boldsymbol{s}$ with $\mathsf{dom}(\boldsymbol{s}) = \lambda(\sigma(D))$ such that $\boldsymbol{s}$ is compatible with $\sigma(D)$ and agrees with $\boldsymbol{w}$ on their common domain (which is ∂D), and $\mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$ for some $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|\mathit{var}(\boldsymbol{s})| + |\boldsymbol{x}_{\sigma(D)}|}$ extending $\boldsymbol{b}$ (note that since $D = \{t\}$, we have $\boldsymbol{x}_{\sigma(D)} = \boldsymbol{x}_D$).

We define a homomorphism $h\colon \boldsymbol{q}_D \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ by taking, for each variable z in $\boldsymbol{q}_D$,

$$h(z) = \begin{cases} \boldsymbol{e}(z), & \text{if } z \in \boldsymbol{x}_D \text{ or } \boldsymbol{s}(z) = \varepsilon, \\ \boldsymbol{s}(z)[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})], & \text{if } \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases}$$

Since $\boldsymbol{b}(z) = \boldsymbol{e}(z)$ for all z in $\mathit{var}(\boldsymbol{w})$ and $\boldsymbol{s}$ agrees with $\boldsymbol{w}$ on ∂D, we obtain (3).

It remains to show that h is a homomorphism. Let $S(\boldsymbol{z}) \in \boldsymbol{q}_D$ with $\boldsymbol{z} \subseteq \lambda(\sigma(D))$. We have the following three options.

- If $\boldsymbol{s}(\boldsymbol{z}) \subseteq \{\varepsilon\}$, then $h(z) = \boldsymbol{e}(z)$ for all $z \in \boldsymbol{z}$, and $S(\boldsymbol{e}(\boldsymbol{z})) \in \mathcal{D}$ since $S(\boldsymbol{z})$ occurs in the (d$'$)-component of $\mathsf{At}^{\boldsymbol{s}}$. It follows that $S(h(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\mathcal{D}}$.
- If neither $\boldsymbol{s}(\boldsymbol{z}) \subseteq \{\varepsilon\}$ nor $\boldsymbol{s}(\boldsymbol{z}) \cap \{\varepsilon\} = \emptyset$, then, since $\boldsymbol{s}$ is compatible with bag t, there is $P(\boldsymbol{a})$ such that $\boldsymbol{z} = \boldsymbol{z}' \cup \boldsymbol{z}''$ for $\boldsymbol{z'} = \{z \in \boldsymbol{z} \mid \boldsymbol{s}(z) = \varepsilon\} \neq \emptyset$ and $\boldsymbol{z''} = \{z \in \boldsymbol{z} \mid \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))\} \neq \emptyset$. Observe that $h(\boldsymbol{z}') = \boldsymbol{e}(\boldsymbol{z}')$ and $h(\boldsymbol{z}'') = \boldsymbol{s}(\boldsymbol{z''})[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$. Because of the (b)-component of $\mathsf{At}^{\boldsymbol{s}}$, atom $P(\boldsymbol{e}(\widetilde{\boldsymbol{a}}))$ is in $\mathcal{D}$ and there is a grounding function $g\colon \boldsymbol{z'} \to \boldsymbol{a}$ such that $S(\boldsymbol{z})[\boldsymbol{z'}/g(\boldsymbol{z'}), \boldsymbol{z''}/\boldsymbol{s}(\boldsymbol{z''})] \in \mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$ and, for all $z \in \boldsymbol{z'}$ with $g(z) = a$, we have $\boldsymbol{e}(z) = \boldsymbol{e}(\widetilde{a})$. It follows that we have $\boldsymbol{e}(\boldsymbol{z'}) = g(\boldsymbol{z}')[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$. Therefore, $h(\boldsymbol{z})$ is the composition of two substitutions, $[\boldsymbol{z'}/g(\boldsymbol{z'}), \boldsymbol{z''}/\boldsymbol{s}(\boldsymbol{z''})]$ and $[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$, whence $S(h(\boldsymbol{z}))$ is generated by $P(\boldsymbol{e}(\widetilde{\boldsymbol{a}}))$.
- If $\boldsymbol{s}(\boldsymbol{z}) \cap \{\varepsilon\} = \emptyset$, then, since $\boldsymbol{s}$ is compatible with bag t, there exists $P(\boldsymbol{a})$ such that $\boldsymbol{s}(\boldsymbol{z}) \subseteq \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$. We also have $S(\boldsymbol{s}(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$. By the (i$'$)-component of $\mathsf{At}^{\boldsymbol{s}}$, we have $P(\boldsymbol{e}(\widetilde{\boldsymbol{a}})) \in \mathcal{D}$. It follows that $S(\boldsymbol{s}(\boldsymbol{z})[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]) \in \mathfrak{C}_{\mathcal{O},\mathcal{D}}$.

For the inductive step, suppose that $\Pi_{\boldsymbol{Q}}^{\text{LOG}}, \mathcal{D} \models G_D^{\boldsymbol{w}}(\boldsymbol{b})$ for some $D \in \mathfrak{R}$, type $\boldsymbol{w}$ with $\mathsf{dom}(\boldsymbol{w}) = \partial D$ and $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{w})|+|\boldsymbol{x}_D|}$. By the definition of $\Pi_{\boldsymbol{Q}}^{\text{LOG}}$, there exists a type $\boldsymbol{s}$ such that $\mathsf{dom}(\boldsymbol{s}) = \lambda(\sigma(D))$ and $\boldsymbol{w}$ agrees with $\boldsymbol{s}$ on their common domain and tuples $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{s})|+|\boldsymbol{x}_{\sigma(D)}|}$ such that $\boldsymbol{e}$ agrees with $\boldsymbol{b}$ on their common variables, and

$$\Pi_{\boldsymbol{Q}}^{\text{LOG}}, \mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e}) \wedge \bigwedge\nolimits_{D' \prec D} G_{D'}^{(\boldsymbol{s} \cup \boldsymbol{w}) \restriction_{\partial D'}}(\boldsymbol{b}_{D'}),$$

where $\boldsymbol{b}_{D'}$ is the restriction of $\boldsymbol{e} \cup \boldsymbol{b}$ to $var((\boldsymbol{s} \cup \boldsymbol{w}) \restriction_{\partial D'}) \cup \boldsymbol{x}_{D'}$. By the induction hypothesis, for each $D' \prec D$, there is a homomorphism $h_{D'} \colon \boldsymbol{q}_{D'} \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that (3) is satisfied.

Let us show that the $h_{D'}$ agree on common variables. Suppose that z is shared by $\boldsymbol{q}_{D'}$ and $\boldsymbol{q}_{D''}$ for $D' \prec D$ and $D'' \prec D$. By the definition of tree decomposition, for every variable z, the nodes $\{t \mid z \in \lambda(t)\}$ induce a connected subtree of T, and so $z \in \lambda(\sigma(D)) \cap \lambda(t') \cap \lambda(t'')$, where t' and t'' are the unique neighbours of $\sigma(D)$ lying in D' and D'', respectively. Since $\boldsymbol{w}' = (\boldsymbol{w} \cup \boldsymbol{s}) \restriction_{\partial D'}$ and $\boldsymbol{w}'' = (\boldsymbol{w} \cup \boldsymbol{s}) \restriction_{\partial D''}$ are the restrictions of $\boldsymbol{w} \cup \boldsymbol{s}$, we have $\boldsymbol{w}'(z) = \boldsymbol{w}''(z)$. By (3) for $h_{D'}$ and $h_{D''}$, we obtain $h_{D'}(z) = (\boldsymbol{e} \cup \boldsymbol{b})(z) = h_{D''}(z)$ if $\boldsymbol{w}'(z) = \boldsymbol{w}''(z) = \varepsilon$, and $h_{D'}(z) = \boldsymbol{w}'(z)[(\boldsymbol{e} \cup \boldsymbol{b})/(\boldsymbol{e} \cup \boldsymbol{b})(\widetilde{\boldsymbol{a}})] = \boldsymbol{w}''(z)[(\boldsymbol{e} \cup \boldsymbol{b})/(\boldsymbol{e} \cup \boldsymbol{b})(\widetilde{\boldsymbol{a}})] = h_{D''}(z)$ if $\boldsymbol{w}'(z) = \boldsymbol{w}''(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$.

We now define h by taking, for each variable z in $\boldsymbol{q}_D$,

$$h(z) = \begin{cases} h_{D'}(z) & \text{if } z \in \lambda(t), \text{ for } z \in D' \text{ and } D' \prec D, \\ \boldsymbol{e}(z), & \text{if } z \in \boldsymbol{x}_D \text{ or } z \in \lambda(\sigma(D)) \text{ and } \boldsymbol{s}(z) = \varepsilon, \\ \boldsymbol{s}(z)[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})], & \text{if } z \in \lambda(\sigma(D)) \text{ and } \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})), \end{cases}$$

If follows that h is well-defined and satisfies (3), and that h is a homomorphism from $\boldsymbol{q}_D$ to $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$. Indeed, take an atom $S(\boldsymbol{z}) \in \boldsymbol{q}_D$. Then either $\boldsymbol{z} \subseteq \lambda(\sigma(D))$, in which case $S(h(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ since $\boldsymbol{s}$ is compatible with $\sigma(D)$ and $\Pi_{\boldsymbol{Q}}^{\text{LOG}}, \mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$ (the argument is similar to the base case), or $S(\boldsymbol{z}) \in \boldsymbol{q}_{D'}$ for some $D' \prec D$, in which case we use the fact that h extends homomorphism $h_{D'}$.

($\Leftarrow$) The proof is by induction on $\prec$. For the basis of induction, fix D of size 1 and $\boldsymbol{w}$ with $\mathsf{dom}(\boldsymbol{w}) = \partial D$. Take $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{w})|+|\boldsymbol{x}_D|}$ and a homomorphism $h \colon \boldsymbol{q}_D \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ satisfying (3). Define a type $\boldsymbol{s}$ and $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{s})|+|\boldsymbol{x}_{\sigma(D)}|}$ in the following manner. First, let $\boldsymbol{e}$ coincide with $\boldsymbol{b}$ on $\boldsymbol{x}_{\sigma(D)}$ and set $\boldsymbol{s}(x) = \varepsilon$ for $x \in \boldsymbol{x}_{\sigma(D)}$. Then, for all $z \in \lambda(\sigma(D)) \setminus \boldsymbol{x}_D$, if $h(z) \in \mathsf{ind}(\mathcal{D})$, then we set $\boldsymbol{s}(z) = \varepsilon$ and $\boldsymbol{e}(z) = h(z)$. Otherwise, we have $h(z) = f(\boldsymbol{a})[\eta]$, for $f(\boldsymbol{a}) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$ and η that maps $\boldsymbol{a}$ to $\mathsf{ind}(\mathcal{D})$. So, we set $\boldsymbol{s}(z) = f(\boldsymbol{a})$ and $\boldsymbol{e}(\widetilde{\boldsymbol{a}}) = \eta(\boldsymbol{a})$. By definition, $\mathsf{dom}(\boldsymbol{s}) = \lambda(\sigma(D))$ and, by (3), $\boldsymbol{s}$ and $\boldsymbol{w}$ agree on the common domain. Since h is a homomorphism, $\Pi_{\boldsymbol{Q}}^{\text{LOG}}, \mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$. Indeed, the (d$'$)-part of $\mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$ is true because of completeness of $\mathcal{D}$, while the (b$'$)- and (i$'$)-parts hold by the definition of the canonical model. Therefore $\Pi_{\boldsymbol{Q}}^{\text{LOG}}, \mathcal{D} \models G_D^{\boldsymbol{w}}(\boldsymbol{b})$.

For the inductive step, D is not necessary of size 1. In this case we construct $\boldsymbol{s}$ and $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{s})|+|\boldsymbol{x}_{\sigma(D)}|}$ exactly like in the basis of induction. In addition, for each $D' \prec D$, let $h_{D'}$ be the restriction of h to $\boldsymbol{q}_{D'}$ and let $\boldsymbol{b}_{D'}$ be the restriction of $\boldsymbol{b} \cup \boldsymbol{e}$ to

$\mathit{var}(\partial D') \cup \boldsymbol{x}_{D'}$. By the inductive hypothesis, we have $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{D} \models G^{\boldsymbol{w}'}_{D'}(\boldsymbol{b}_{D'})$. Then, by the argument similar to the basis of induction, we obtain $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$, whence $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{D} \models G^{\boldsymbol{w}}_{D}(\boldsymbol{b})$.

C Correctness of $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$

Lemma 4. *Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$. For any data instance $\mathcal{D}$ complete for $\mathcal{O}$, any predicate $G^{\boldsymbol{w}}_n$ and any tuple $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|\mathit{var}(\boldsymbol{w})| + |\boldsymbol{x}^{\geq n}|}$, we have $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{D} \models G^{\boldsymbol{w}}_n(\boldsymbol{b})$ iff there is a homomorphism $h\colon \boldsymbol{q}_n \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that*

$$h(z) = \begin{cases} \boldsymbol{b}(z), & \text{for all } z \in \boldsymbol{x}^{\geq n} \text{ and all } z \in \boldsymbol{z}^n_{\exists} \text{ with } \boldsymbol{w}(z) = \varepsilon, \\ \boldsymbol{w}(z)[\boldsymbol{a}/\boldsymbol{b}(\widetilde{\boldsymbol{a}})], & \text{for all } z \in \boldsymbol{z}^n_{\exists} \text{ with } \boldsymbol{w}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases} \tag{4}$$

Proof. ($\Rightarrow$) The proof is by induction on n from M to 0.

For the base case ($n = M$), first suppose that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{D} \models G^{\boldsymbol{w}}_M(\boldsymbol{b}, \boldsymbol{p})$, for some type $\boldsymbol{w}$ for $\boldsymbol{z}^M$ and some $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|\mathit{var}(\boldsymbol{w})|}$ and $\boldsymbol{p} \in \mathsf{ind}(\mathcal{D})^{|\boldsymbol{x}^{\geq n}|}$. The only rules in $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ with head predicate $G^{\boldsymbol{w}}_M$ are $G^{\boldsymbol{w}}_M(\mathit{var}(\boldsymbol{w}), \boldsymbol{x}^{\geq M}) \leftarrow \mathsf{At}^{\boldsymbol{s}}(\mathit{var}(\boldsymbol{s}), \boldsymbol{x}^M)$, for some type $\boldsymbol{s}$ for $\boldsymbol{y}^M$ that is compatible with $\boldsymbol{y}^M$ and extends $\boldsymbol{w}$. Therefore, there ist a type $\boldsymbol{s}$ for $\boldsymbol{y}^M$ that is compatible with $\boldsymbol{y}^M$ and extends $\boldsymbol{w}$ such that $\mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$ for some $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|\mathit{var}(\boldsymbol{s})| + |\boldsymbol{x}^{\geq M}|}$ extending $\boldsymbol{b}$.

We now define a homomorphism h from $\boldsymbol{q}_M$ to $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$. We set $h(z) = \boldsymbol{e}(z)$ for z with $\boldsymbol{s}(z) = \varepsilon$ and $h(z) = \boldsymbol{s}(z)[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$ for z with $\boldsymbol{s}(v) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$. Note that since $\boldsymbol{s}$ extends $\boldsymbol{w}$, we have $\boldsymbol{b}(z) = \boldsymbol{e}(z)$ for all $z \in \mathit{var}(\boldsymbol{w})$. Therefore, we have (4) for all $z \in \boldsymbol{z}^M_{\exists} \cup \boldsymbol{x}^{\geq M}$.

It remains to show that h is indeed a homomorphism. Let $S(\boldsymbol{z}) \in \boldsymbol{q}_M$ with $\boldsymbol{z} \subseteq \boldsymbol{y}^M$. Then we have the following options.

- If $\boldsymbol{s}(z) \subseteq \{\varepsilon\}$, Then $h(\boldsymbol{z}) = \boldsymbol{e}(\boldsymbol{z})$ and $S(\boldsymbol{e}(\boldsymbol{z})) \in \mathcal{D}$ because of the (d$'$)-component of At^s. It follows that $S(\boldsymbol{h}(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\mathcal{D}}$.
- If neither $\boldsymbol{s}(\boldsymbol{z}) \subseteq \{\varepsilon\}$ nor $\boldsymbol{s}(\boldsymbol{z}) \cap \{\varepsilon\} = \emptyset$, then, since $\boldsymbol{s}$ is compatible with bag $\boldsymbol{y}^M$, there is $P(\boldsymbol{a})$ such that $\boldsymbol{z} = \boldsymbol{z}' \cup \boldsymbol{z}''$ for $\boldsymbol{z}' = \{z \in \boldsymbol{z} \mid \boldsymbol{s}(z) = \varepsilon\} \neq \emptyset$ and $\boldsymbol{z}'' = \{z \in \boldsymbol{z} \mid \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))\} \neq \emptyset$. Observe that $h(\boldsymbol{z}') = \boldsymbol{e}(\boldsymbol{z}')$ and $h(\boldsymbol{z}'') = \boldsymbol{s}(\boldsymbol{z}'')[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$. Because of the (b)-component of $\mathsf{At}^{\boldsymbol{s}}$, atom $P(\boldsymbol{e}(\widetilde{\boldsymbol{a}}))$ is in $\mathcal{D}$ and there is a grounding function $g\colon \boldsymbol{z}' \to \boldsymbol{a}$ such that $S(\boldsymbol{z})[\boldsymbol{z}'/g(\boldsymbol{z}'), \boldsymbol{z}''/\boldsymbol{s}(\boldsymbol{z}'')] \in \mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$ and, for all $z \in \boldsymbol{z}'$ with $g(z) = a$, we have $\boldsymbol{e}(z) = \boldsymbol{e}(\widetilde{a})$. It follows that we have $\boldsymbol{e}(\boldsymbol{z}') = g(\boldsymbol{z}')[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$. Therefore, $h(\boldsymbol{z})$ is the composition of two substitutions, $[\boldsymbol{z}'/g(\boldsymbol{z}'), \boldsymbol{z}''/\boldsymbol{s}(\boldsymbol{z}'')]$ and $[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]$, whence $S(h(\boldsymbol{z}))$ is generated by $P(\boldsymbol{e}(\widetilde{\boldsymbol{a}}))$.
- If $\boldsymbol{s}(\boldsymbol{z}) \cap \{\varepsilon\} = \emptyset$, then, since $\boldsymbol{s}$ is compatible with $\boldsymbol{y}^M$, there is $P(\boldsymbol{a})$ such that $\boldsymbol{s}(z) \subseteq \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a}))$. We also have $S(\boldsymbol{s}(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\{P(\boldsymbol{a})\}}$. Note that $P(\boldsymbol{e}(\widetilde{\boldsymbol{a}})) \in \mathcal{D}$ due to the (d$'$)-component of At^s. It follows that $S(\boldsymbol{s}(\boldsymbol{z})[\boldsymbol{a}/\boldsymbol{e}(\widetilde{\boldsymbol{a}})]) \in \mathfrak{C}_{\mathcal{O},\mathcal{D}}$.

For the inductive step, suppose that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{D} \models G^{\boldsymbol{w}}_n(\boldsymbol{b})$. By the definition of $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$, there exist a type $\boldsymbol{s}$ for $\boldsymbol{y}^n$ compatible with $\boldsymbol{y}^n$ such that $\boldsymbol{w}$ agrees with $\boldsymbol{s}$ on their common domain and a tuple $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{s})|+|\boldsymbol{x}^n|}$ such that $\boldsymbol{e}(z) = \boldsymbol{b}(z)$ for all $z \in var(\boldsymbol{w}) \cup \boldsymbol{x}^n$, and

$$\Pi_{\boldsymbol{Q}}^{\textsc{Lin}}, \mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e}) \wedge G_{n+1}^{\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}}(\boldsymbol{b}'),$$

where $\boldsymbol{b}'$ is the restriction of $\boldsymbol{e}$ to $var(\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}) \cup \boldsymbol{x}^{\geq n+1}$.

By the induction hypothesis, there is a homomorphism $h'\colon \boldsymbol{q}_{n+1} \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that

$$h'(z) = \begin{cases} \boldsymbol{b}'(z), & \text{for all } z \in \boldsymbol{x}^{\geq n+1} \text{ and all } z \in \boldsymbol{z}_{\exists}^{n+1} \text{ with } \boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}(z) = \varepsilon, \\ \boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}(z)[\boldsymbol{a}/\boldsymbol{b}'(\widetilde{\boldsymbol{a}})], & \text{for all } z \in \boldsymbol{z}_{\exists}^{n+1} \text{ with } \boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases}$$

Now we define h on every z in $\boldsymbol{q}_n$ by taking

$$h(z) = \begin{cases} h'(z) & \text{if } z \in \boldsymbol{z}^{n+1}, \\ \boldsymbol{e}(z), & \text{if } z \in \boldsymbol{x}^n \text{ or } z \in \boldsymbol{y}^n \text{ and } \boldsymbol{s}(z) = \varepsilon, \\ \boldsymbol{s}(z)[\boldsymbol{a}/e(\widetilde{\boldsymbol{a}})], & \text{if } z \in \boldsymbol{y}^n \text{ and } \boldsymbol{s}(z) \in \mathsf{term}_{\mathcal{O}}(P(\boldsymbol{a})). \end{cases}$$

If follows that h is well defined and (4) holds for all $z \in \boldsymbol{z}_{\exists}^n \cup \boldsymbol{x}^{\geq n}$. It also follows that h is a homomorphism from $\boldsymbol{q}_n$ to $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$. Indeed, take an atom $S(\boldsymbol{z}) \in \boldsymbol{q}_n$. Then either $\boldsymbol{z} \subseteq \boldsymbol{y}^n$, in which case $S(h(\boldsymbol{z})) \in \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ since $\boldsymbol{s}$ is compatible with $\boldsymbol{y}^n$ and $\Pi_{\boldsymbol{Q}}^{\textsc{Lin}}, \mathcal{D} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{e})$ (the argument is similar to the base case), or $S(\boldsymbol{z}) \in \boldsymbol{q}_{n+1}$, in which case we use the fact that h extends homomorphism h'.

($\Leftarrow$) We proceed by induction on n from M to 0. Take a type $\boldsymbol{w}$ for $\boldsymbol{z}^n$, a tuple $\boldsymbol{b} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{w})|+|\boldsymbol{x}^{\geq n}|}$, and a homomorphism $h\colon \boldsymbol{q}_n \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ such that (4) holds. We define a type $\boldsymbol{s}$ for $\boldsymbol{y}^n$ and a tuple $\boldsymbol{e} \in \mathsf{ind}(\mathcal{D})^{|var(\boldsymbol{s})|+|\boldsymbol{x}^n|}$ coinciding with $\boldsymbol{b}$ on their common domain in the following manner. For each variable $z \in \boldsymbol{y}^n$, if $h(z) \in \mathsf{ind}(\mathcal{D})$, then we set $\boldsymbol{s}(z) = \varepsilon$ and $\boldsymbol{e}(z) = h(z)$; otherwise, $h(z)$ is of the form $f(\boldsymbol{a})[\boldsymbol{d}/\boldsymbol{a}]$, where $f(\boldsymbol{a})$ is a Skolem term and $\boldsymbol{d} \subseteq \mathsf{ind}(\mathcal{D})$, and we set $\boldsymbol{s}(z) = f(\boldsymbol{a})$ and $\boldsymbol{e}(\widetilde{\boldsymbol{a}}) = \boldsymbol{d}$. Since h is a homomorphism, $\boldsymbol{s}$ is compatible with $\boldsymbol{y}^n$. It is also clear that $\boldsymbol{s}$ extends $\boldsymbol{w}$. It follows that the following rule appears in $\Pi_{\boldsymbol{Q}}^{\textsc{Lin}}$

$$G_n^{\boldsymbol{w}}(var(\boldsymbol{w}), \boldsymbol{x}^{\geq n}) \leftarrow \mathsf{At}^{\boldsymbol{s}}(var(\boldsymbol{s}), \boldsymbol{x}^n) \wedge G_{n+1}^{\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}}(var(\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}), \boldsymbol{x}^{\geq n+1})$$

(or $G_M^{\boldsymbol{w}}(var(\boldsymbol{w}), \boldsymbol{x}^{\geq M}) \leftarrow \mathsf{At}^{\boldsymbol{s}}(var(\boldsymbol{s}), \boldsymbol{x}^M)$ for the basis of induction, $n = M$). Since h is a homomorphism, each of the ground atoms obtained by taking an atom from $\mathsf{At}^{\boldsymbol{s}}(var(\boldsymbol{s}), \boldsymbol{x}^n)$ and substituting $\boldsymbol{e}$ for $var(\boldsymbol{s}) \cup \boldsymbol{x}^n$ is present in $\mathcal{D}$. Indeed, the (d′)-part of $\mathsf{At}^{\boldsymbol{s}}$ is true because of completeness of $\mathcal{D}$, while the (b′)- and (i′)-parts hold by the definition of canonical model. By applying the induction hypothesis to the predicate $G_{n+1}^{\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}}$ and the homomorphism $h'\colon \boldsymbol{q}_{n+1} \to \mathfrak{C}_{\mathcal{O},\mathcal{D}}$ obtained by restricting h to the variables of $\boldsymbol{q}_{n+1}$, we obtain $\Pi_{\boldsymbol{Q}}^{\textsc{Lin}}, \mathcal{D} \models G_{n+1}^{\boldsymbol{s}}(\boldsymbol{b}')$, where $\boldsymbol{b}'$ is the restrictions of $\boldsymbol{e}$ to $var(\boldsymbol{s}\restriction_{\boldsymbol{z}^{n+1}}) \cup \boldsymbol{x}^{\geq n+1}$ (this argument is not needed for the basis of induction, $n = M$). Thus we can conclude that $\Pi_{\boldsymbol{Q}}^{\textsc{Lin}}, \mathcal{D} \models G_n^{\boldsymbol{w}}(\boldsymbol{b})$.

ALGORITHM 1: Non-deterministic procedure for answering bounded-leaf OMQs over single-atom instances.

```
Data: bounded-leaf OMQ $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}))$, single-atom data instance $\mathcal{D} = \{\alpha_0\}$, tuple $\boldsymbol{a}$ from
      $\mathsf{ind}(\mathcal{D})$
Result: true if $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}(\boldsymbol{a})$ and false otherwise
fix some (directed) join tree $T$ for $\boldsymbol{q}$, let $\beta_0$ be its root atom;   /* nodes=query atoms */
frontier $\longleftarrow \{(\emptyset, \beta_0)\}$;   /* first map $\beta_0$, no constraints on mapping */
push $\alpha_0$ onto stack;   /* initialize stack with data atom */
countLoop $\longleftarrow 0$;   /* number of iterations of the while loop */
maxLoop $\longleftarrow 4PQ^2\boldsymbol{n}^{\boldsymbol{n}}$;   /* bound on number of iterations */
numNulls $\longleftarrow 0$;   /* total number of nulls created so far */
while frontier $\neq \emptyset$ and countLoop $\leq$ maxLoop do
    countLoop $\longleftarrow$ countLoop $+ 1$;
    guess one of the 3 options;
    if Option 1 then   /* map frontier atom to top atom on stack */
        $\alpha \longleftarrow$ top(stack);
        guess element $(M, \beta)$ from frontier;
        check homomorphism g from $\beta$ to $\alpha$ such that $g(z) = t$ for every $z \mapsto t \in M$ and $g(x)$
          appears in $\alpha_0$ for every $x \in \boldsymbol{x}$;
        remove $(M, \beta)$ from frontier;
        foreach atom $\gamma$ that is a child of $\beta$ in $T$ do   /* add children of $\beta$ to
          frontier */
            $M_\gamma \longleftarrow \{z \mapsto t \mid z \in \mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma)$ and $g(z) = t\}$;
            frontier $\longleftarrow$ frontier $\cup \{(M_\gamma, \gamma)\}$;
    else if Option 2 then   /* push new atom onto stack */
        $\alpha \longleftarrow$ top(stack);
        guess rule $\rho$ that is applicable to $\alpha$ using homomorphism $h$;
        let $\gamma$ be result of applying $\rho$ to $\beta$ using $h$, with fresh nulls starting from numNulls;
        push $\gamma$ (with fresh nulls marked) onto stack;
        numNulls $\longleftarrow$ numNulls $+$ (number of new nulls used in $\gamma$);
    else if Option 3 and |stack| $> 1$ and no marked null in top(stack) occurs in frontier then
        pop top atom from stack;   /* pop atom from stack */
    else return false;
if frontier $= \emptyset$ then   /* all query atoms have been mapped */
    return true;
```

D Answering Bounded-Leaf OMQs on Single-Atom Data

Theorem 6. *Checking whether $\mathcal{O}, \{P(\boldsymbol{a})\} \models \boldsymbol{q}(\boldsymbol{a})$ is in* LOGCFL *for bounded-leaf acyclic CQs $\boldsymbol{q}(\boldsymbol{x})$ and $\mathcal{O}$ consisting of linear TGDs.*

A LOGCFL OMQ answering algorithm is given in Algorithm 1. Intuitively, we non-deterministically construct a homomorphism from $\boldsymbol{q}(\boldsymbol{x})$ to $\mathfrak{C}_{\mathcal{O},\mathcal{D}}$ in a step-by-step manner by traversing its join tree from root to leaves. To track our position in the query we use variable frontier which records information about the atoms to be mapped at the current step together with images of the variables that have been passed from the predecessor nodes. A bound on the number of leaves gives a logarithmic bound on the size of frontier.

We use the stack to store the anonymous part of the canonical model (which is not subject to a logarithmic bound). We assume that the data instance consists of a single atom which we put into stack. Then, we non-deterministically choose one of the following options:

map an atom in frontier across the top atom of the stack;
push onto the stack a new atom that is generated by the atom that was previously on the top of the stack;
pop an atom from the stack (available only if the top atom of the stack is not 'blocked' by the current position of frontier).

We also need a counter to keep the number of potential steps, which is linear with respect to the size of the query.

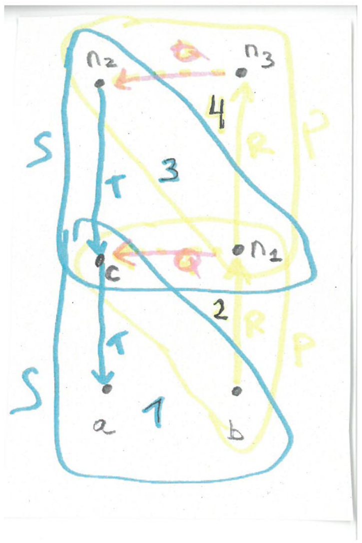
To illustrate, suppose $\mathcal{D} = \{S(a,b,c)\}$ and let $\mathcal{O}$ is the set of the following rules:

$$\begin{aligned}
S(x,y,z) &\to \exists v\, P(z,y,v),\\
P(x,y,z) &\to \exists v\, S(x,z,v),\\
S(x,y,z) &\to T(z,x),\\
P(x,y,z) &\to R(x,y) \land Q(z,x).
\end{aligned}$$

Consider the CQ $\boldsymbol{q}(x_0, x_5)$ depicted below:

$$\begin{tikzcd}
x_0 \arrow[r, "R"] & x_1 \arrow[r, "R"] & x_2 \arrow[r, "Q"] & x_3 \arrow[r, "T"] & x_4 \arrow[r, "T"] & x_5
\end{tikzcd}$$

Then we have $\mathcal{O}, \mathcal{D} \models \boldsymbol{q}$ because of the following fragment of the canonical model:



The algorithm discovers that by starting with stack $= \{S(a,b,c)\}$ and guessing the following sequence of options:

```
 map $x_0$ to $b$
push $P(c, b, n_1)$
 map $x_1$ to $n_1$
push $S(c, n_1, n_2)$
push $P(x_2, n_1, n_3)$
 map $x_2$ to $n_3$
 map $x_3$ to $n_2$
  pop $P(x_2, n_1, n_3)$
 map $x_4$ to $c$
  pop $S(c, n_1, n_2)$
 map $x_5$ to $a$
```

Now we provide a formal proof of correctness of Algorithm 1 and estimate the resources required.

D.1 Proof Trees

Definition 1. *A proof tree for $q(\boldsymbol{a})$ with respect to atom α_0 and linear TGDs $\mathcal{O}$ is a rooted node-labelled tree with the following properties:*

1. *the root node is labelled α_0;*
2. *for every node labelled α that has a child labelled β:*
   - *β can be obtained by applying a single rule $\rho \in \mathcal{O}$ to α;*
   - *if ρ is a rule of the form $\gamma_1(\boldsymbol{x}) \to \exists \boldsymbol{y}\, \gamma_2(\boldsymbol{x}', \boldsymbol{y})$, then fresh nulls are used to instantiate existential positions in β (that is, these nulls cannot be used elsewhere in the tree as the nulls created by another rule application);*
3. *there is a homomorphism of $q(\boldsymbol{a})$ into the set of atoms appearing in the node labels.*

Lemma 7. *If there is a proof tree for $q(\boldsymbol{a})$ w.r.t. atom α_0 and $\mathcal{O}$, then $\mathcal{O}, \{\alpha_0\} \models \boldsymbol{q}(\boldsymbol{a})$.*

Proof. Immediate. Homomorphism of query into the set of atoms of proof tree, which can be homomorphically mapped into the canonical model. □

Lemma 8. *Let P be the number of predicates in $\mathcal{O}$, $\boldsymbol{n}$ be the maximum arity of predicates in $\mathcal{O}$, and Q be the number of atoms in $\boldsymbol{q}$. If $\mathcal{O}, \{\alpha_0\} \models \boldsymbol{q}(\boldsymbol{a})$, then there exists a proof tree for $q(\boldsymbol{a})$ with respect to α_0 and $\mathcal{O}$ of size at most $Q \cdot P \cdot (2\boldsymbol{n})^{\boldsymbol{n}}$.*

Proof. Consider a match of the query in the Skolem chase and the derivation relationships between the atoms of the chase. If the derivation path between two nodes involved in the match contains two isomorphic atoms, then we can shorten the path (i.e. change the match so that the path does not contain any isomorphic atoms). The total number of non-isomorphic atoms is bounded by $P \cdot (2\boldsymbol{n})^{\boldsymbol{n}}$ as there are P choices of predicate, and for every of the (at most $\boldsymbol{n}$) positions, choose either one of the ($\leq \boldsymbol{n}$) constants in α_0 or from at most $\boldsymbol{n}$ different nulls. □

D.2 Proof of Correctness

The following proposition shows that the algorithm is complete.

Proposition 1. *If $\mathcal{O}, \{\alpha_0\} \models \boldsymbol{q}(\boldsymbol{a})$, then some execution of the algorithm on input $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}), \{\alpha_0\}, \boldsymbol{a})$ returns yes.*

Proof. Suppose $\mathcal{O}, \{\alpha_0\} \models \boldsymbol{q}(\boldsymbol{a})$, and let T be a proof tree of size at most $Q \cdot P \cdot (2\boldsymbol{n})^{\boldsymbol{n}}$ (which is guaranteed to exist by the lemma), with h the associated homomorphism of $q(\boldsymbol{a})$.

We will show how to define an execution of the algorithm that returns yes. At the start of the algorithm, we fix some joint tree J for $\boldsymbol{q}(\boldsymbol{a})$, fix some β_0 as root of J, and initialize frontier to $\{(\emptyset, \beta_0)\}$. We then push α_0 onto the stack and give the initial assignments to countLoop, maxLoop, and numNulls.

During the execution of the algorithm, new nulls will be created, with every such null corresponding to some null in the proof tree T. To this end, we will maintain a mapping f from the set of terms occurring in stack to the terms occurring in T, which maps every constant to itself, and every null in stack to some null occurring in T. We will use $f(\gamma)$ to denote the result of replacing each term t in a stack atom γ by $f(t)$. We will show that at every point during the execution of the algorithm, we have the following invariants:

(I0) The mapping f is injective.
(I1) Let $\gamma_1, \ldots, \gamma_k$ be the atoms in the stack (from bottom to top). Then there is a branch of T whose sequence of atoms (starting from the root) begins by $f(\gamma_1), \ldots, f(\gamma_k)$.
(I2) If γ occurs on the stack and contains a marked null n, then $f(\gamma)$ is the atom in T in which $f(n)$ was first created.

We will also incrementally construct a homomorphism g from the subquery $\boldsymbol{q}'$ of $\boldsymbol{q}(\boldsymbol{a})$ consisting of all atoms that have been removed from the frontier at some point so far into the set of atoms S that have been added to stack at some point so far, satisfying the following conditions:

(I3) for every $z \in \mathsf{vars}(\boldsymbol{q}')$, $f(g(z)) = h(z)$;
(I4) if $z \mapsto n$ has appeared in some tuple of the frontier thus far in the execution, then g is defined for z and $g(z) = n$.

Before the first iteration of the while loop, we let f be the function mapping each constant in α_0 to itself, and we let g have empty domain. Clearly, f and g thus defined satisfy the invariants. Let us next suppose that at the current point of execution, the mappings f and g satisfy the invariants. We need to show which non-deterministic choices should be made by the algorithm during the next iteration so that the invariants are preserved. We have the following cases:

Case 1: If α is the atom on the top of the stack and there is a frontier atom $\beta \in \boldsymbol{q}$ such that $h(\beta) = f(\alpha)$, then we will choose option 1. We guess the unique element (M, β) on the frontier that contains the atom β (it is easy to see that an atom can occur in at most one frontier element). We now argue that the required homomorphism from β to α exists. First note that since α is on the stack, we know from the invariant (I0) that f

is injective on the terms in α. This, together with our assumption that $h(\beta) = f(\alpha)$, allows us to extend g to all variables in β by setting $g(z) = f^{-1}(h(z))$ for every variable z in β. Observe that $g(\beta) = \alpha$, as required, and that $g(z) = n$ due to (I4). Further note that if β contains an answer variable x, then $h(x)$ is a constant in α_0, and the same holds for $g(x) = f^{-1}(h(x)) = h(x)$. We have thus shown that the required mapping g exists, and so we can proceed to remove (M, β) from the frontier. We will also add to the frontier the new element (M_γ, γ) for every child γ of β, where $M_\gamma = \{z \mapsto t \mid z \in \mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma) \text{ and } g(z) = t\}$.

We have now completed the current iteration of the while loop, and it remains to show that the invariants continue to hold. As both the stack and the mapping f remain unchanged, the invariants (I0), (I1), and (I2) will remain true. The subquery $\boldsymbol{q}'$ now additionally includes the atom β (as the element (M, β) was removed from the frontier), and we have extended the homomorphism g to this new atom. By setting $g(z) = f^{-1}(h(z))$ for new variables z, we ensure that $f(g(z)) = h(z)$, and thus continue to satisfy (I3). Finally, let us consider a new frontier element (M_γ, γ) where $M_\gamma = \{z \mapsto t \mid z \in \mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma) \text{ and } g(z) = t\}$. If $z \mapsto t \in M_\gamma$, then g is defined for z and such that $g(z) = t$, as required for (I4).

Case 2: If Case 1 does not apply, but there exists a frontier atom β such that $h(\beta)$ is a successor of $f(\alpha)$ in T (with $\alpha = \mathsf{top}(\mathsf{stack})$), then we perform Option 2. If there are multiple atoms β satisfying the preceding condition, then we choose β so that the distance between $h(\beta)$ and $f(\alpha)$ is minimal among all atoms satisfying the property. We then consider the unique child κ of $f(\alpha)$ that lies along the path from $f(\alpha)$ to $h(\beta)$ in T. By the definition of a proof tree, we know that there exists a rule $\rho \in \mathcal{O}$ such that κ is obtained from $f(\alpha)$ by applying ρ using homomorphism ℓ. We construct an atom γ by (i) replacing every term t in κ that is in the range of f by $fw^{-1}(t)$, and (ii) replacing every null in κ that is not part of the range of f by a fresh null, using the earliest available identifiers starting from $\mathsf{numNulls}$. We mark these fresh nulls and push the atom γ onto stack, then update $\mathsf{countLoop}$ and $\mathsf{numNulls}$. To keep track of the new nulls, we extend the mapping f so that each fresh null in γ is mapped to the unique null in T that it replaced.

We now show that the invariants are preserved. By definition, our extension of f preserves injectivity (I0). By assumption, prior to the addition of γ, the sequence of atoms on the stack was mapped by f into an initial segment of a branch of T, ending in $f(\alpha)$. Since $f(\gamma) = \kappa$ is a child of $f(\alpha)$ in T, property (I1) is preserved. Next note every new marked null n in the stack belongs to γ, and $f(\gamma) = \kappa$ is the atom in T in which $f(n)$ was created, yielding (I2). As the homomorphism g and $\mathsf{frontier}$ are left unchanged, the invariants (I3) and (I4) are trivially preserved.

Case 3: If neither Case 1 nor Case 2 applies, then we will perform Option 3. Note that first that Option 3 is applicable. Indeed, if the top atom α contains a marked null n that occurs in some frontier atom β, then we know that $f(n)$ was created in the atom $f(\alpha)$ of T, and hence $h(\beta)$ must be a successor of $f(\alpha)$ (and so we would be in Case 2). We can thus perform Option 3 by popping the atom α from stack. It is easily verified that this action does not affect the satisfaction of the invariants.

We have just shown how to choose and execute, at every iteration of the while loop, one of the three options. It remains to show that the frontier will become empty after at most $\mathsf{maxLoop}$ iterations of the while loop, leading the algorithm to return yes. First observe that whenever we choose Option 1, we 'advance' the frontier by replacing the element corresponding to the selected atom β in T by elements corresponding to the children of β. It follows that we can perform Option 1 at most Q times (where Q is the number of query atoms), and that during the Q-th execution of Option 1, the $\mathsf{frontier}$ will become empty. We next aim to bound the maximum number of iterations of the while loop that may occur prior to the first execution of Option 1, and between one execution of Option 1 and the subsequent execution of Option 1. To this end, we observe that once we have chosen Option 2, then we will continue to perform Option 2 until the conditions for choosing Option 1 are satisfied. Indeed, let us define the distance between the frontier and stack as

$$\min\{d_\beta \mid (M,\beta) \in \mathsf{frontier}, h(\beta) \text{ is successor of } f(\alpha) \text{ and } d_\beta \text{ is the distance from } f(\alpha) \text{ to } h(\beta)\},$$

where $\alpha = \mathsf{top}(\mathsf{stack})$. Then it is easy to see that our strategy for choosing the frontier atom β in Option 2 ensures that the distance decreases by 1 every time we perform Option 2. Thus, we will continue to select Option 2 until the distance is equal to zero (i.e. we have $h(\beta) = f(\alpha)$), in which case Option 1 can be applied. It follows that the number of consecutive applications of Option 2 is bounded by the longest branch in T. As for Option 3, we know that each application of this option pops a symbol from the stack, and that the stack always corresponds, symbol for symbol, to an initial portion of a branch in T. It follows that the maximal number of consequence applications of Option 3 cannot exceed the length of the longest branch in T. Putting this altogether, we have that before the first execution of Option 1, and between every subsequent pair of executions of Option 1, there can be at most $|T|$ executions of Option 3 and $|T|$ executions of Option 2. Thus, the total number of iterations of the while loop cannot exceed $2Q|T|$. Since we have chosen T so that $|T| \leq Q \cdot P \cdot (2\boldsymbol{n})^{\boldsymbol{n}}$, this yields a total of at most $4PQ^2\boldsymbol{n}^{\boldsymbol{n}}$ iterations, as required. □

Proposition 2. *If some execution of the algorithm on input* $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}), \{\alpha_0\}, \boldsymbol{a})$ *returns yes, then* $\mathcal{O}, \{\alpha_0\} \models \boldsymbol{q}(\boldsymbol{a})$.

Proof. Suppose that some execution of the algorithm on input $(\mathcal{O}, \boldsymbol{q}(\boldsymbol{x}), \{\alpha_0\}, \boldsymbol{a})$ returns yes, and fix one particular successful execution. Without loss of generality, we may suppose that the execution has the fewest number of iterations, say N, of the while loop among all successful executions. We show how to use the execution of the algorithm to build a proof tree T for $\boldsymbol{q}(\boldsymbol{a})$ w.r.t. atom α_0 and $\mathcal{O}$, along with a witnessing homomorphism h from $\boldsymbol{q}(\boldsymbol{a})$ to the set of atoms appearing in the node labels of T.

We will use the notation T_i and h_i respectively to refer to the proof tree T and mapping h that we have defined based upon the first i iterations of the while loop, and we will denote by $\boldsymbol{q}'_i$ the set of atoms in $\boldsymbol{q}(\boldsymbol{a})$ that have been removed from the frontier within the first i iterations. Initially, we let T_0 be the proof tree consisting of a single node labelled α_0, h_0 be the empty mapping, and $\boldsymbol{q}'_0$ be the empty query. We prove by induction on $0 \leq i \leq N$ that

(a) T_i satisfies the first two conditions of being a proof tree

(b) h_i is a homomorphism of $\boldsymbol{q}'_i \subseteq \boldsymbol{q}(\boldsymbol{a})$ into (the set of labels of) T_i
(c) if (M, β) belongs to $\mathsf{frontier}$ at the end of iteration i:
  – if $z \mapsto t \in M$, then h_i is defined for z and $h_i(z) = t$;
  – if h_i is defined for z and z appears in β, then M contains $z \mapsto h_i(z)$;
(d) if α has been added to stack at any time before the end of iteration i, then there is a node in T_i labelled α.

These properties clearly hold when $i = 0$, and we will show that they hold for $i = j+1$ assuming that they hold when $i = j$. We have three cases, depending on which option was used for the $(j+1)$st iteration of the while loop.

Option 1 In this case, let α be the atom that was on the top of the stack at the beginning of this iteration, and let (M, β) be the element that was guessed from the frontier. We know that there is a homomorphism g from β to α such that $g(z) = t$ for every $z \mapsto t \in M$ and $g(x)$ appears in α_0 for every $x \in \boldsymbol{x}$. We define h_{j+1} as follows: $h_{j+1}(z) = g(z)$ for every z occurring in β, and $h_{j+1}(z) = h_j(z)$ for every z in the domain of h_j. We note that h_{j+1} is well defined, since if z occurs both in β and in the domain of h_j, then M contains $z \mapsto h_j(z)$, and thus $g(z) = h_j(z)$.

We set $T_{j+1} = T_j$ and note that this means that (a) and (d) continue to hold (for (d), we use the fact that stack is not modified by Option 1). Next we observe that $\boldsymbol{q}'_{j+1} = \boldsymbol{q}'_j \cup \{\beta\}$. We know from our induction hypothesis that h_j is a homomorphism of $\boldsymbol{q}'_j \subseteq \boldsymbol{q}(\boldsymbol{a})$ into T_j. Since h_{j+1} agrees with h_j on their common domain, it follows that h_{j+1} is a homomorphism of $\boldsymbol{q}'_j \subseteq \boldsymbol{q}(\boldsymbol{a})$ into $T_{j+1} = T_j$. Moreover, we know from (d) that T_j contains a node labelled α. As g is a homomorphism from β to α, and h_{j+1} agrees with g on their common domain, it follows that h_{j+1} is a homomorphism of $\{\beta\}$ into T_{j+1}. Putting this together, we can infer that h_{j+1} is a homomorphism of $\boldsymbol{q}'_{j+1}$ into T_{j+1}, so (b) is satisfied.

To show (c), observe that at the end of iteration $j+1$, $\mathsf{frontier}$ contains all elements (M, δ) that were present after iteration j, as well as the new elements correponding to the children of β that were added to the frontier during the application of Option 1. More precisely, for every atom γ that is a child of β in the join tree for $\boldsymbol{q}(\boldsymbol{a})$, we will have added the pair (M_γ, γ) to $\mathsf{frontier}$, where

$$M_\gamma = \{z \mapsto t \mid z \in \mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma) \text{ and } g(z) = t\}.$$

First, consider some pair (M, δ) that was already in $\mathsf{frontier}$ after iteration j. It follows from the induction hypothesis and our definition of h_{j+1} that if $z \mapsto t \in M$, then h_{j+1} is defined for z and $h_{j+1}(z) = t$. Next, suppose that h_{j+1} is defined for z which occurs in δ. If h_j is defined for z, then the induction hypothesis and definition of h_{j+1} yield $z \mapsto h_{j+1}(z)$. If $h_{j+1}(z)$ is defined, but $h_j(z)$ is not defined, then z must occur in β. Because $\delta \neq \beta$ and δ cannot be a descendant of β (since atoms are visited from root to leaves), the connectivity condition on join trees implies that the parent atom of β in the join tree, call it β', contains the variable z. By the way the algorithm is defined, a pair containing the atom β' must already have been added and then later removed from $\mathsf{frontier}$ by the end of iteration j. It follows then from the induction hypothesis that h_j is defined for z, and thus M contains $z \mapsto h_j(z)$, which is the same as $z \mapsto h_{j+1}(z)$. We have thus shown that (c) holds for all pairs (M, δ) that was already in $\mathsf{frontier}$ after iteration j, and it remains to show that the condition holds for the new pairs added during iteration $j+1$. Consider some such pair (M_γ, γ). If $z \mapsto t$ occurs in M_γ, then by construction, $z \in \mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma)$ and $g(z) = t$. Since g and h_{j+1} coincide on their shared domain, we obtain $h_{j+1}(z) = t$, so the first half of (c) is satisfied. To show the second part, suppose that h_{j+1} is defined for z and z appears in γ. Since h_{j+1} is defined for z, the variable z must belong to some atom that has already been removed from the frontier, and given the order in which the join tree is explored, this atom cannot be a descendant of γ. Because of the connectivity condition for join trees, it must be the case that z occurs in β. Since $z \in \mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma)$, we know that M_γ contains $z \mapsto g(z)$, which is equal to $z \mapsto h_{j+1}(z)$. We have thus shown that conditions (a)–(d) all continue to hold at the end of iteration $j+1$.

Option 2 Let α be the atom on that was on the top of the stack at the beginning of this iteration, ρ be the guessed rule that is applicable to α using homomorphism f, and γ be result of applying ρ to β using h, with fresh nulls starting from $\mathsf{numNulls}$. By (d), we know that there is a node n in T_j that is labelled α. We let T_{j+1} be the result of adding a new node n' to T_j as a child of n and labelling this node by γ. Condition (a) holds because of the induction hypothesis and the fact that the label of the new node corresponds to a rule application, with existential variables replaced by fresh nulls, not used anywhere else in T_j. Condition (d) also holds since stack now contains one additional atom, γ, which occurs as a label in T_{j+1}. As Option 2 does not modify $\mathsf{frontier}$, we can trivially satisfy conditions (b) and (c) by setting $h_{j+1} = h_j$.

Option 3 If Option 3 is chosen, then we set $T_{j+1} = T_j$ and $h_{j+1} = h_j$. Since this option does not modify $\mathsf{frontier}$ nor add any new elements to stack, it follows from the induction hypothesis that conditions (a)–(d) continue to hold after iteration $j+1$.

To complete the proof, we observe that $\boldsymbol{q}'_N$ must be equal to $\boldsymbol{q}(\boldsymbol{a})$ (as every query atom must be added and later removed from the frontier at some point during the execution), and thus T_N is a proof tree for $\boldsymbol{q}(\boldsymbol{a})$ with respect to $\{\alpha_0\}$ and $\mathcal{O}$. By Lemma 7, $\mathcal{O}, \{\alpha_0\} \models \boldsymbol{q}(\boldsymbol{a})$. □

D.3 Membership in LogCFL

Proposition 3. *The algorithm can be made to run on a logspace-bounded NAuxPDA.*

Proof. The maximum number of iterations of the while-loop is bounded by $4PQ^2\boldsymbol{n}^{\boldsymbol{n}}$, where P is the number of predicates in $\mathcal{O}$, $\boldsymbol{n}$ is the maximum arity of predicates in $\mathcal{O}$, and Q is the number of atoms in $\boldsymbol{q}$. As we assume that $\boldsymbol{n}$ is bounded by a fixed constant, this gives a polynomial bound on the number of iterations. Next we note that the maximum values of the counters $\mathsf{countLoop}$ and $\mathsf{numNulls}$ are also bounded polynomially in the input, as they increase a constant amount with each iteration of the while loop. Thus, the values of these counters can be stored using logarithmic space. We also note that the algorithm only manipulates a polynomial number of different constants and null values, so each tuple on the frontier can be stored using logarithmic space. Finally, we note that because we are considering tree-shaped queries whose number of leaves is bounded by a fixed constant, there can be at most a constant number of tuples on the frontier at any point during the algorithm. Thus, the whole frontier can be stored using logarithmic space.