The Complexity of Ontology-Based Data Access with OWL 2 QL and Bounded Treewidth Queries

Meghyn Bienvenu
CNRS & University of Montpellier, France

Stanislav Kikot
Birkbeck
University of London, UK

Roman Kontchakov
Birkbeck
University of London, UK

Vladimir V. Podolskii
Steklov Mathematical Institute, Moscow, Russia

Vladislav Ryzhikov
Free University of Bozen-Bolzano, Italy

Michael Zakharyaschev
Birkbeck
University of London, UK

ABSTRACT

Our concern is the overhead of answering *OWL 2 QL* ontology-mediated queries (OMQs) in ontology-based data access compared to evaluating their underlying tree-shaped and bounded treewidth conjunctive queries (CQs). We show that OMQs with bounded-depth ontologies have nonrecursive datalog (NDL) rewritings that can be constructed and evaluated in LOGCFL for combined complexity, even in NL if their CQs are tree-shaped with a bounded number of leaves, and so incur no overhead in complexity-theoretic terms. For OMQs with arbitrary ontologies and bounded-leaf CQs, NDL-rewritings are constructed and evaluated in LOGCFL. We show experimentally feasibility and scalability of our rewritings compared to standard NDL-rewritings. On the negative side, we prove that answering OMQs with tree-shaped CQs is not fixed-parameter tractable if the ontology depth or the number of leaves in the CQs is regarded as the parameter, and that answering OMQs with a fixed ontology (of infinite depth) is NP-complete for tree-shaped and LOGCFL for bounded-leaf CQs.

CCS Concepts

•Information systems→Query languages •Theory of computation → Complexity theory and logic; Description logics •Computing methodologies → Knowledge representation and reasoning

Keywords

Ontology-based data access; ontology-mediated query; query rewriting; combined & parameterised complexity.

PODS 2017 Raleigh, North Carolina, USA

DOI: 10.475/123_4

1. INTRODUCTION

The main aim of ontology-based data access (OBDA) [46, 39] is to facilitate access to complex data for non-expert end-users. The ontology, given by a logical theory $\mathcal{T}$, provides a unified conceptual view of one or more data sources, so the users do not have to know the actual structure of the data and can formulate their queries in the vocabulary of the ontology, which is connected to the data schema by a mapping $\mathcal{M}$. The instance $\mathcal{M}(\mathcal{D})$ obtained by applying $\mathcal{M}$ to a given dataset $\mathcal{D}$ is interpreted under the open-world assumption, and additional facts can be *inferred* using the domain knowledge provided by the ontology. A certain answer to a query $\boldsymbol{q}(\boldsymbol{x})$ over $\mathcal{D}$ is any tuple of constants $\boldsymbol{a}$ such that $\mathcal{T}, \mathcal{M}(\mathcal{D}) \models \boldsymbol{q}(\boldsymbol{a})$. OBDA is closely related to querying incomplete databases under (ontological) constraints, data integration [17], and data exchange [2].

In the classical approach to OBDA [11, 46], the computation of certain answers is reduced to standard database query evaluation: given an ontology-mediated query (OMQ) $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, one constructs a first-order (FO) query $\boldsymbol{q}'(\boldsymbol{x})$, called a rewriting of $\boldsymbol{Q}$, such that, for all datasets $\mathcal{D}$ and mappings $\mathcal{M}$,

$$\mathcal{T}, \mathcal{M}(\mathcal{D}) \models \boldsymbol{q}(\boldsymbol{a}) \quad \Longleftrightarrow \quad \mathcal{I}_{\mathcal{M}(\mathcal{D})} \models \boldsymbol{q}'(\boldsymbol{a}), \qquad (1)$$

where $\mathcal{I}_{\mathcal{M}(\mathcal{D})}$ is the FO-structure comprised of the atoms in $\mathcal{M}(\mathcal{D})$. When the form of $\mathcal{M}$ is appropriately restricted (e.g., $\mathcal{M}$ is a GAV mapping), one can further unfold $\boldsymbol{q}'(\boldsymbol{x})$ using $\mathcal{M}$ to obtain an FO-query that can be evaluated directly over the original dataset $\mathcal{D}$ (so there is no need to materialise $\mathcal{M}(\mathcal{D})$).

For reduction (1) to hold for all OMQs, it is necessary to restrict the expressivity of $\mathcal{T}$ and $\boldsymbol{q}$. The *DL-Lite* family of description logics [11] was specifically designed to ensure (1) for OMQs with conjunctive queries (CQs) $\boldsymbol{q}$. Other ontology languages with this property include linear and sticky tuple-generating dependencies (tgds) [8, 9], and the *OWL 2 QL* profile [41] of the W3C-standardised Web Ontology Language *OWL 2*, the focus of this work. Like many other ontology languages,

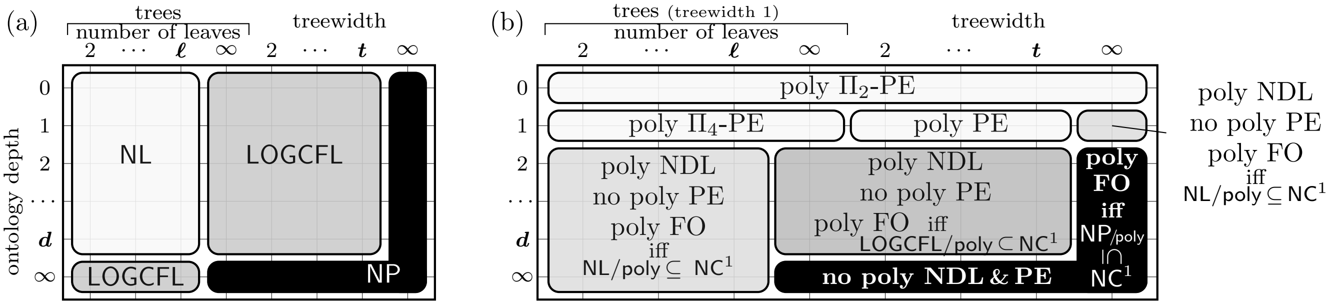


Figure 1: OMQ answering in *OWL 2 QL* (a) combined complexity and (b) the size of rewritings.

OWL 2 QL admits only unary and binary predicates, but arbitrary relational instances can be queried due to the mapping. Various types of FO-rewritings $\boldsymbol{q}'(\boldsymbol{x})$ have been developed and implemented for the preceding languages [46, 43, 37, 49, 13, 18, 48, 34, 25, 40, 36], and a few mature OBDA systems have emerged, including pioneering MASTRO [10], commercial Stardog [44] and Ultrawrap [50], and the Optique platform [21] with the query answering engine Ontop [47, 38].

Our concern here is the overhead of OMQ answering—i.e., checking whether the left-hand side of (1) holds—compared to evaluating the underlying CQs. At first sight, there is no apparent difference between the two problems when viewed through the lens of computational complexity: OMQ answering is in AC^0 for data complexity by (1) and NP-complete for combined complexity [11], which in both cases corresponds to the complexity of evaluating CQs in the relational setting. Further analysis revealed, however, that answering OMQs is already NP-hard for combined complexity when the underlying CQs are tree-shaped (acyclic) [33], which sharply contrasts with the well-known LOGCFL-completeness of evaluating bounded treewidth CQs [56, 12, 24]. This surprising difference motivated a systematic investigation of the combined complexity of OMQ answering along two dimensions: (*i*) the query topology (treewidth $\boldsymbol{t}$ of CQs, and the number $\boldsymbol{\ell}$ of leaves in tree-shaped CQs), and (*ii*) the existential depth $\boldsymbol{d}$ of ontologies (i.e., the length of the longest chain of labelled nulls in the chase on any data). The resulting landscape, displayed in Fig. 1 (a) (under the assumption that datasets are given as RDF graphs and $\mathcal{M}$ is the identity) [11, 33, 31, 5], indicates three tractable cases:

- $\mathsf{OMQ}(\boldsymbol{d},\boldsymbol{t},\infty)$**:** ontologies of depth $\le \boldsymbol{d}$ coupled with CQs of treewidth $\le \boldsymbol{t}$ (for fixed $\boldsymbol{d},\boldsymbol{t}$);
- $\mathsf{OMQ}(\boldsymbol{d},1,\boldsymbol{\ell})$**:** ontologies of depth $\le \boldsymbol{d}$ with tree-shaped CQs with $\le \boldsymbol{\ell}$ leaves (for fixed $\boldsymbol{d},\boldsymbol{\ell}$);
- $\mathsf{OMQ}(\infty,1,\boldsymbol{\ell})$**:** ontologies of arbitrary depth and tree-shaped CQs with $\le \boldsymbol{\ell}$ leaves (for fixed $\boldsymbol{\ell}$).

Observe in particular that when the ontology depth is bounded by a fixed constant, the complexity of OMQ answering is precisely the same as for evaluating the underlying CQs. If we place no restriction on the ontology, then tractability of tree-shaped queries can be recovered by bounding the number of leaves, but we have LOGCFL rather than the expected NL.

While the results in Fig. 1(a) appear to answer the question of the additional cost incurred by adding an *OWL 2 QL* ontology, they only tell part of the story. Indeed, in the context of classical rewriting-based OBDA [46], it is not the abstract complexity of OMQ answering that matters, but the cost of computing and evaluating OMQ rewritings. Fig. 1(b) summarises what is known about the size of positive existential (PE), nonrecursive datalog (NDL) and FO-rewritings [32, 23, 31, 5]. Thus, we see, for example, that PE-rewritings for OMQs from $\mathsf{OMQ}(\boldsymbol{d},\boldsymbol{t},\infty)$ can be of super-polynomial size, and so are not computable and evaluable in polynomial time, even though Fig. 1(a) shows that such OMQs can be answered in LOGCFL. The same concerns $\mathsf{OMQ}(\boldsymbol{d},1,\boldsymbol{\ell})$ and $\mathsf{OMQ}(\infty,1,\boldsymbol{\ell})$, which can be answered in NL and LOGCFL, respectively, but do not enjoy polynomial-size PE-rewritings. Moreover, our experiments show that standard rewriting engines exhibit exponential behaviour on OMQs drawn from $\mathsf{OMQ}(1,1,2)$ lying in the intersection of the three tractable classes.

Our first aim is to show that the positive complexity results in Fig. 1(a) can in fact be achieved using query rewriting. To this end, we develop NDL-rewritings for the three tractable cases that can be computed and evaluated by algorithms of optimal combined complexity. In theory, such algorithms are known to be space efficient and highly parallelisable. We demonstrate practical efficiency of our optimal NDL-rewritings by comparing them with the NDL-rewritings produced by Clipper [18], Presto [49] and Rapid [13], using a sequence of OMQs from the class $\mathsf{OMQ}(1,1,2)$.

Our second aim is to understand the contribution of the ontology depth and the number of leaves in tree-shaped CQs to the complexity of OMQ answering. (As follows from Fig. 1 (a), if these parameters are unbounded, this problem is harder than evaluating the underlying CQs unless LOGCFL = NP.) Unfortunately, it turns out that answering OMQs with ontologies of finite depth and tree-shaped CQs is not fixed-parameter tractable if either the ontology depth or the number of leaves in CQs is regarded as a parameter. More precisely, we prove that the problem is $W[2]$-hard in the former case and $W[1]$-hard in the latter. These results suggest that the ontology depth and the number of leaves are inherently in the exponent of the size of the input in any OMQ answering algorithm.

Finally, we revisit the NP- and LOGCFL-hardness results for OMQs with tree-shaped CQs. The known NP and LOGCFL lower bounds have been established using sequences $(\mathcal{T}_n, \boldsymbol{q}_n)$ of OMQs, where the depth of $\mathcal{T}_n$ grows with n [33, 5]. One might thus hope to make answering OMQs with tree-shaped CQs easier by restricting the ontology signature, size, or even by fixing the whole ontology, which is very relevant for applications as a typical OBDA scenario has users posing different queries over the same ontology. Our third main result is that this is not the case: we present ontologies $\mathcal{T}_\dagger$ and $\mathcal{T}_\ddagger$ of infinite depth such that answering OMQs $(\mathcal{T}_\dagger, \boldsymbol{q})$ with tree-shaped $\boldsymbol{q}$ and $(\mathcal{T}_\ddagger, \boldsymbol{q})$ with linear $\boldsymbol{q}$ is NP- and LOGCFL-hard for query complexity, respectively. We also show that no algorithm can construct FO-rewritings of the OMQs $(\mathcal{T}_\dagger, \boldsymbol{q})$ in polynomial time unless $\mathsf{P} = \mathsf{NP}$, even though polynomial-size FO-rewritings of these OMQs do exist.

The paper is organised as follows. We begin in Section 2 by introducing the *OWL 2 QL* ontology language and key notions like OMQ answering and query rewriting. In Section 3, we first identify fragments of NDL which can be evaluated in LOGCFL or NL, and then we use these results to develop NDL-rewritings of optimal combined complexity for the three tractable cases. Section 4 concerns the parameterised complexity of OMQ answering with tree-shaped CQs. For ontologies of finite depth, we show $W[2]$-hardness (resp. $W[1]$-hardness) when the ontology depth (resp. number of leaves) is taken as the parameter. For the infinite depth case, we show in Section 5 that NP-hardness applies even for a fixed ontology. The final section of the paper presents preliminary experiments comparing our new rewritings to those produced by existing rewriting engines and discusses possible directions for future work.

2. PRELIMINARIES

An *OWL 2 QL ontology* (*TBox* in description logic), $\mathcal{T}$, is a finite set of sentences (*axioms*) of the forms

$$\begin{array}{ll} \forall x(\tau(x) \to \tau'(x)), & \forall x(\tau(x) \land \tau'(x) \to \bot), \\ \forall xy\,(\varrho(x,y) \to \varrho'(x,y)), & \forall xy\,(\varrho(x,y) \land \varrho'(x,y) \to \bot), \\ \forall x\,\varrho(x,x), & \forall x\,(\varrho(x,x) \to \bot), \end{array}$$

where $\tau(x)$ and $\varrho(x,y)$ are defined, using unary predicates A and binary predicates P, by the grammars

$$\begin{array}{rcl} \tau(x) & ::= & \top \mid A(x) \mid \exists y\,\varrho(x,y), \\ \varrho(x,y) & ::= & \top \mid P(x,y) \mid P(y,x). \end{array}$$

When writing ontology axioms, we omit the universal quantifiers and denote by $\boldsymbol{R}_\mathcal{T}$ the set of binary predicates P occurring in $\mathcal{T}$ and their inverses P^-, assuming that $P^{--} = P$. For every $\varrho \in \boldsymbol{R}_\mathcal{T}$, we take a fresh unary predicate A_ϱ and add $A_\varrho(x) \leftrightarrow \exists y\,\varrho(x,y)$ to $\mathcal{T}$. The resulting ontology is said to be in *normal form*, and we assume, without loss of generality, that all our ontologies are in normal form.

A *data instance*, $\mathcal{A}$, is a finite set of unary or binary ground atoms (called an *ABox* in description logic). We denote by $\mathsf{ind}(\mathcal{A})$ the set of individual constants in $\mathcal{A}$ and write $\varrho(a,b) \in \mathcal{A}$ if $P(a,b) \in \mathcal{A}$ and $\varrho = P$, or $P(b,a) \in \mathcal{A}$ and $\varrho = P^-$. We say that $\mathcal{A}$ is *complete* for an ontology $\mathcal{T}$ if $\mathcal{T}, \mathcal{A} \models S(\boldsymbol{a})$ implies $S(\boldsymbol{a}) \in \mathcal{A}$, for any ground atom $S(\boldsymbol{a})$ with $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$.[1]

A *conjunctive query* (CQ) $\boldsymbol{q}(\boldsymbol{x})$ is a formula of the form $\exists \boldsymbol{y}\,\varphi(\boldsymbol{x}, \boldsymbol{y})$, where φ is a conjunction of atoms $S(\boldsymbol{z})$ all of whose variables are among $\mathsf{var}(\boldsymbol{q}) = \boldsymbol{x} \cup \boldsymbol{y}$. We denote by $\mathsf{avar}(\boldsymbol{q})$ the *answer variables* $\boldsymbol{x}$ of $\boldsymbol{q}(\boldsymbol{x})$ and assume, without loss of generality, that CQs contain no constants. We often regard a CQ as the set of its atoms. With every CQ $\boldsymbol{q}$, we associate its *Gaifman graph* $\mathcal{G}$ whose vertices are the variables of $\boldsymbol{q}$ and whose edges are the pairs $\{u, v\}$ such that $P(u,v) \in \boldsymbol{q}$, for some P. We call $\boldsymbol{q}$ *connected* if $\mathcal{G}$ is connected, *tree-shaped* if $\mathcal{G}$ is a tree, and *linear* if $\mathcal{G}$ is a tree with two leaves.

An *ontology-mediated query* (OMQ) is a pair $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, where $\mathcal{T}$ is an ontology and $\boldsymbol{q}(\boldsymbol{x})$ a CQ. A tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$ is a *certain answer* to $\boldsymbol{Q}(\boldsymbol{x})$ over a data instance $\mathcal{A}$ if $\mathcal{I} \models \boldsymbol{q}(\boldsymbol{a})$ for all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$; in this case we write $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$. If $\boldsymbol{x} = \emptyset$, then a certain answer to $\boldsymbol{Q}$ over $\mathcal{A}$ is 'yes' if $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}$ and 'no' otherwise. The *OMQ answering problem* (for a class of OMQs) is to decide whether $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ holds, given an OMQ $\boldsymbol{Q}(\boldsymbol{x})$ (in the class), $\mathcal{A}$ and $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$. If $\mathcal{T}$, $\boldsymbol{q}(\boldsymbol{x})$, and $\mathcal{A}$ are regarded as input, we speak about *combined complexity* of OMQ answering; if $\mathcal{A}$ and $\mathcal{T}$ are regarded as fixed, we speak about *query complexity.*

Every consistent *knowledge base* (KB) $(\mathcal{T}, \mathcal{A})$ has a *canonical model* (or *chase* in database theory) [1] $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ with the property that $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(\boldsymbol{a})$, for all CQs $\boldsymbol{q}(\boldsymbol{x})$ and $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$. In our constructions, we use the following definition of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, where without loss of generality we assume that $\mathcal{T}$ contains no binary predicates P with $\mathcal{T} \models \forall xy\,P(x,y)$. The domain, $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, consists of $\mathsf{ind}(\mathcal{A})$ and the *witnesses* (or *labelled nulls*) of the form $w = a\varrho_1 \dots \varrho_n$, for $n \geq 1$, such that

– $a \in \mathsf{ind}(\mathcal{A})$ and $\mathcal{T}, \mathcal{A} \models \exists y\,\varrho_1(a,y)$;
– $\mathcal{T} \not\models \varrho_i(x,x)$, for $1 \leq i \leq n$;
– $\mathcal{T} \models \exists x\,\varrho_i(x,y) \to \exists z\,\varrho_{i+1}(y,z)$
  but $\mathcal{T} \not\models \varrho_i(x,y) \to \varrho_{i+1}(y,x)$, for $1 \leq i < n$.

We denote by $\boldsymbol{W}_\mathcal{T}$ the set of words $\varrho_1 \dots \varrho_n \in \boldsymbol{R}_\mathcal{T}^*$ satisfying the last two conditions. Every $a \in \mathsf{ind}(\mathcal{A})$ is interpreted in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ by itself, and unary and binary predicates are interpreted as follows:

– $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models A(u)$ iff either $u \in \mathsf{ind}(\mathcal{A})$ and $\mathcal{T}, \mathcal{A} \models A(u)$, or $u = w\varrho$ with $\mathcal{T} \models \exists y\,\varrho(y,x) \to A(x)$;

[1]If the meaning is clear from the context, we use set-theoretic notation for lists.

– $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models P(u,v)$ iff one of the 3 conditions holds: (i) $u,v \in \mathsf{ind}(\mathcal{A})$ and $\mathcal{T},\mathcal{A} \models P(u,v)$; (ii) $u = v$ and $\mathcal{T} \models P(x,x)$; (iii) $\mathcal{T} \models \varrho(x,y) \to P(x,y)$ and either $v = u\varrho$ or $u = v\varrho^-$.

$\mathcal{T}$ is *of depth* ∞ if $\boldsymbol{W}_\mathcal{T}$ is infinite, and *of depth* $\boldsymbol{d} < \infty$ if $\boldsymbol{d}$ is the maximum length of the words in $\boldsymbol{W}_\mathcal{T}$. (Note that the depth of $\mathcal{T}$ is computable in NL; cf. [22, 7] for related results on chase termination for tgds.)

An FO-formula $\boldsymbol{q}'(\boldsymbol{x})$, possibly with equality, is an *FO-rewriting of an OMQ* $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ if, for *any* data instance $\mathcal{A}$ and any tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$,

$$\mathcal{T},\mathcal{A} \models \boldsymbol{q}(\boldsymbol{a}) \qquad \text{iff} \qquad \mathcal{I}_\mathcal{A} \models \boldsymbol{q}'(\boldsymbol{a}), \tag{2}$$

where $\mathcal{I}_\mathcal{A}$ is the FO-structure over the domain $\mathsf{ind}(\mathcal{A})$ such that $\mathcal{I}_\mathcal{A} \models S(\boldsymbol{a})$ iff $S(\boldsymbol{a}) \in \mathcal{A}$, for any ground atom $S(\boldsymbol{a})$. If $\boldsymbol{q}'(\boldsymbol{x})$ is a positive existential formula, we call it a *PE-rewriting of* $\boldsymbol{Q}(\boldsymbol{x})$. A PE-rewriting whose matrix is a Π_k-formula (with respect to $\land$ and $\lor$) is called a Π_k-*rewriting*. The size $|\boldsymbol{q}'|$ of $\boldsymbol{q}'$ is the number of symbols in it.

We also consider rewritings in the form of nonrecursive datalog queries. A *datalog program*, Π, is a finite set of Horn clauses $\forall \boldsymbol{z}\, (\gamma_0 \leftarrow \gamma_1 \land \dots \land \gamma_m)$, 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.

An *NDL query* is a pair $(\Pi, G(\boldsymbol{x}))$, where Π is an NDL program and $G(\boldsymbol{x})$ a predicate. A tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$ is an *answer to* $(\Pi, G(\boldsymbol{x}))$ *over* a data instance $\mathcal{A}$ if $G(\boldsymbol{a})$ holds in the first-order structure with domain $\mathsf{ind}(\mathcal{A})$ obtained by closing $\mathcal{A}$ under the clauses in Π; in this case we write $\Pi, \mathcal{A} \models G(\boldsymbol{a})$. The problem of checking whether $\boldsymbol{a}$ is an answer to $(\Pi, G(\boldsymbol{x}))$ over $\mathcal{A}$ is called the *query evaluation problem*. The *arity of* Π is the maximal arity, $\mathsf{r}(\Pi)$, of predicates in Π. 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. NDL queries are *equivalent* if they have exactly the same answers over any data instance.

An NDL query $(\Pi, G(\boldsymbol{x}))$ is an *NDL-rewriting of an OMQ* $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ *over complete data instances* in case $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\Pi, \mathcal{A} \models G(\boldsymbol{a})$, for any complete $\mathcal{A}$ and any $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$. Rewritings *over arbitrary data instances* are defined by dropping the completeness condition. Given an NDL-rewriting $(\Pi, G(\boldsymbol{x}))$ of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances, we denote by Π^* the result of replacing each predicate S in Π with a fresh IDB predicate S^* of the same arity and adding the clauses

$$\begin{aligned}
A^*(x) &\leftarrow \tau(x), && \text{if } \mathcal{T} \models \tau(x) \to A(x),\\
P^*(x,y) &\leftarrow \varrho(x,y), && \text{if } \mathcal{T} \models \varrho(x,y) \to P(x,y),\\
P^*(x,x) &\leftarrow \top(x), && \text{if } \mathcal{T} \models P(x,x),
\end{aligned}$$

where $\top(x)$ is an EDB predicate for the active domain [29]. Clearly, $(\Pi^*, G(\boldsymbol{x}))$ is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances and $|\Pi^*| \le |\Pi| + |\mathcal{T}|^2$.

Finally, we remark that, in the definition of rewriting, we can assume that $\mathcal{A}$ is *consistent* with $\mathcal{T}$ [8].

3. OPTIMAL NDL-REWRITINGS

To construct theoretically optimal NDL-rewritings for OMQs in the three tractable classes, we first identify two types of NDL queries whose evaluation problems are in NL and LOGCFL for combined complexity.

3.1 NL and LOGCFL fragments of NDL

To simplify the analysis of non-Boolean NDL queries, it is convenient to regard certain variables as parameters to be instantiated with constants from the candidate answer. Formally, an NDL query $(\Pi, G(x_1, \dots, x_n))$ is called *ordered* if each of its IDB predicates Q comes with fixed variables $x_{i_1}, \dots, x_{i_k}$ $(1 \le i_1 < \dots < i_k \le n)$, called the *parameters of* Q, such that (i) every occurrence of Q in Π is of the form $Q(y_1, \dots, y_m, x_{i_1}, \dots, x_{i_k})$, (ii) the parameters of G are $x_1, \dots, x_n$, and (iii) parameters of the head of every clause include all the parameters of the predicates in the body. Observe that Boolean NDL queries are trivially ordered. The *width* $\mathsf{w}(\Pi, G)$ of an ordered (Π, G) is the maximal number of non-parameter variables in a clause of Π.

Example 1. The NDL query $(\Pi, G(x))$, where

$$\Pi = \{\, G(x) \leftarrow R(x,y) \land Q(x),\ Q(x) \leftarrow R(y,x) \,\},$$

is ordered with parameter x and width 1 (the conditions do not restrict the EDB predicate R). Replacing $Q(x)$ by $Q(y)$ in the first clause yields a query that is not ordered in view of (i). A further swap of $Q(x)$ in the second clause with $Q(y)$ would satisfy (i) but not (iii).

As all the NDL-rewritings we construct are ordered, with their parameters being the answer variables, from now on we only consider ordered NDL queries.

Given an NDL query $(\Pi, G(\boldsymbol{x}))$, a data instance $\mathcal{A}$ and a tuple $\boldsymbol{a}$ with $|\boldsymbol{x}| = |\boldsymbol{a}|$, the $\boldsymbol{a}$*-grounding* $\Pi^{\boldsymbol{a}}_\mathcal{A}$ *of* Π *on* $\mathcal{A}$ is the set of ground clauses obtained by first replacing each parameter in Π by the corresponding constant from $\boldsymbol{a}$, and then performing the standard grounding [16] of Π using the constants from $\mathcal{A}$. The size of $\Pi^{\boldsymbol{a}}_\mathcal{A}$ is bounded by $|\Pi| \cdot |\mathcal{A}|^{\mathsf{w}(\Pi,G)}$, and so we can check whether $\Pi, \mathcal{A} \models G(\boldsymbol{a})$ holds in time $poly(|\Pi| \cdot |\mathcal{A}|^{\mathsf{w}(\Pi,G)})$.

3.1.1 Linear NDL in NL

An NDL program is *linear* [1] if the body of its every clause contains at most one IDB predicate.

Theorem 2. *For any* $\mathsf{w} > 0$*, evaluation of linear NDL queries of width* $\le \mathsf{w}$ *is* NL*-complete for combined complexity.*

Proof. Let $(\Pi, G(\boldsymbol{x}))$ be a linear NDL query. Deciding whether $\Pi, \mathcal{A} \models G(\boldsymbol{a})$ is reducible to finding a path to $G(\boldsymbol{a})$ from a certain set X in the grounding graph $\mathfrak{G}$ constructed as follows. The vertices of $\mathfrak{G}$ are the IDB atoms of $\Pi^{\boldsymbol{a}}_{\mathcal{A}}$, and $\mathfrak{G}$ has an edge from $Q(\boldsymbol{c})$ to $Q'(\boldsymbol{c}')$ iff $\Pi^{\boldsymbol{a}}_{\mathcal{A}}$ contains $Q'(\boldsymbol{c}') \leftarrow Q(\boldsymbol{c}) \land S_1(\boldsymbol{c}_1) \land \dots \land S_k(\boldsymbol{c}_k)$ with $S_i(\boldsymbol{c}_i) \in \mathcal{A}$, for $1 \le i \le k$ (we assume $\mathcal{A}$ contains all $c = c$, for $c \in \mathsf{ind}(\mathcal{A})$). The set X consists of all vertices $Q(\boldsymbol{c})$ with IDB predicates Q being of in-degree 0 in the dependency graph of Π for which there is a clause $Q(\boldsymbol{c}) \leftarrow S_1(\boldsymbol{c}_1) \land \dots \land S_k(\boldsymbol{c}_k)$ in $\Pi^{\boldsymbol{a}}_{\mathcal{A}}$ with $S_i(\boldsymbol{c}_i) \in \mathcal{A}$ $(1 \le i \le k)$. Bounding the width of (Π, G) ensures that $\mathfrak{G}$ is of polynomial size and can be constructed by a deterministic Turing machine with read-only input, write-once output and logarithmic-size work tapes. ❑

The transformation * of NDL-rewritings over complete data instances into NDL-rewritings over arbitrary data instances does not preserve linearity. A more involved construction is given in the proof of the following:

Lemma 3. *Fix any* $\mathsf{w} > 0$*. There is an* L^{NL}*-transducer that, for any linear NDL-rewriting* $(\Pi, G(\boldsymbol{x}))$ *of an OMQ* $\boldsymbol{Q}(\boldsymbol{x})$ *over complete data instances with* $\mathsf{w}(\Pi, G) \le \mathsf{w}$*, computes a linear NDL-rewriting* $(\Pi', G(\boldsymbol{x}))$ *of* $\boldsymbol{Q}(\boldsymbol{x})$ *over arbitrary data instances such that* $\mathsf{w}(\Pi', G) \le \mathsf{w}+1$*.*

3.1.2 Skinny NDL in LOGCFL

The complexity class LOGCFL can be defined in terms of *nondeterministic auxiliary pushdown automata* (NAuxPDAs) [14], which are nondeterministic Turing machines with an additional work tape constrained to operate as a pushdown store. Sudborough [53] 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). It is known that LOGCFL can also be defined in terms of logspace-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. Moreover, there is an algorithm that, given such a circuit $\boldsymbol{C}$, computes the output using an NAuxPDA in logarithmic space in the size of $\boldsymbol{C}$ and exponential time in the depth of $\boldsymbol{C}$ [55, pp. 392–397].

We call an NDL query (Π, G) *skinny* if the body of any clause in Π has at most two[2] atoms.

Lemma 4. *For any skinny* $(\Pi, G(\boldsymbol{x}))$ *and data instance* $\mathcal{A}$*, query evaluation can be done by an NAuxPDA in space* $\log |\Pi| + \mathsf{w}(\Pi, G) \cdot \log |\mathcal{A}|$ *and time* $2^{O(\mathsf{d}(\Pi, G))}$*.*

Proof. Define a monotone Boolean circuit $\boldsymbol{C}$ as follows: its output is $G(\boldsymbol{a})$; for every atom γ in the head of a clause in $\Pi^{\boldsymbol{a}}_{\mathcal{A}}$, we take an OR-gate whose output is γ and inputs are the bodies of the clauses with head γ; for every such body, we take an AND-gate whose inputs are the atoms in the body. We set input γ to 1 iff $\gamma \in \mathcal{A}$. Clearly, $\boldsymbol{C}$ is a semi-unbounded fan-in circuit of depth $O(\mathsf{d}(\Pi, G))$ with $O(|\Pi| \cdot |\mathcal{A}|^{\mathsf{w}(\Pi, G)})$ gates. Observing that our $\boldsymbol{C}$ can be computed by a deterministic logspace Turing machine, we conclude that the query evaluation problem can be solved by an NAuxPDA in the required space and time. ❑

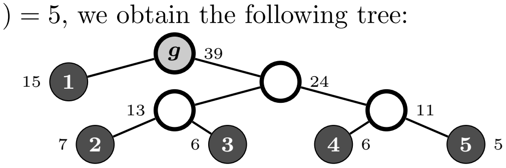
We use weight functions as a means of generalising skinny programs. A function ν from the predicate names in Π to $\mathbb{N}$ is a *weight function for* an NDL query $(\Pi, G(\boldsymbol{x}))$ if $\nu(Q) > 0$ and $\nu(Q) \ge \nu(P_1) + \dots + \nu(P_k)$, for any clause $Q(\boldsymbol{z}) \leftarrow P_1(\boldsymbol{z}_1) \land \dots \land P_k(\boldsymbol{z}_k)$ in Π. First, using the Huffman code, we show that any NDL query can be transformed into an equivalent skinny NDL query whose depth increases by a logarithm of the weight function.

Lemma 5. *Any* $(\Pi, G(\boldsymbol{x}))$ *with a weight function* ν *is equivalent to a skinny* $(\Pi', G(\boldsymbol{x}))$ *with* $|\Pi'| = O(|\Pi|^2)$*,* $\mathsf{w}(\Pi', G) \le \mathsf{w}(\Pi, G)$ *and* $\mathsf{d}(\Pi', G) \le \mathsf{d}(\Pi, G) + \log \nu(G)$*.*

Proof. The proof is by induction on $\mathsf{d}(\Pi, G)$. We take $\Pi' = \Pi$ if $\mathsf{d}(\Pi, G) = 0$. Otherwise, let Π contain a clause ψ of the form $G(\boldsymbol{z}) \leftarrow P_1(\boldsymbol{z}_1) \land \dots \land P_k(\boldsymbol{z}_k)$. Suppose that, for each i $(1 \le i \le k)$, we have an NDL query (Π'_i, P_i) equivalent to (Π, P_i) and such that

$$\begin{aligned} \mathsf{d}(\Pi'_i, P_i) &\le \mathsf{d}(\Pi, P_i) + \log \nu(P_i) \\ &\le \mathsf{d}(\Pi, G) - 1 + \log \nu(P_i). \end{aligned} \tag{3}$$

We construct the Huffman tree [27] for the alphabet $\{1, \dots, k\}$, where the frequency of i is $\nu(P_i)/\nu(G)$ (by definition, $\nu(G) > 0$). For example, for $\nu(G) = 39$, $\nu(P_1) = 15$, $\nu(P_2) = 7$, $\nu(P_3) = 6$, $\nu(P_4) = 6$ and $\nu(P_5) = 5$, we obtain the following tree:



In general, the Huffman tree is a binary tree with k leaves $1, \dots, k$, a root g and $k - 2$ internal nodes and such that the length of the path from g to any leaf i is $\le \lceil \log(\nu(G)/\nu(P_i)) \rceil$. For each internal node v of the tree, we take a predicate $P_v(\boldsymbol{z}_v)$, where $\boldsymbol{z}_v$ is the union of $\boldsymbol{z}_u$ for all descendants u of v; for the root g, we take $P_g(\boldsymbol{z}_g) = G(\boldsymbol{z})$. Let Π'_ψ be the extension of the union of the Π'_i $(1 \le i \le k)$ with clauses $P_v(\boldsymbol{z}_v) \leftarrow P_{u_1}(\boldsymbol{z}_{u_1}) \land P_{u_2}(\boldsymbol{z}_{u_2})$, for each v with immediate successors u_1 and u_2. The number of the new clauses is $k - 1$. By (3), we have:

$$\begin{aligned} \mathsf{d}(\Pi'_\psi, G) &\le \max_i\{\lceil \log(\nu(G)/\nu(P_i)) \rceil + \mathsf{d}(\Pi'_i, P_i)\} \\ &\le \max_i\{\log(\nu(G)/\nu(P_i)) + \mathsf{d}(\Pi, G) + \log \nu(P_i)\} \\ &= \mathsf{d}(\Pi, G) + \log \nu(G). \end{aligned}$$

Let Π' be the result of applying this transformation to each clause in Π with head $G(\boldsymbol{z})$. It is readily seen that (Π', G) is as required; in particular, $|\Pi'| = O(|\Pi|^2)$. ❑

We now use Lemmas 4 and 5 to obtain the following:

[2]In fact, Lemma 4 holds for NDL queries with any *bounded* number of atoms, not only two.

Theorem 6. *For any $c \geq 1$ and $\mathsf{w} \geq 1$, evaluation of NDL queries $(\Pi, G(\boldsymbol{x}))$ having a weight function ν such that $\mathsf{d}(\Pi, G) + \log \nu(G) \leq c \log |\Pi|$ and $\mathsf{w}(\Pi, G) \leq \mathsf{w}$ is in* LOGCFL *for combined complexity.*

We say that a class of OMQs is *skinny-reducible* if, for some fixed $c \geq 1$ and $\mathsf{w} \geq 1$, there is an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer that, given any OMQ $\boldsymbol{Q}(\boldsymbol{x})$ in the class, computes its NDL-rewriting $(\Pi, G(\boldsymbol{x}))$ over complete data instances with a weight function ν such that $\mathsf{d}(\Pi, G) + \log \nu(G) \leq c \log |\Pi|$ and $\mathsf{w}(\Pi, G) \leq \mathsf{w}$. Theorem 6 and the transformation * give the following:

Corollary 7. *Answering OMQs from any skinny-reducible class is in* LOGCFL *for combined complexity.*

We now use the preceding results to construct optimal NDL-rewritings for our three classes of tractable OMQs. Appendix A.6 gives concrete examples of our rewritings.

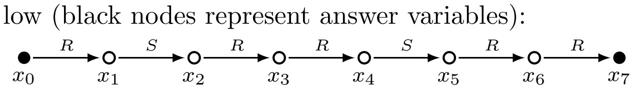
3.2 LOGCFL rewritings for OMQ($\boldsymbol{d}$, $\boldsymbol{t}$, ∞)

Recall (see, e.g., [20]) that a *tree decomposition* of an undirected graph $\mathcal{G} = (V, E)$ is a pair (T, λ), where T is an (undirected) tree and λ a function from the nodes of T to 2^V such that

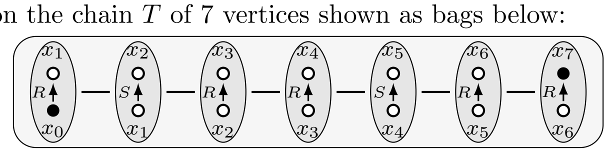
- for every $v \in V$, there exists a node t with $v \in \lambda(t)$;
- for every $e \in E$, there exists a node t with $e \subseteq \lambda(t)$;
- for every $v \in V$, the nodes $\{t \mid v \in \lambda(t)\}$ induce a connected subgraph of T (called a *subtree* of T).

We call the set $\lambda(t) \subseteq V$ a *bag for* t. The *width* of (T, λ) is $\max_{t \in T} |\lambda(t)| - 1$. The *treewidth of a graph* $\mathcal{G}$ is the minimum width over all tree decompositions of $\mathcal{G}$. The *treewidth of a CQ* is the treewidth of its Gaifman graph.

Example 8. Consider the CQ $\boldsymbol{q}(x_0, x_7)$ depicted below (black nodes represent answer variables):



Its natural tree decomposition of treewidth 1 is based on the chain T of 7 vertices shown as bags below:



In this section, we prove the following:

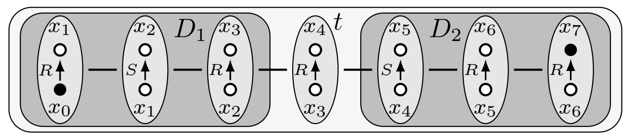
Theorem 9. *For any fixed $\boldsymbol{d} \geq 0$ and $\boldsymbol{t} \geq 1$, the class* $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$ *is skinny-reducible.*

In a nutshell, we split recursively a given CQ $\boldsymbol{q}$ into sub-CQs $\boldsymbol{q}_D$ based on subtrees D of the tree decomposition of $\boldsymbol{q}$, and combine their rewritings into a rewriting of $\boldsymbol{q}$. To guarantee compatibility of these rewritings, we use 'boundary conditions' $\boldsymbol{w}$ that describe the types of points on the boundaries of the $\boldsymbol{q}_D$ and, for each possible boundary condition $\boldsymbol{w}$, we define recursively a fresh IDB predicate $G_D^{\boldsymbol{w}}$. We now formalise the construction and illustrate it using the CQ from Example 8.

Fix a connected CQ $\boldsymbol{q}(\boldsymbol{x})$ and a tree decomposition (T, λ) of its Gaifman graph $\mathcal{G} = (V, E)$. 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 $t' \notin D$. The *degree* $\mathsf{deg}(D)$ of D is the number of its boundary nodes (T itself is the only subtree of T of degree 0). 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 10 ([5]). *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.*

In Example 8, t splits T into D_1 and D_2 as follows:



We define recursively a set $\mathfrak{D}$ of subtrees of T, a binary 'predecessor' relation $\prec$ on $\mathfrak{D}$, and a function σ on $\mathfrak{D}$ indicating the splitting node. We begin by adding T to $\mathfrak{D}$. Take any $D \in \mathfrak{D}$ that has not been split yet. If D is of size 1, then $\sigma(D)$ is the only node of D. Otherwise, by Lemma 10, we find a node t in D that splits it into $D_1, \dots, D_k$. We set $\sigma(D) = t$ and, for $1 \leq i \leq k$, add D_i to $\mathfrak{D}$ and set $D_i \prec D$; then, we apply the procedure recursively to each of $D_1, \dots, D_k$. In Example 8 with t splitting T, we have $\sigma(T) = t$, $D_1 \prec T$ and $D_2 \prec T$.

For each $D \in \mathfrak{D}$, we recursively define a set of atoms

$$\boldsymbol{q}_D \quad = \quad \big\{S(\boldsymbol{z}) \in \boldsymbol{q} \mid \boldsymbol{z} \subseteq \lambda(\sigma(D))\big\} \ \cup \ \bigcup\nolimits_{D' \prec D} \boldsymbol{q}_{D'}.$$

By the definition of tree decomposition, $\boldsymbol{q}_T = \boldsymbol{q}$. Denote by $\boldsymbol{x}_D$ the subset of $\boldsymbol{x}$ that occurs in $\boldsymbol{q}_D$. In Example 8, $\boldsymbol{x}_T = \{x_0, x_7\}$, $\boldsymbol{x}_{D_1} = \{x_0\}$ and $\boldsymbol{x}_{D_2} = \{x_7\}$. Let ∂D be the union of all $\lambda(t) \cap \lambda(t')$ for boundary nodes t of D and its neighbours t' in T *outside* D. In our example, $\partial T = \emptyset$, $\partial D_1 = \{x_3\}$ and $\partial D_2 = \{x_4\}$.

Let $\mathcal{T}$ be an ontology of depth $\leq \boldsymbol{d}$. A *type* is a partial map $\boldsymbol{w}$ from V to $\boldsymbol{W}_{\mathcal{T}}$; its domain is denoted by $\mathsf{dom}(\boldsymbol{w})$. 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 $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, with $\boldsymbol{w}(z) = w$ indicating that z is mapped to an element of the form aw (for some $a \in \mathsf{ind}(\mathcal{A})$), and with $\boldsymbol{w}(z) = \varepsilon$ that z is mapped to an individual constant. We say that a type $\boldsymbol{w}$ is *compatible* with a bag t if, for all $y, z \in \lambda(t) \cap \mathsf{dom}(\boldsymbol{w})$, we have

- if $z \in \boldsymbol{x}$, then $\boldsymbol{w}(z) = \varepsilon$;
- if $A(z) \in \boldsymbol{q}$, then either $\boldsymbol{w}(z) = \varepsilon$ or $\boldsymbol{w}(z) = w\varrho$ with $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$;
- if $P(y, z) \in \boldsymbol{q}$, then one of the three conditions holds: (*i*) $\boldsymbol{w}(y) = \boldsymbol{w}(z) = \varepsilon$; (*ii*) $\boldsymbol{w}(y) = \boldsymbol{w}(z)$ and $\mathcal{T} \models P(x, x)$; (*iii*) $\mathcal{T} \models \varrho(x, y) \to P(x, y)$ and either $\boldsymbol{w}(z) = \boldsymbol{w}(y)\varrho$ or $\boldsymbol{w}(y) = \boldsymbol{w}(z)\varrho^-$.

In the sequel we abuse notation and use sets of variables in place of sequences assuming that they are ordered in some (fixed) way. For example, we use $\boldsymbol{x}_D$ for a tuple of variables in the set $\boldsymbol{x}_D$ (ordered in some way). Also, given a tuple $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}_D|}$ and $x \in \boldsymbol{x}_D$, we write $\boldsymbol{a}(x)$ to refer to the component of $\boldsymbol{a}$ that corresponds to x (that is, the component with the same index).

We now define an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$. For any $D \in \mathfrak{D}$ and type $\boldsymbol{w}$ with $\mathsf{dom}(\boldsymbol{w}) = \partial D$, let $G_D^{\boldsymbol{w}}(\partial D, \boldsymbol{x}_D)$ be a fresh IDB predicate with parameters $\boldsymbol{x}_D$ (note that ∂D and $\boldsymbol{x}_D$ may be not disjoint). For each 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, the NDL program $\Pi_{\boldsymbol{Q}}^{\mathrm{Log}}$ contains

$$G_D^{\boldsymbol{w}}(\partial D, \boldsymbol{x}_D) \leftarrow \mathsf{At}^{\boldsymbol{s}} \ \wedge \bigwedge\nolimits_{D' \prec D} G_{D'}^{(\boldsymbol{s} \cup \boldsymbol{w}) \upharpoonright \partial D'}(\partial D', \boldsymbol{x}_{D'}),$$

where $(\boldsymbol{s} \cup \boldsymbol{w}) \upharpoonright \partial D'$ is the restriction of the union $\boldsymbol{s} \cup \boldsymbol{w}$ to $\partial D'$ (since $\mathsf{dom}(\boldsymbol{s} \cup \boldsymbol{w})$ covers $\partial D'$, the domain of the restriction is $\partial D'$), and $\mathsf{At}^{\boldsymbol{s}}$ is the conjunction of

(a) $A(z)$, for $A(z) \in \boldsymbol{q}$ with $\boldsymbol{s}(z) = \varepsilon$, and $P(y, z)$, for $P(y, z) \in \boldsymbol{q}$ with $\boldsymbol{s}(y) = \boldsymbol{s}(z) = \varepsilon$;

(b) $y = z$, for $P(y, z) \in \boldsymbol{q}$ with $\boldsymbol{s}(y) \neq \varepsilon$ or $\boldsymbol{s}(z) \neq \varepsilon$;

(c) $A_\varrho(z)$, for z with $\boldsymbol{s}(z) = \varrho w$, for some w.

The conjuncts in (a) ensure that atoms all of whose variables are assigned ε hold in the data instance. The conjuncts in (b) ensure that if one variable in a binary atom is not mapped to ε, then the images of both its variables share the same initial individual. Finally, the conjuncts in (c) ensure that if a variable is to be mapped to $a\varrho w$, then $a\varrho w$ is indeed in the domain of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$.

Example 11. With the query in Example 8, consider now the following ontology $\mathcal{T}$:

$$P(x, y) \to S(x, y), \qquad A_P(x) \leftrightarrow \exists y\, P(x, y),$$
$$P(x, y) \to R(y, x), \qquad A_{P^-}(x) \leftrightarrow \exists y\, P(y, x)$$

(the remaining normalisation axioms are omitted). Since $\lambda(t) = \{x_3, x_4\}$, there are two types compatible with t that can contribute to the rewriting: $\boldsymbol{s}_1 = \{x_3 \mapsto \varepsilon,\ x_4 \mapsto \varepsilon\}$ and $\boldsymbol{s}_2 = \{x_3 \mapsto \varepsilon,\ x_4 \mapsto P^-\}$. So we have $\mathsf{At}^{\boldsymbol{s}_1} = R(x_3, x_4)$ and $\mathsf{At}^{\boldsymbol{s}_2} = A_{P^-}(x_4) \wedge (x_3 = x_4)$. Thus, the predicate $G_T^{\boldsymbol{\varepsilon}}$ is defined by two clauses with the head $G_T^{\boldsymbol{\varepsilon}}(x_0, x_7)$ and the following bodies:

$$G_{D_1}^{x_3 \mapsto \varepsilon}(x_3, x_0) \wedge R(x_3, x_4) \wedge G_{D_2}^{x_4 \mapsto \varepsilon}(x_4, x_7),$$
$$G_{D_1}^{x_3 \mapsto \varepsilon}(x_3, x_0) \wedge A_{P^-}(x_4) \wedge (x_3 = x_4) \wedge G_{D_2}^{x_4 \mapsto P^-}(x_4, x_7),$$

for $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, respectively. Although $\{x_3 \mapsto P,\ x_4 \mapsto \varepsilon\}$ is also compatible with t, its predicate $G_{D_1}^{x_3 \mapsto P}$ will have no definition in the rewriting, and hence can be omitted. The same is true of the other compatible types $\{x_3 \mapsto \varepsilon,\ x_4 \mapsto R\}$ and $\{x_3 \mapsto R^-,\ x_4 \mapsto \varepsilon\}$.

By induction on $\prec$, one can now show that $(\Pi_{\boldsymbol{Q}}^{\mathrm{Log}}, G_T^{\boldsymbol{\varepsilon}})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$; see Appendix A.3 for details.

Fix now $\boldsymbol{d}$ and $\boldsymbol{t}$, and consider $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ from $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$. Let T be a tree decomposition of $\boldsymbol{q}$ of treewidth $\leq \boldsymbol{t}$; we may assume w.l.o.g. that T has at most $|\boldsymbol{q}|$ nodes. We take the following weight function: $\nu(G_D^{\boldsymbol{w}}) = |D|$. Clearly, $\nu(G_T^{\boldsymbol{\varepsilon}}) \leq |\boldsymbol{Q}|$. By Lemma 10, $\mathsf{w}(\Pi_{\boldsymbol{Q}}^{\mathrm{Log}}, G_T^{\boldsymbol{\varepsilon}}) \leq |\partial D| \leq 2(\boldsymbol{t}+1)$ and $\mathsf{d}(\Pi_{\boldsymbol{Q}}^{\mathrm{Log}}, G_T^{\boldsymbol{\varepsilon}}) \leq 2\log|T| = 2\log\nu(G_T^{\boldsymbol{\varepsilon}}) \leq 2\log|\boldsymbol{Q}|$. Since $|\mathfrak{D}| \leq |T|^2$ and there are at most $|\mathcal{T}|^{2\boldsymbol{d}(\boldsymbol{t}+1)}$ options for $\boldsymbol{w}$, there are polynomially many predicates $G_D^{\boldsymbol{w}}$, and so $\Pi_{\boldsymbol{Q}}^{\mathrm{Log}}$ is of polynomial size. Thus, by Corollary 7, the constructed NDL-rewriting over arbitrary data instances can be evaluated in LOGCFL. Finally, we note that a tree decomposition of treewidth $\leq \boldsymbol{t}$ can be computed using an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer [24], and so the NDL-rewriting can also be constructed by an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer.

The obtained NDL-rewriting implies that answering OMQs $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathcal{T}$ of finite depth $\boldsymbol{d}$ and $\boldsymbol{q}$ of treewidth $\boldsymbol{t}$ over any data instance $\mathcal{A}$ can be done in time

$$\mathit{poly}(|\mathcal{T}|^{\boldsymbol{dt}},\ |\boldsymbol{q}|,\ |\mathcal{A}|^{\boldsymbol{t}}). \tag{4}$$

3.3 NL rewritings for OMQ($\boldsymbol{d}$, 1, $\boldsymbol{\ell}$)

Theorem 12. *Let $\boldsymbol{d} \geq 0$ and $\boldsymbol{\ell} \geq 2$ be fixed. There is an L^{NL}-transducer that, given an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ in $\mathsf{OMQ}(\boldsymbol{d}, 1, \boldsymbol{\ell})$, constructs its polynomial-size linear NDL-rewriting of width $\leq 2\boldsymbol{\ell}$.*

Let $\mathcal{T}$ be an ontology of finite depth $\boldsymbol{d}$, and let $\boldsymbol{q}(\boldsymbol{x})$ be a tree-shaped CQ with at most $\boldsymbol{\ell}$ leaves. Fix one of the variables of $\boldsymbol{q}$ as root, and let M be the maximal distance to a leaf from the root. For $0 \leq n \leq M$, let $\boldsymbol{z}^n$ denote the set of all variables of $\boldsymbol{q}$ at distance n from the root; clearly, $|\boldsymbol{z}^n| \leq \boldsymbol{\ell}$. We call the $\boldsymbol{z}^n$ *slices* of $\boldsymbol{q}$ and observe that they satisfy the following: for every $P(z, z') \in \boldsymbol{q}$ with $z \neq z'$, there exists $n < M$ such that

$$\text{either } z \in \boldsymbol{z}^n \text{ and } z' \in \boldsymbol{z}^{n+1} \quad \text{or} \quad z' \in \boldsymbol{z}^n \text{ and } z \in \boldsymbol{z}^{n+1}.$$

For $0 \leq n \leq M$, let $\boldsymbol{q}_n(\boldsymbol{z}_\exists^n, \boldsymbol{x}^n)$ be the query consisting of all atoms $S(\boldsymbol{z})$ of $\boldsymbol{q}$ such that $\boldsymbol{z} \subseteq \bigcup_{n \leq k \leq M} \boldsymbol{z}^k$, where $\boldsymbol{x}^n$ is the subset of $\boldsymbol{x}$ that occurs in $\boldsymbol{q}_n$ and $\boldsymbol{z}_\exists^n = \boldsymbol{z}^n \setminus \boldsymbol{x}$.

By a *type for slice* $\boldsymbol{z}^n$, we mean a total map $\boldsymbol{w}$ from $\boldsymbol{z}^n$ to $\boldsymbol{W}_{\mathcal{T}}$. Analogously to Section 3.2, we define the notions of types compatible with slices. Specifically, we call $\boldsymbol{w}$ *locally compatible* with $\boldsymbol{z}^n$ if for every $z \in \boldsymbol{z}^n$:

– if $z \in \boldsymbol{x}$, then $\boldsymbol{w}(z) = \varepsilon$;
– if $A(z) \in \boldsymbol{q}$, then either $\boldsymbol{w}(z) = \varepsilon$ or $\boldsymbol{w}(z) = w\varrho$ with $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$;
– if $P(z, z) \in \boldsymbol{q}$, then either $\boldsymbol{w}(z) = \varepsilon$ or $\mathcal{T} \models P(x, x)$.

If $\boldsymbol{w}, \boldsymbol{s}$ are types for $\boldsymbol{z}^n$ and $\boldsymbol{z}^{n+1}$, respectively, then we say $(\boldsymbol{w}, \boldsymbol{s})$ *is compatible* with $(\boldsymbol{z}^n, \boldsymbol{z}^{n+1})$ if $\boldsymbol{w}$ is locally compatible with $\boldsymbol{z}^n$, $\boldsymbol{s}$ is locally compatible with $\boldsymbol{z}^{n+1}$,

– for every $P(z, z') \in \boldsymbol{q}$ with $z \in \boldsymbol{z}^n$ and $z' \in \boldsymbol{z}^{n+1}$, one of the three condition holds: $\boldsymbol{w}(z) = \boldsymbol{s}(z') = \varepsilon$, or $\boldsymbol{w}(z) = \boldsymbol{s}(z')$ with $\mathcal{T} \models P(x, x)$, or $\mathcal{T} \models \varrho(x, y) \to P(x, y)$ with either $\boldsymbol{s}(z') = \boldsymbol{w}(z)\varrho$ or $\boldsymbol{w}(z) = \boldsymbol{s}(z')\varrho^-$.

Consider the NDL program $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ defined as follows. For every $0 \leq n < M$ and every pair of types $(\boldsymbol{w}, \boldsymbol{s})$ that is compatible with $(\boldsymbol{z}^n, \boldsymbol{z}^{n+1})$, we include the clause

$$G^{\boldsymbol{w}}_n(\boldsymbol{z}^n_\exists, \boldsymbol{x}^n) \leftarrow \mathsf{At}^{\boldsymbol{w}\cup\boldsymbol{s}}(\boldsymbol{z}^n, \boldsymbol{z}^{n+1}) \wedge G^{\boldsymbol{s}}_{n+1}(\boldsymbol{z}^{n+1}_\exists, \boldsymbol{x}^{n+1}),$$

where $\boldsymbol{x}^n$ are the parameters of $G^{\boldsymbol{w}}_n$ and $\mathsf{At}^{\boldsymbol{w}\cup\boldsymbol{s}}(\boldsymbol{z}^n, \boldsymbol{z}^{n+1})$ is the conjunction of atoms (a)–(c) as defined in Section 3.2, for the union $\boldsymbol{w} \cup \boldsymbol{s}$. For every type $\boldsymbol{w}$ locally compatible with $\boldsymbol{z}^M$, we include the clause

$$G^{\boldsymbol{w}}_M(\boldsymbol{z}^M_\exists, \boldsymbol{x}^M) \leftarrow \mathsf{At}^{\boldsymbol{w}}(\boldsymbol{z}^M).$$

(Recall that $\boldsymbol{z}^M$ is a disjoint union of $\boldsymbol{z}^M_\exists$ and $\boldsymbol{x}^M$.) We use G with parameters $\boldsymbol{x}$ as the goal predicate and include $G(\boldsymbol{x}) \leftarrow G^{\boldsymbol{w}}_0(\boldsymbol{z}^0_\exists, \boldsymbol{x})$ for every predicate $G^{\boldsymbol{w}}_0$ occurring in the head of one of the preceding clauses.

By induction on n, we show in Appendix A.4 that $(\Pi^{\text{LIN}}_{\boldsymbol{Q}}, G(\boldsymbol{x}))$ is a rewriting of $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ over complete data instances. It should be clear that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ is a linear NDL program of width $\leq 2\boldsymbol{\ell}$ and containing $\leq |\boldsymbol{q}|\cdot|\mathcal{T}|^{2\boldsymbol{d}\boldsymbol{\ell}}$ predicates. Moreover, it takes only logarithmic space to store a type $\boldsymbol{w}$, which allows us to show that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ can be computed by an L^{NL}-transducer. We apply Lemma 3 to obtain an NDL-rewriting for arbitrary data instances, and then use Theorem 2 to conclude that the resulting program can be evaluated in NL.

The obtained NDL-rewriting implies that answering $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathcal{T}$ of finite depth $\boldsymbol{d}$ and tree-shaped $\boldsymbol{q}$ with $\boldsymbol{\ell}$ leaves over any data $\mathcal{A}$ can be done in time

$$poly(|\mathcal{T}|^{\boldsymbol{d}\boldsymbol{\ell}},\ |\boldsymbol{q}|,\ |\mathcal{A}|^{\boldsymbol{\ell}}). \tag{5}$$

3.4 LOGCFL rewritings for OMQ(∞, 1, ℓ)

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

Theorem 13. *For any fixed $\boldsymbol{\ell} \geq 2$, $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ is skinny-reducible.*

For OMQs with bounded-leaf CQs and ontologies of unbounded depth, our rewriting uses the notion of tree witness [34]. Consider an OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$. Let $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ be a pair of disjoint sets of variables in $\boldsymbol{q}$ such that $\mathfrak{t}_\mathsf{i} \neq \emptyset$ but $\mathfrak{t}_\mathsf{i} \cap \boldsymbol{x} = \emptyset$. Set

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

If $\boldsymbol{q}_\mathfrak{t}$ is a minimal subset of $\boldsymbol{q}$ containing every atom of $\boldsymbol{q}$ with a variable from $\mathfrak{t}_\mathsf{i}$ and such that there is a homomorphism $h\colon \boldsymbol{q}_\mathfrak{t} \to \mathcal{C}_{\mathcal{T},\{A_\varrho(a)\}}$ with $h^{-1}(a) = \mathfrak{t}_\mathsf{r}$, we call $\mathfrak{t}$ a *tree witness for $\boldsymbol{Q}(\boldsymbol{x})$ generated by ϱ*. Intuitively, $\mathfrak{t}$ identifies a minimal subset of $\boldsymbol{q}$ that can be mapped to the tree-shaped part of the canonical model consisting of labelled nulls: the variables in $\mathfrak{t}_\mathsf{r}$ are mapped to an individual constant, say, a, at the root of a tree and the $\mathfrak{t}_\mathsf{i}$ are mapped to the labelled nulls of the form aw, for some $w \in \boldsymbol{W}_\mathcal{T}$ that begins with ϱ. Note that the same tree witness can be generated by different ϱ.

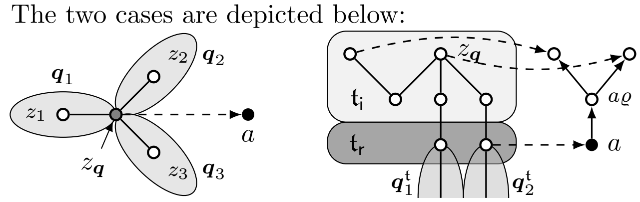
The logarithmic-depth NDL-rewriting for OMQs from $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ is based on the following observation:

Lemma 14 ([31]). *Every tree T of size n has a node splitting it into subtrees of size $\leq \lceil n/2 \rceil$.*

Let $\boldsymbol{Q}(\boldsymbol{x}_0) = (\mathcal{T}, \boldsymbol{q}_0(\boldsymbol{x}_0))$ be an OMQ with a tree-shaped CQ. We will repeatedly apply Lemma 14 to decompose the CQ into smaller and smaller subqueries. Formally, for a tree-shaped CQ $\boldsymbol{q}$, we denote by $z_{\boldsymbol{q}}$ a vertex in the Gaifman graph $\mathcal{G}$ of $\boldsymbol{q}$ that satisfies the condition of Lemma 14; if $|\mathsf{var}(\boldsymbol{q})| = 2$ and $\boldsymbol{q}$ has at least one existentially quantified variable, then we assume that $z_{\boldsymbol{q}}$ is such. Let $\mathfrak{Q}$ be the smallest set that contains $\boldsymbol{q}_0(\boldsymbol{x}_0)$ and the following CQs, for every $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ with existentially quantified variables:

- for each z_i adjacent to $z_{\boldsymbol{q}}$ in $\mathcal{G}$, the CQ $\boldsymbol{q}_i(\boldsymbol{x}_i)$ comprising all binary atoms with both z_i and $z_{\boldsymbol{q}}$, and all atoms whose variables cannot reach $z_{\boldsymbol{q}}$ in $\mathcal{G}$ without passing by z_i, where $\boldsymbol{x}_i$ is the set of variables in $\boldsymbol{x} \cup \{z_{\boldsymbol{q}}\}$ that occur in $\boldsymbol{q}_i$;
- for each tree witness $\mathfrak{t}$ for $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathfrak{t}_\mathsf{r} \neq \emptyset$ and $z_{\boldsymbol{q}} \in \mathfrak{t}_\mathsf{i}$, the CQs $\boldsymbol{q}^\mathfrak{t}_1(\boldsymbol{x}^\mathfrak{t}_1), \dots, \boldsymbol{q}^\mathfrak{t}_k(\boldsymbol{x}^\mathfrak{t}_k)$ that correspond to the connected components of the set of atoms of $\boldsymbol{q}$ that are not in $\boldsymbol{q}_\mathfrak{t}$, where each $\boldsymbol{x}^\mathfrak{t}_i$ is the set of variables in $\boldsymbol{x} \cup \mathfrak{t}_\mathsf{r}$ that occur in $\boldsymbol{q}^\mathfrak{t}_i$.

The two cases are depicted below:



Note that $\mathfrak{t}_\mathsf{r} \neq \emptyset$ ensures that part of the query without $\boldsymbol{q}_\mathfrak{t}$ is mapped onto individual constants.

The NDL program $\Pi^{\text{TW}}_{\boldsymbol{Q}}$ 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 $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$, if it has no existentially quantified variables, then we include the clause $G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$. Otherwise, we include the clause

$$G_{\boldsymbol{q}}(\boldsymbol{x}) \quad\leftarrow \bigwedge_{S(\boldsymbol{z})\in\boldsymbol{q},\ \boldsymbol{z}\subseteq\{z_{\boldsymbol{q}}\}} S(\boldsymbol{z}) \quad\wedge \bigwedge_{1\leq i\leq n} G_{\boldsymbol{q}_i}(\boldsymbol{x}_i),$$

where $\boldsymbol{q}_1(\boldsymbol{x}_1), \dots, \boldsymbol{q}_n(\boldsymbol{x}_n)$ are the subqueries induced by the neighbours of $z_{\boldsymbol{q}}$ in $\mathcal{G}$, and, for each tree witness $\mathfrak{t}$ for $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathfrak{t}_\mathsf{r} \neq \emptyset$ and $z_{\boldsymbol{q}} \in \mathfrak{t}_\mathsf{i}$ and for every ϱ generating $\mathfrak{t}$, the following clause

$$G_{\boldsymbol{q}}(\boldsymbol{x}) \quad\leftarrow A_\varrho(z_0) \;\wedge \bigwedge_{z\in\mathfrak{t}_\mathsf{r}\setminus\{z_0\}} (z = z_0) \;\wedge \bigwedge_{1\leq i\leq k} G_{\boldsymbol{q}^\mathfrak{t}_i}(\boldsymbol{x}^\mathfrak{t}_i),$$

where z_0 is any variable in $\mathfrak{t}_\mathsf{r}$ and $\boldsymbol{q}^\mathfrak{t}_1, \dots, \boldsymbol{q}^\mathfrak{t}_k$ are the connected components of $\boldsymbol{q}$ without $\boldsymbol{q}_\mathfrak{t}$. Finally, if $\boldsymbol{q}_0$ is Boolean, then we include clauses $G_{\boldsymbol{q}_0} \leftarrow A(x)$ for all unary predicates A such that $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}_0$.

The program $\Pi^{\text{TW}}_{\boldsymbol{Q}}$ is inspired by a similar construction from [31]. By adapting the proof, we can show that $(\Pi^{\text{TW}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}(\boldsymbol{x}_0))$ is indeed a rewriting; see Appendix A.5.

Now fix $\boldsymbol{\ell} > 1$ and consider $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ from the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$. The size of $\Pi^{\text{TW}}_{\boldsymbol{Q}}$ is polynomially bounded in $|\boldsymbol{Q}|$ since $\boldsymbol{q}$ has $O(|\boldsymbol{q}|^{\boldsymbol{\ell}})$ tree witnesses and tree-shaped subqueries. It is readily seen that the function ν defined by setting $\nu(G_{\boldsymbol{q}}) = |\boldsymbol{q}|$ is a weight function for $(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0})$ with $\nu(G_{\boldsymbol{q}_0}) \leq |\boldsymbol{Q}|$. Moreover, by Lemma 14, $\mathsf{d}(\Pi, G_{\boldsymbol{q}}) \leq \log \nu(G_{\boldsymbol{q}}) + 1$; and clearly, $\mathsf{w}(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}) \leq \boldsymbol{\ell} + 1$. By Corollary 7, the obtained NDL-rewritings can be evaluated in LOGCFL. 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 14, and in LOGCFL to decide whether $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}_0$ [5] or whether a (logspace) representation of a possible tree witness is indeed a tree witness. This allows us to show that $(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0})$ can be generated by an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer. It also follows that answering OMQs $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with a tree-shaped CQ with $\boldsymbol{\ell}$ leaves over any data instance $\mathcal{A}$ can be done in time

$$poly(|\mathcal{T}|, |\boldsymbol{q}|^{\boldsymbol{\ell}}, |\mathcal{A}|^{\boldsymbol{\ell}}). \qquad (6)$$

4. PARAMETERISED COMPLEXITY

The upper bounds (4) and (6) for the time required to evaluate NDL-rewritings of OMQs from $\mathsf{OMQ}(\boldsymbol{d}, 1, \infty)$ and $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ contain $\boldsymbol{d}$ and $\boldsymbol{\ell}$ in the exponent of $|\mathcal{T}|$ and $|\boldsymbol{q}|$. Moreover, if we allow $\boldsymbol{d}$ and $\boldsymbol{\ell}$ to grow while keeping CQs tree-shaped, the combined complexity of OMQ answering will jump to NP; see Fig. 1(a). In this section, we regard $\boldsymbol{d}$ and $\boldsymbol{\ell}$ as parameters and show that answering tree-shaped OMQs is not fixed-parameter tractable.

4.1 Ontology Depth

Consider the following problem *pDepth*-TreeOMQ:

Instance: an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ with $\mathcal{T}$ of finite depth and tree-shaped Boolean CQ $\boldsymbol{q}$.
Parameter: the depth of $\mathcal{T}$.
Problem: decide whether $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}$.

Theorem 15. *pDepth-TreeOMQ is $W[2]$-hard.*

Proof. The proof is by reduction of the problem p-HittingSet, which is known to be $W[2]$-complete [20]:

Instance: a hypergraph $H = (V, E)$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: decide whether there is $A \subseteq V$ such that $|A| = k$ and $e \cap A \neq \emptyset$, for every $e \in E$.

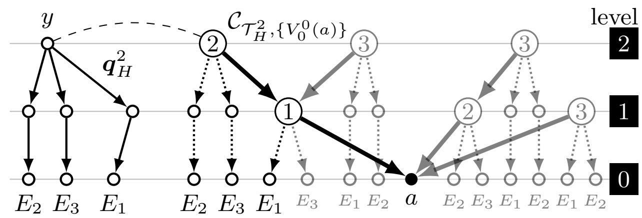
(Such a set A of vertices is called a *hitting set of size k*.) Suppose that $H = (V, E)$ is a hypergraph with vertices $V = \{v_1, \dots, v_n\}$ and hyperedges $E = \{e_1, \dots, e_m\}$. Let $\mathcal{T}^k_H$ be the (normal form of an) ontology with the following axioms, for $1 \leq l \leq k$:

$$V^{l-1}_i(x) \to \exists z \, \big(P(z, x) \land V^l_{i'}(z)\big), \qquad \text{for } 0 \leq i < i' \leq n,$$
$$V^l_i(x) \to E^l_j(x), \qquad \text{for } v_i \in e_j,\ e_j \in E,$$
$$E^l_j(x) \to \exists z \, \big(P(x, z) \land E^{l-1}_j(z)\big), \quad \text{for } 1 \leq j \leq m.$$

Let $\boldsymbol{q}^k_H$ be a tree-shaped Boolean CQ with the following atoms, for $1 \leq j \leq m$:

$$P(y, z^{k-1}_j), \quad P(z^l_j, z^{l-1}_j) \text{ for } 1 \leq l < k, \quad \text{and } E^0_j(z^0_j).$$

The first axiom of $\mathcal{T}^k_H$ generates a tree of depth k, with branching ranging from n to 1, such that the points w of level k are labelled with subsets $X \subseteq V$ of size k that are read off the path from the root to w. The CQ $\boldsymbol{q}^k_H$ is a star with rays corresponding to the hyperedges of H. The second and third axioms generate 'pendants' ensuring that, for any hyperedge e, the central point of the CQ can be mapped to a point with a label X iff X and e have a common vertex. The canonical model of $(\mathcal{T}^2_H, \{V^0_0(a)\})$ and the CQ $\boldsymbol{q}^2_H$, for $H = (V, \{e_1, e_2, e_3\})$ with $V = \{1, 2, 3\}$, $e_1 = \{1, 3\}$, $e_2 = \{2, 3\}$ and $e_3 = \{1, 2\}$, is shown below:



Points (i) at level l belong to V^l_i. In Appendix B.1 we prove that $\mathcal{T}^k_H, \{V^0_0(a)\} \models \boldsymbol{q}^k_H$ iff H has a hitting set of size k. In the example above, $\{1, 2\}$ is a hitting set of size 2, which corresponds to the homomorphism from $\boldsymbol{q}^2_H$ into the part of $\mathcal{C}_{\mathcal{T}^2_H, \{V^0_0(a)\}}$ shown in black. ❑

By Theorem 9, OMQs $(\mathcal{T}, \boldsymbol{q})$ from $\mathsf{OMQ}(\boldsymbol{d}, 1, \infty)$ can be answered (via NDL-rewriting) over a data instance $\mathcal{A}$ in time $poly(|\mathcal{T}|^{\boldsymbol{d}}, |\boldsymbol{q}|, |\mathcal{A}|)$. Theorem 15 shows that no algorithm can do this in time $f(\boldsymbol{d}) \cdot poly(|\mathcal{T}|, |\boldsymbol{q}|, |\mathcal{A}|)$, for any computable function f, unless $W[2] = \mathrm{FPT}$.

4.2 Number of Leaves

Next we consider the problem *pLeaves*-TreeOMQ:

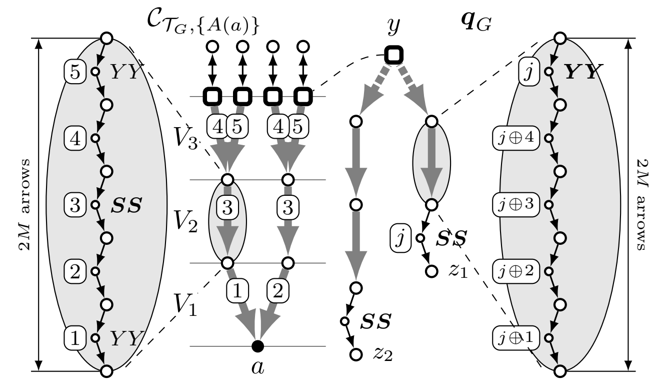
Instance: an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ with $\mathcal{T}$ of finite depth and tree-shaped Boolean CQ $\boldsymbol{q}$.
Parameter: the number of leaves in $\boldsymbol{q}$.
Problem: decide whether $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}$.

Theorem 16. *pLeaves-TreeOMQ is $W[1]$-hard.*

Proof. The proof is by reduction of the following $W[1]$-complete PartitionedClique problem [19]:

Instance: a graph $G = (V, E)$ whose vertices are partitioned into p sets $V_1, \dots, V_p$.
Parameter: p, the number of partitions.
Problem: decide whether G has a clique of size p containing one vertex from each V_i.

Consider a graph $G = (V, E)$ with $V = \{v_1, \dots, v_M\}$ partitioned into $V_1, \dots, V_p$. The ontology $\mathcal{T}_G$ will create a tree rooted at $A(a)$ whose every branch corresponds to selecting one vertex from each V_i. Each branch has length $(p \cdot 2M) + 1$ and consists of p 'blocks' of length $2M$, plus an extra edge at the end (used for padding). Each block corresponds to an enumeration of V, with positions $2j$ and $2j + 1$ being associated with v_j. In the ith block of a branch, we will select a vertex v_{j_i} from V_i by marking the positions $2j_i$ and $2j_i + 1$ with the binary predicate S; we also mark the positions of the neighbours of v_{j_i} in G with the predicate Y. We use the unary predicate B to mark the end of the pth block (square nodes in the picture below). The left side of the picture illustrates the construction for $p = 3$, where $V_1 = \{v_1, v_2\}$, $V_2 = \{v_3\}$, $V_3 = \{v_4, v_5\}$, and $E = \{\{v_1, v_3\}, \{v_3, v_5\}\}$.



Since vertices are enumerated in the same order in every block, to check whether the selected vertex v_{j_i} for V_i is a neighbour of the vertices selected from $V_{i+1}, \dots, V_p$, it suffices to check that positions $2j_i$ and $2j_i + 1$ in blocks $i+1, \dots, p$ are marked YY. Moreover, the distance between the positions of a vertex in consecutive blocks is always $2M - 2$. The idea is thus to construct a CQ $\boldsymbol{q}_G$ (right side of the picture) which, starting from a variable labelled B (mapped to the end of a pth block), splits into $p-1$ branches, with the ith branch checking for a sequence of i evenly-spaced YY markers leading to an SS marker. The distance from the end of the pth block (marked B) to the positions $2j_i$ and $2j_i+1$ in the pth block (where the first YY should occur) depends on the choice of v_{j_i}. We thus add an outgoing edge at the end of the pth block, which can be navigated in both directions, to be able to 'consume' any *even number* of query atoms preceding the first YY.

The Boolean CQ $\boldsymbol{q}_G$ looks as follows (for readability, we use atoms with star-free regular expressions):

$$B(y) \wedge \bigwedge_{1 \le i < p} \big(U^{2M-2} \cdot (YY \cdot U^{2M-2})^i \cdot SS\big)(y, z_i),$$

and the ontology $\mathcal{T}_G$ contains the following axioms:

$$\begin{array}{ll}
A(x) \to \exists y\, L^1_j(x,y), & \text{for } v_j \in V_1,\\
\exists z\, L^k_j(z,x) \to \exists y\, L^{k+1}_j(x,y), & \text{for } 1 \le k < 2M,\ v_j \in V,\\
\exists z\, L^{2M}_j(z,x) \to \exists y\, L^1_{j'}(x,y), & \text{for } v_j \in V_i,\ v_{j'} \in V_{i+1},\\
L^k_j(x,y) \to S(y,x), & \text{for } k \in \{2j, 2j+1\},\\
L^k_j(x,y) \to Y(y,x), & \text{for } \{v_j, v_{j'}\} \in E\\
& \quad \text{and } k \in \{2j', 2j'+1\},\\
L^k_j(x,y) \to U(y,x), & \text{for } 1 \le k \le 2M,\ v_j \in V,\\
\exists z\, L^{2M}_j(z,x) \to B(x), & \text{for } v_j \in V_p,\\
B(x) \to \exists y\, \big(U(x,y) \wedge U(y,x)\big). &
\end{array}$$

We prove in the appendix that $\mathcal{T}_G, \{A(a)\} \models \boldsymbol{q}_G$ iff G has a clique containing one vertex from each set V_i. ❑

By (6), OMQs $(\mathcal{T}, \boldsymbol{q})$ from $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ can be answered (via NDL-rewriting) over a data instance $\mathcal{A}$ in time $\mathit{poly}(|\mathcal{T}|, |\boldsymbol{q}|^{\boldsymbol{\ell}}, |\mathcal{A}|^{\boldsymbol{\ell}})$. Theorem 16 shows that no algorithm can do this in time $f(\boldsymbol{\ell}) \cdot \mathit{poly}(|\mathcal{T}|, |\boldsymbol{q}|, |\mathcal{A}|)$, for any computable function f, unless $W[1] = \mathrm{FPT}$.

One may consider various other types of parameters that can hopefully reduce the complexity of OMQ answering. Obvious candidates are the size of ontology, the size of ontology signature or the number of role inclusions in ontologies. (Indeed, it is shown in [6] that in the absence of role inclusions, tree-shaped OMQ answering is tractable.) Unfortunately, bounding any of these parameters does not make OMQ answering easier, as we establish in Section 5 that already one *fixed* ontology makes the problem NP-hard for tree-shaped CQs and LOGCFL-hard for linear ones.

5. OMQS WITH A FIXED ONTOLOGY

In a typical OBDA scenario [30], users are provided with an ontology in a familiar signature (developed by a domain expert) with which they formulate their queries. Thus, it is of interest to identify the complexity of answering tree-shaped OMQs $(\mathcal{T}, \boldsymbol{q})$ with a fixed $\mathcal{T}$ of infinite depth (see Fig. 1). Surprisingly, we show that the problem is NP-hard even when both $\mathcal{T}$ and $\mathcal{A}$ are fixed (in the database setting, answering tree-shaped CQs is in LOGCFL for combined complexity).

THEOREM 17. *There is an ontology $\mathcal{T}_\dagger$ such that answering OMQs of the form $(\mathcal{T}_\dagger, \boldsymbol{q})$ with Boolean tree-shaped CQs $\boldsymbol{q}$ is NP-hard for query complexity.*

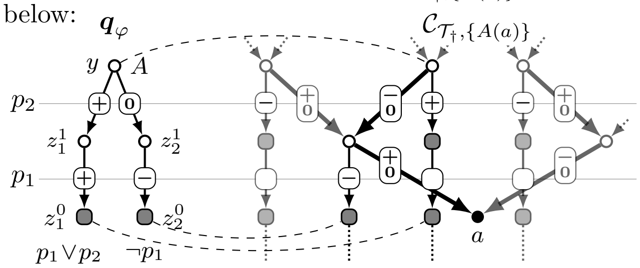
PROOF. The proof is by reduction of SAT. Given a CNF φ with variables $p_1, \dots, p_k$ and clauses $\chi_1, \dots, \chi_m$, take a Boolean CQ $\boldsymbol{q}_\varphi$ with $A(y)$ and, for $1 \le j \le m$, the following atoms with $z^k_j = y$:

$$\begin{array}{ll}
P_+(z^l_j, z^{l-1}_j), & \text{if } p_l \text{ occurs in } \chi_j \text{ positively},\\
P_-(z^l_j, z^{l-1}_j), & \text{if } p_l \text{ occurs in } \chi_j \text{ negatively},\\
P_0(z^l_j, z^{l-1}_j), & \text{if } p_l \text{ does not occur in } \chi_j,\\
B_0(z^0_j). &
\end{array}$$

Thus, $\boldsymbol{q}_\varphi$ is a star with centre $A(y)$ and m rays encoding the χ_j by the binary predicates P_+, P_- and P_0. Let $\mathcal{T}_\dagger$ be an ontology with the axioms

$$\begin{array}{l}
A(x) \to \exists y\, \big(P_+(y,x) \wedge P_0(y,x) \wedge B_-(y) \wedge A(y)\big),\\
\quad B_-(y) \to \exists x'\, \big(P_-(y,x') \wedge B_0(x')\big),\\
A(x) \to \exists y\, \big(P_-(y,x) \wedge P_0(y,x) \wedge B_+(y) \wedge A(y)\big),\\
\quad B_+(y) \to \exists x'\, \big(P_+(y,x') \wedge B_0(x')\big),\\
B_0(x) \to \exists y\, \big(P_+(x,y) \wedge P_-(x,y) \wedge P_0(x,y) \wedge B_0(y)\big).
\end{array}$$

Intuitively, $(\mathcal{T}_\dagger, \{A(a)\})$ generates an infinite binary tree whose nodes of depth n represent all 2^n truth assignments to n propositional variables. The CQ $\boldsymbol{q}_\varphi$ can only be mapped along a branch of this tree towards its root a, with the image of y, the centre of the star, giving a satisfying assignment for φ. Each non-root node of the tree also starts an infinite 'sink' branch of B_0-nodes, where the remainder of the ray for χ_j can be mapped as soon as one of its literals is satisfied. We show in Appendix C.1 that $\mathcal{T}_\dagger, \{A(a)\} \models \boldsymbol{q}_\varphi$ iff φ is satisfiable. To illustrate, the CQ $\boldsymbol{q}_\varphi$ for $\varphi = (p_1 \lor p_2) \land \neg p_1$ and a fragment of the canonical model $\mathcal{C}_{\mathcal{T}_\dagger,\{A(a)\}}$ are shown below:



Here, ● are the points in B_0 and the labels on arrows indicate the subscripts of the binary predicates P (the empty label means all three: $+$, $-$ and 0); predicates A, B_+, B_- are not shown in $\mathcal{C}_{\mathcal{T}_\dagger,\{A(a)\}}$. ❑

The proof above uses OMQs $\boldsymbol{Q}_\varphi = (\mathcal{T}_\dagger, \boldsymbol{q}_\varphi)$ over a data instance with a single individual constant. Thus:

Corollary 18. *No polynomial-time algorithm can construct FO- or NDL-rewritings for the OMQs $\boldsymbol{Q}_\varphi$ unless* $\mathsf{P} = \mathsf{NP}$.

Proof. Indeed, if a polynomial-time algorithm could find a rewriting $\boldsymbol{q}'_\varphi$ of $\boldsymbol{Q}_\varphi$, then we would be able to check whether φ is satisfiable in polynomial time by evaluating $\boldsymbol{q}'_\varphi$ over the data instance $\{A(a)\}$. ❑

Curiously enough, Corollary 18 can be complemented with the following theorem:

Theorem 19. *The $\boldsymbol{Q}_\varphi$ have polynomial FO-rewritings.*

Proof. Define $\boldsymbol{q}'_\varphi$ as the FO-sentence

$$\forall xy\,\big((x = y) \land A(x) \land \varphi^*\big) \;\lor\; \exists xy\,\big((x \neq y) \land \boldsymbol{q}^*_\varphi(x, y)\big),$$

where φ^* is $\top$ if φ is satisfiable and $\bot$ otherwise, and $\boldsymbol{q}^*_\varphi(x, y)$ is the polynomial-size FO-rewriting of $\boldsymbol{Q}_\varphi$ over data with *at least 2* constants [23, Corollary 14]. Recall that the proof of Theorem 17 shows that, if $\mathcal{A}$ has a single constant, a, and there is a homomorphism from $\boldsymbol{q}_\varphi$ to $\mathcal{C}_{\mathcal{T}_\dagger,\mathcal{A}}$, then $A(a) \in \mathcal{A}$ and φ is satisfiable. Thus, the first disjunct of $\boldsymbol{q}'_\varphi$ is an FO-rewriting of $\boldsymbol{Q}_\varphi$ over data instances with a single constant; the case of at least 2 constants follows from [23, Corollary 14]. ❑

Whether the OMQs $\boldsymbol{Q}_\varphi$ have a polynomial-size PE- or NDL-rewritings remains open. We have only managed to construct a modification $\bar{\boldsymbol{q}}_\varphi(x)$ of $\boldsymbol{q}_\varphi$ with the following interesting properties (details are given in Appendix C.2). Let $\mathfrak{T}$ be the class of data instances representing finite binary trees with root a whose edges are labelled with P_+ and P_-, and some of whose leaves are labelled with B_0. Let $\mathcal{QL}$ be any query language such that, for every $\mathcal{QL}$-query $\Phi(x)$ and every $\mathcal{A} \in \mathfrak{T}$, the answer to $\Phi(a)$ over $\mathcal{A}$ can be computed in time polynomial in $|\Phi|$ and $|\mathcal{A}|$. Typical examples of $\mathcal{QL}$ are modal-like languages such as certain fragments of XPath [35] or description logic instance queries [4].

Theorem 20. *The OMQs $(\mathcal{T}_\dagger, \bar{\boldsymbol{q}}_\varphi(x))$ do not have polynomial-size rewritings in $\mathcal{QL}$ unless* $\mathsf{NP} \subseteq \mathsf{P/poly}$.

To our surprise, Theorem 20 is not applicable to PE.[3]

Theorem 21. *Evaluating PE-queries over trees in $\mathfrak{T}$ is* NP*-hard.*

Finally, we consider bounded-leaf CQs (whose evaluation is NL-complete in the database setting) with fixed ontology and data.

Theorem 22. *There is an ontology $\mathcal{T}_\ddagger$ such that answering OMQs of the form $(\mathcal{T}_\ddagger, \boldsymbol{q})$ with Boolean linear CQs $\boldsymbol{q}$ is* LOGCFL*-hard for query complexity.*

The proof is by reduction of the recognition problem for the hardest LOGCFL language $\mathcal{L}$ [26, 52]. We construct an ontology $\mathcal{T}_\ddagger$ and a logspace transducer that converts the words w in the alphabet of $\mathcal{L}$ to linear CQs $\boldsymbol{q}_w$ such that $w \in \mathcal{L}$ iff $\mathcal{T}_\ddagger, \{A(a)\} \models \boldsymbol{q}_w$.

6. EXPERIMENTS & CONCLUSIONS

The main positive result of this paper is the development of theoretically optimal NDL-rewritings for three classes $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$, $\mathsf{OMQ}(\boldsymbol{d}, 1, \boldsymbol{\ell})$, $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ of OMQs. It was known that answering such OMQs is tractable, but the proofs employed elaborate algorithms tailored for each of the three cases. We have shown that the optimal complexity can be achieved *via NDL-rewriting*, thus reducing OMQ answering to standard query evaluation. This result is practically relevant as many user queries are tree-shaped (see, e.g., [45] for evidence in the RDF setting), and indeed, recent tools for query formulation over ontologies (like [51]) produce tree-shaped CQs. Moreover, the majority of important real-world *OWL 2* ontologies are of finite depth; see [15] for statistics. In the context of OBDA, *OWL 2 QL* ontologies are often built starting from the database schemas (bootstrapping [28]), which typically do not contain cycles such as 'every manager is managed by a manager.' For example, the NPD FactPages ontology,[4] designed to facilitate querying the datasets of the Norwegian Petroleum Directorate, is of depth 5.

The starting point of our research was the observation that standard query rewriting systems tend to produce suboptimal rewritings of the OMQs in these three classes. This is obviously so for UCQ-rewriters [46, 43, 13, 25, 40, 36]. However, this is also true of more elaborate PE-rewriters (which use disjunctions inside conjunctions) [47, 54] whose rewritings in theory can be

[3] This result might be known but we could not find it in the literature, and so provide a proof in Appendix C.3.
[4] http://sws.ifi.uio.no/project/npd-v2/

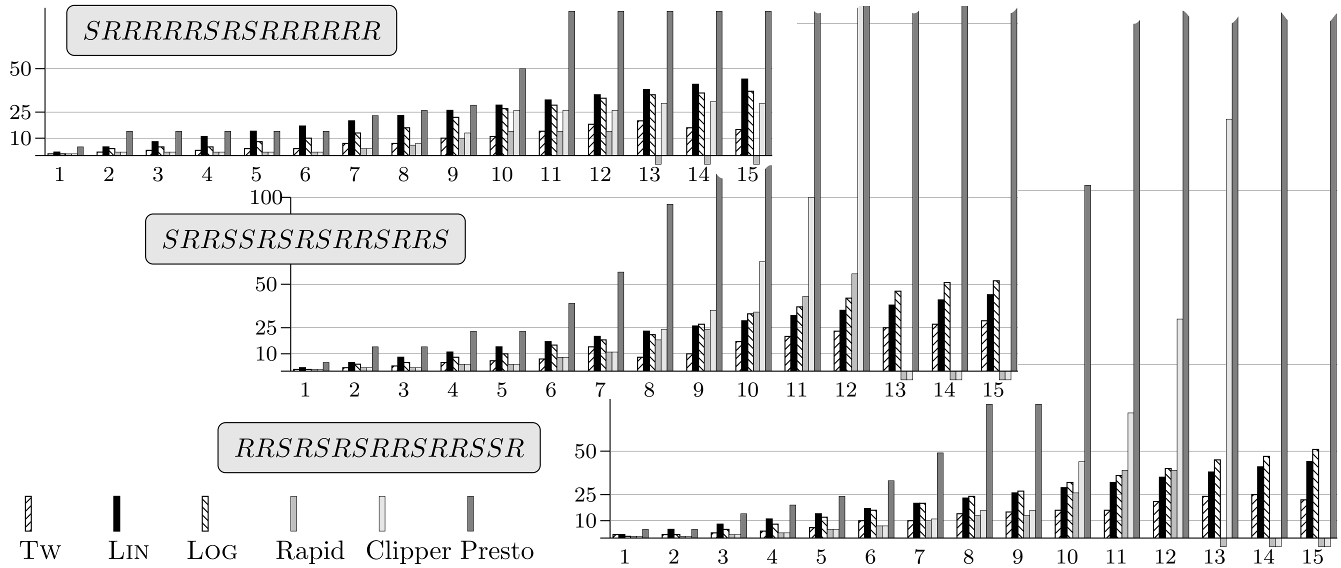


Figure 2: The size of NDL-rewritings produced by different algorithms.

of superpolynomial size; see Fig. 1(b). Surprisingly, even NDL-rewriters such as Clipper [18], Presto [49] and Rapid [13] do not fare much better in practice. To illustrate, we generated three sequences of OMQs in the class $\mathsf{OMQ}(1,1,2)$ (lying in the intersection of $\mathsf{OMQ}(\boldsymbol{d},\boldsymbol{t},\infty)$, $\mathsf{OMQ}(\boldsymbol{d},1,\boldsymbol{\ell})$ and $\mathsf{OMQ}(\infty,1,\boldsymbol{\ell})$) with the ontology from Example 11 and linear CQs of up to 15 atoms as in Example 8 (which are associated with words from $\{R,S\}^*$). By Fig. 1(a), answering these OMQs can be done in NL. The barcharts in Fig. 2 show the number of clauses in their NDL-rewritings produced by Clipper, Presto and Rapid, as well as by our algorithms Lin, Log and Tw from Sections 3.2–3.4, respectively. The first three NDL-rewritings display a clear exponential growth, with Clipper and Rapid failing to produce rewritings for longer CQs. In contrast, our rewritings grow linearly in accord with theory.

We evaluated the rewritings over a few randomly generated data instances using off-the-shelf datalog engine RDFox [42]. The experiments (details are in the appendix) show that our rewritings are usually executed faster than those produced by Clipper, Presto and Rapid.

The version of RDFox we used did not seem to take advantage of the structure of the $\mathsf{NL}/\mathsf{LOGCFL}$ rewritings by simply materialising all the predicates without using magic sets or optimising programs before execution. It would be interesting to see whether the nonrecursiveness and parallelisability of our rewritings can be utilised to produce efficient execution plans. One could also investigate whether our rewritings can be efficiently implemented using views in standard DBMSs.

Our rewriting algorithms are based on the same idea: pick a point splitting the given CQ into sub-CQs, rewrite the sub-CQs recursively, and then formulate rules that combine the resulting rewritings. The difference between the algorithms is in the choice of the splitting points, which determines the execution plans for OMQs and has a big impact on their performance. The experiments show that none of the three splitting strategies systematically outperforms the others. This suggests that execution times may be dramatically improved by employing an 'adaptable' splitting strategy that would work similarly to query execution planners in DBMSs and use statistical information about the relational tables to generate efficient NDL programs. Integrity constraints should also be exploited to optimise rewritings.

Having observed that (i) the ontology depth and (ii) the number of leaves in tree-shaped CQs occur in the exponent of our upper bounds for the complexity of OMQ answering algorithms, we regarded (i) and (ii) as parameters and investigated the parameterised complexity of the OMQ answering problem. We proved that the problem is $W[2]$-hard in the former case and $W[1]$-hard in the latter (it remains open whether these lower bounds are tight). Furthermore, we established that answering OMQs with a fixed ontology (of infinite depth) is NP-complete for tree-shaped CQs and LOGCFL-complete for linear CQs, which dashed hopes of taming intractability by restricting the ontology size, signature, etc. One remaining open problem is whether answering OMQs with a fixed ontology and tree-shaped CQs is fixed-parameter tractable if the number of leaves is regarded as the parameter.

A more general avenue for future research is to extend the study of succinctness and optimality of rewritings to suitable ontology languages with predicates of higher-arity, such as linear and sticky tgds.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[3] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[5] M. Bienvenu, S. Kikot, and V. V. Podolskii. 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*, pages 317–328. IEEE Computer Society, 2015.

[6] M. Bienvenu, M. Ortiz, M. Simkus, and G. Xiao. Tractable queries for lightweight description logics. In *Proc. of the 23nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*, pages 768–774. IJCAI/AAAI, 2013.

[7] M. Calautti, G. Gottlob, and A. Pieris. Chase termination for guarded existential rules. In *Proc. of the 34th ACM Symposium on Principles of Database Systems, PODS 2015*, pages 91–103, 2015.

[8] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics*, 14:57–83, 2012.

[9] A. Calì, G. Gottlob, and A. Pieris. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012.

[10] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: the *DL-Lite* family. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[12] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

[13] A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting for OWL 2 QL. In *Proc. of CADE-23*, volume 6803 of *LNCS*, pages 192–206. Springer, 2011.

[14] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, 1971.

[15] B. Cuenca Grau, I. Horrocks, M. Krötzsch, C. Kupke, D. Magka, B. Motik, and Z. Wang. Acyclicity notions for existential rules and their application to query answering in ontologies. *Journal of Artificial Intelligence Research (JAIR)*, 47:741–808, 2013.

[16] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[17] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[18] T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI 2012)*, pages 726–733. AAAI, 2012.

[19] M. R. Fellows, D. Hermelin, F. A. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009.

[20] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[21] M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jiménez-Ruiz, D. Lanti, M. Rezk, G. Xiao, Ö. Özçep, and R. Rosati. Optique: Zooming in on big data. *IEEE Computer*, 48(3):60–67, 2015.

[22] T. Gogacz and J. Marcinkowski. All-instances termination of chase is undecidable. In *Proc. of the 41st Int. Colloquium Automata, Languages, and Programming (ICALP 2014), Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2014.

[23] G. Gottlob, S. Kikot, R. Kontchakov, V. V. Podolskii, T. Schwentick, and M. Zakharyaschev. The price of query rewriting in ontology-based data access. *Artificial Intelligence*, 213:42–59, 2014.

[24] G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL certificates. In *Proc. of the 26th Int. Colloquium on Automata, Languages and Programming (ICALP-99)*, volume 1644 of *Lecture Notes in Computer Science*, pages 361–371. Springer, 1999.

[25] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pages 2–13. IEEE Computer Society, 2011.

[26] S. A. Greibach. The hardest context-free language. *SIAM J. Comput.*, 2(4):304–310, 1973.

[27] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the*

Institute of Radio Engineers, 40(9):1098–1101, 1952.

[28] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skjæveland, E. Thorstensen, and J. Mora. Bootox: Bootstrapping OWL 2 ontologies and R2RML mappings from relational databases. In *Proc. of the ISWC 2015 Posters & Demonstrations Track at the 14th Int. Semantic Web Conf. (ISWC-2015)*, volume 1486 of *CEUR Workshop Proceedings*. CEUR-WS, 2015.

[29] M. Kaminski, Y. Nenov, and B. Cuenca Grau. Datalog rewritability of Disjunctive Datalog programs and non-Horn ontologies. *Artificial Intelligence*, 236:90–118, 2016.

[30] E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, and I. Horrocks. Ontology based access to exploration data at Statoil. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC 2015), Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 93–112. Springer, 2015.

[31] S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev. 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)*, pages 57:1–57:10. ACM, 2014.

[32] S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev. Exponential lower bounds and separation for query rewriting. In *Proc. of the 39th Int. Colloquium on Automata, Languages and Programming (ICALP 2012)*, volume 7392 of *Lecture Notes in Computer Science*, pages 263–274. Springer, 2012.

[33] S. Kikot, R. Kontchakov, and M. Zakharyaschev. On (in)tractability of OBDA with OWL 2 QL. In *Proc. of the 24th Int. Workshop on Description Logics (DL 2011)*, volume 745, pages 224–234. CEUR-WS, 2011.

[34] S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 275–285. AAAI, 2012.

[35] C. Koch. Processing queries on tree-structured data efficiently. In *Proc. of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2006)*, pages 213–224. ACM, 2006.

[36] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. Sound, complete and minimal UCQ-rewriting for existential rules. *Semantic Web*, 6(5):451–475, 2015.

[37] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in DL-Lite. In *Proc. of the 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 247–257. AAAI Press, 2010.

[38] R. Kontchakov, M. Rezk, M. Rodriguez-Muro, G. Xiao, and M. Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 552–567. Springer, 2014.

[39] M. Lenzerini. Ontology-based data management. *ACM SIGMOD Blog*, May 2013.

[40] J. Mora, R. Rosati, and Ó. Corcho. Kyrie2: query rewriting under extensional constraints in ELHIO. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014)*, volume 8796 of *Lecture Notes in Computer Science*, pages 568–583. Springer, 2014.

[41] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. *OWL 2 Web Ontology Language Profiles*. W3C Recommendation, 2012. Available at http://www.w3.org/TR/owl2-profiles/.

[42] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee. RDFox: A highly-scalable RDF store. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC 2015), Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2015.

[43] H. Pérez-Urbina, B. Motik, and I. Horrocks. A comparison of query rewriting techniques for DL-Lite. In *Proc. of the 22nd Int. Workshop on Description Logics (DL 2009)*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS, 2009.

[44] H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. Evaluation of query rewriting approaches for OWL 2. In *Proc. of SSWS+HPCSW 2012*, volume 943 of *CEUR Workshop Proceedings*. CEUR-WS, 2012.

[45] F. Picalausa and S. Vansummeren. What are real SPARQL queries like? In *Proc. of the Int. Workshop on Semantic Web Information Management (SWIM)*. ACM, 2011.

[46] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *Journal on Data Semantics*, X:133–173, 2008.

[47] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of the 12th Int. Semantic Web Conf. (ISWC 2013)*, volume 8218 of *Lecture Notes in Computer Science*, pages 558–573. Springer, 2013.

[48] R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proc. of the*

9th Extended Semantic Web Conf. (EWSC 2012), volume 7295 of *Lecture Notes in Computer Science*, pages 360–374. Springer, 2012.

[49] R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. of the 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 290–300. AAAI Press, 2010.

[50] J. F. Sequeda, M. Arenas, and D. P. Miranker. OBDA: query rewriting or materialization? In practice, both! In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 535–551. Springer, 2014.

[51] A. Soylu, M. Giese, E. Jimenez-Ruiz, G. Vega-Gorgojo, and I. Horrocks. Experiencing optiquevqs: A multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society*, 15(1):129–152, 2016.

[52] I. H. Sudborough. A note on tape-bounded complexity classes and linear context-free languages. *Journal of the ACM*, 22(4):499–500, Oct. 1975.

[53] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978.

[54] M. Thomazo. Compact rewritings for existential rules. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*. IJCAI/AAAI, 2013.

[55] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991.

[56] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of the 7th Int. Conf. on Very Large Data Bases (VLDB)*, pages 82–94. IEEE Computer Society, 1981.

APPENDIX

A. PROOFS FOR SECTION 3

A.1 Lemma 3

LEMMA 3. *Fix any* $\mathsf{w} > 0$. *There is an* L^{NL}*-transducer that, for any linear NDL-rewriting* $(\Pi, G(\boldsymbol{x}))$ *of an OMQ* $\boldsymbol{Q}(\boldsymbol{x})$ *over complete data instances with* $\mathsf{w}(\Pi, G) \leq \mathsf{w}$, *computes a linear NDL-rewriting* $(\Pi', G(\boldsymbol{x}))$ *of* $\boldsymbol{Q}(\boldsymbol{x})$ *over arbitrary data instances such that* $\mathsf{w}(\Pi', G) \leq \mathsf{w}{+}1$.

PROOF. Let $(\Pi, G(\boldsymbol{x}))$ be a linear NDL-rewriting of the OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ over complete data instances such that $\mathsf{w}(\Pi, G) \leq \mathsf{w}$. We will replace every clause λ in Π by a set of clauses λ^* defined as follows. Suppose λ is of the form

$$Q(\boldsymbol{z}) \leftarrow I \land EQ \land E_1 \land \ldots \land E_n,$$

where I is the only IDB body atom in λ, EQ contains all equality body atoms, and $E_1, \ldots, E_n$ are the EDB body atoms not involving equality. For every atom E_i, we define a set $\upsilon(E_i)$ of atoms by taking

$$\begin{aligned}\upsilon(E_i) = \{B(z) \mid \mathcal{T} \models B(x) \to A(x)\} \cup{} \\ \{\varrho(y_i, z) \mid \mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)\}, \\ \text{if } E_i = A(z), \\ \upsilon(E_i) = \{\varrho(z, z') \mid \mathcal{T} \models \varrho(x, y) \to P(x, y)\}, \\ \text{if } E_i = P(z, z'),\end{aligned}$$

where y_i is a fresh variable not occurring in λ; we assume $P^-(z, z')$ coincides with $P(z', z)$, for all binary predicates P. Intuitively, $\upsilon(E_i)$ captures all atoms that imply E_i with respect to $\mathcal{T}$. Then λ^* consists of the following clauses:

$$\begin{aligned}&Q_0(\boldsymbol{z}_0) \leftarrow I, \\ &Q_{i+1}(\boldsymbol{z}_i) \leftarrow H_i(\boldsymbol{z}_i) \land E'_i, \quad \text{for } 1 \leq i \leq n \text{ and } E'_i \in \upsilon(E_i), \\ &Q(\boldsymbol{z}) \leftarrow H_{n+1}(\boldsymbol{z}_n) \land EQ,\end{aligned}$$

where $\boldsymbol{z}_i$ is the restriction of $\boldsymbol{z}$ to variables occurring in I if $i = 0$ and in $Q_i(\boldsymbol{z}_i)$ and E'_i except for y_i if $i > 0$ (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 cannot exceed $\mathsf{w}(\Pi, G) + 1$ (the possible increase of 1 is due to the replacement of unary atoms $A(z)$ by binary atoms $\varrho(y_i, z)$).

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 $Q(\boldsymbol{z}) \leftarrow I \land EQ \land E_1 \land \ldots \land E_n$ by the (possibly exponentially larger) set of clauses of the form

$$Q(\boldsymbol{z}) \leftarrow I \land EQ \land E'_1 \land \ldots \land E'_n,$$

for all $E'_i \in \upsilon(E_i)$ 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 $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$, where $\mathcal{A}$ is an arbitrary data instance. Let $\mathcal{A}'$ be the complete data instance obtained from $\mathcal{A}$ by adding the ground atoms:

$$\begin{aligned}P(a,b) \quad &\text{if} \quad \varrho(a,b) \in \mathcal{A} \text{ and } \mathcal{T} \models \varrho(x,y) \to P(x,y); \\ A(a) \quad &\text{if} \quad B(a) \in \mathcal{A} \text{ and } \mathcal{T} \models B(x) \to A(x); \\ A(a) \quad &\text{if} \quad \varrho(a,b) \in \mathcal{A} \text{ and } \mathcal{T} \models \exists y\, \varrho(y,x) \to A(x).\end{aligned}$$

(We write $\varrho(a, b) \in \mathcal{A}$ for $P(a, b) \in \mathcal{A}$ if $\varrho = P$ and for $P(b, a)$ if $\varrho = P^-$.) Clearly, $\mathcal{T}, \mathcal{A}' \models \boldsymbol{q}(\boldsymbol{a})$, so we must have $\Pi, \mathcal{A}' \models G(\boldsymbol{a})$. A simple inductive argument (on the order of derivation of ground atoms) shows that whenever a clause $Q(\boldsymbol{z}) \leftarrow I \land EQ \land E_1 \land \ldots \land E_n$ 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 $Q(\boldsymbol{z}) \leftarrow I \land EQ \land E'_1 \land \ldots \land E'_n$ and a substitution $\boldsymbol{c}'$ extending $\boldsymbol{c}$ (on the fresh variables y_i) that allows us to derive $Q(\boldsymbol{c}'(\boldsymbol{z}))$ using Π''. Indeed,

- if $E_i = A(z)$, then $A(\boldsymbol{c}(z)) \in \mathcal{A}'$, so there must exist either a unary ground atom $B(\boldsymbol{c}(z)) \in \mathcal{A}$ such that $\mathcal{T} \models B(x) \to A(x)$ or a binary ground atom $\varrho(a, \boldsymbol{c}(z)) \in \mathcal{A}$, for some $a \in \mathsf{ind}(\mathcal{A})$, such that $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$; in the latter case, we set $\boldsymbol{c}'(y_i) = a$;
- similarly, if $E_i = P(z, z')$, then there must exist a binary ground atom $\varrho(\boldsymbol{c}(z), \boldsymbol{c}(z')) \in \mathcal{A}$ such that $\mathcal{T} \models \varrho(x, y) \to P(x, y)$.

It then suffices to choose $Q(\boldsymbol{z}) \leftarrow I \land EQ \land E'_1 \land \ldots \land E'_n$ with atoms E'_i whose form match that of the ground atoms in $\mathcal{A}$ corresponding to E_i.

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

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

A.2 Theorem 6

Next, we combine the transformation in Lemma 5 with the established complexity in Lemma 4 to obtain the combined complexity upper bound:

THEOREM 6. *For any* $c \geq 1$ *and* $\mathsf{w} \geq 1$, *evaluation of NDL queries* $(\Pi, G(\boldsymbol{x}))$ *having a weight function* ν *such that* $\mathsf{d}(\Pi, G) + \log \nu(G) \leq c \log |\Pi|$ *and* $\mathsf{w}(\Pi, G) \leq \mathsf{w}$ *is in* LOGCFL *for combined complexity.*

PROOF. By Lemma 5, (Π, G) is equivalent to a skinny (Π', G) such that $|\Pi'| = O(|\Pi|^2)$, $\mathsf{w}(\Pi', G) \leq \mathsf{w}$, and $\mathsf{d}(\Pi', G) \leq \mathsf{d}(\Pi, G) + \log \nu(G)$. By Lemma 4, query evaluation for (Π', G) over $\mathcal{A}$ is done by an NAuxPDA in space $\log |\Pi'| + \mathsf{w}(\Pi', G) \cdot \log |\mathcal{A}| = O(\log |\Pi| + \log |\mathcal{A}|)$ and time $2^{O(\mathsf{d}(\Pi', G))} \leq |\Pi|^{O(1)}$. ❑

A.3 LOG-rewritings

LEMMA 23. *For any complete data instance* $\mathcal{A}$, *any* $D \in \mathfrak{D}$, *any type* $\boldsymbol{w}$ *with* $\mathsf{dom}(\boldsymbol{w}) = \partial D$ *and any tuples* $\boldsymbol{b} \in \mathsf{ind}(\mathcal{A})^{|\partial D|}$ *and* $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}_D|}$, *we have*

$\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}}_D(\boldsymbol{b}, \boldsymbol{a})$ *iff there is a homomorphism* $h\colon \boldsymbol{q}_D \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ *such that*

$$h(x) = \boldsymbol{a}(x), \quad \text{for } x \in \boldsymbol{x}_D, \qquad \text{and} \quad h(z) = \boldsymbol{b}(z)\boldsymbol{w}(z), \quad \text{for } z \in \partial D. \tag{7}$$

PROOF. ($\Rightarrow$) The proof is by induction on $\prec$. For the basis of induction, let D be of size 1. By the definition of $\Pi^{\text{LOG}}_{\boldsymbol{Q}}$, there exists a type $\boldsymbol{s}$ such that $\mathsf{dom}(\boldsymbol{s}) = \lambda(\sigma(D))$ and $\boldsymbol{w}$ agrees with $\boldsymbol{s}$ on ∂D and a respective tuple $\boldsymbol{c} \in \mathsf{ind}(\mathcal{A})^{|\lambda(\sigma(D))|}$ such that $\boldsymbol{c}(z) = \boldsymbol{b}(z)$, for all $z \in \partial D$, and $\boldsymbol{c}(x) = \boldsymbol{a}(x)$, for all $x \in \boldsymbol{x}_D$, and $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{c})$. Then, for any atom $S(\boldsymbol{z}) \in \boldsymbol{q}_D$, we have $\boldsymbol{z} \subseteq \lambda(\sigma(D))$, whence $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models S(h(\boldsymbol{z}))$ as $\boldsymbol{w}$ agrees with $\boldsymbol{s}$ on ∂D.

For the inductive step, suppose that we have $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}}_D(\boldsymbol{b}, \boldsymbol{a})$. By the definition of $\Pi^{\text{LOG}}_{\boldsymbol{Q}}$, 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 a respective tuple $\boldsymbol{c} \in \mathsf{ind}(\mathcal{A})^{|\lambda(\sigma(D))|}$ such that $\boldsymbol{c}(z) = \boldsymbol{b}(z)$, for all $z \in \partial D$, and $\boldsymbol{c}(x) = \boldsymbol{a}(x)$, for all $x \in \boldsymbol{x}_D$, and

$$\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{c}) \wedge \bigwedge\nolimits_{D' \prec D} G^{(\boldsymbol{s}\cup\boldsymbol{w}) \restriction \partial D'}_{D'}(\boldsymbol{b}_{D'}, \boldsymbol{a}_{D'}),$$

where $\boldsymbol{b}_{D'}$ and $\boldsymbol{a}_{D'}$ are the restrictions of $\boldsymbol{b} \cup \boldsymbol{c}$ to $\partial D'$ and of $\boldsymbol{a}$ to $\boldsymbol{x}_{D'}$, respectively. By the induction hypothesis, for any $D' \prec D$, there is a homomorphism $h_{D'}\colon \boldsymbol{q}_{D'} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that (7) 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 $z \in V$, 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 $\boldsymbol{w}'(z) = \boldsymbol{w}''(z)$. This implies that

$$h_{D'}(z) = \boldsymbol{c}(z)\boldsymbol{w}'(z) = \boldsymbol{c}(z)\boldsymbol{w}''(z) = h_{D''}(z).$$

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

$$h(z) = \begin{cases} h_{D'}(z) & \text{if } z \in \lambda(t), \\ & \quad \text{for } t \in D' \text{ and } D' \prec D, \\ \boldsymbol{c}(z) \cdot (\boldsymbol{w} \cup \boldsymbol{s})(z), & \text{if } z \in \lambda(\sigma(D)). \end{cases}$$

If follows that h is well defined, h satisfies (7) and that h is a homomorphism from $\boldsymbol{q}_D$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Indeed, take an atom $S(\boldsymbol{z}) \in \boldsymbol{q}_D$. Then either $\boldsymbol{z} \subseteq \lambda(\sigma(D))$, in which case $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models S(h(\boldsymbol{z}))$ since $\boldsymbol{w}$ is compatible with $\sigma(D)$ and $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{c})$, or $S(\boldsymbol{z}) \in \boldsymbol{q}_{D'}$ for some $D' \prec D$, in which case we use the fact that h extends a homomorphism $h_{D'}$.

($\Leftarrow$) The proof is by induction on $\prec$. Fix D and $\boldsymbol{w}$ such that $|\boldsymbol{w}| = |\partial D|$. Take tuples $\boldsymbol{b} \in \mathsf{ind}(\mathcal{A})^{|\partial D|}$ and $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}_D|}$, and a homomorphism $h\colon \boldsymbol{q}_D \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ satisfying (7). Define a type $\boldsymbol{s}$ and a tuple $\boldsymbol{c} \in \mathsf{ind}(\mathcal{A})^{|\lambda(\sigma(D))|}$ by taking, for all $z \in \lambda(\sigma(D))$,

$$\boldsymbol{s}(z) = w \text{ and } \boldsymbol{c}(z) = a, \quad \text{if } h(z) = aw, \text{ for } a \in \mathsf{ind}(\mathcal{A}).$$

By definition, $\mathsf{dom}(\boldsymbol{s}) = \lambda(\sigma(D))$ and, by (7), $\boldsymbol{s}$ and $\boldsymbol{w}$ agree on the common domain. For the inductive step, for each $D' \prec D$, let $h_{D'}$ be the restriction of h to $\boldsymbol{q}_{D'}$ and let $\boldsymbol{b}_{D'}$ and and $\boldsymbol{a}_{D'}$ be the restrictions of $\boldsymbol{b} \cup \boldsymbol{c}$ to $\partial D'$ and of $\boldsymbol{a}$ to $\boldsymbol{x}_{D'}$, respectively. By the inductive hypothesis, $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}'}_{D'}(\boldsymbol{b}_{D'}, \boldsymbol{a}_{D'})$. (This argument is not needed for the basis of induction.) Since h is a homomorphism, we have $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models \mathsf{At}^{\boldsymbol{s}}(\boldsymbol{c})$, whence, $\Pi^{\text{LOG}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}}_D(\boldsymbol{b}, \boldsymbol{a})$. ❑

It follows that answering OMQs $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathcal{T}$ of finite depth $\boldsymbol{d}$ and $\boldsymbol{q}$ of treewidth $\boldsymbol{t}$ over any data instance $\mathcal{A}$ can be done in time

$$poly(|\mathcal{T}|^{\boldsymbol{dt}}, |\boldsymbol{q}|, |\mathcal{A}|^{\boldsymbol{t}}). \tag{4}$$

Indeed, we can evaluate $(\Pi^{\text{LOG}}_{\boldsymbol{Q}}, G^{\boldsymbol{\varepsilon}}_T(\boldsymbol{x}))$ in time polynomial in $|\Pi^{\text{LOG}}_{\boldsymbol{Q}}|$ and $|\mathcal{A}|^{\mathsf{w}(\Pi^{\text{LOG}}_{\boldsymbol{Q}}, G^{\boldsymbol{\varepsilon}}_{T})}$, which are bounded by a polynomial in $|\mathcal{T}|^{2\boldsymbol{d}(\boldsymbol{t}+1)}$, $|\boldsymbol{q}|$ and $|\mathcal{A}|^{2(\boldsymbol{t}+1)}$.

A.4 LIN-rewritings

LEMMA 24. *For any complete data instance* $\mathcal{A}$, *any predicate* $G^{\boldsymbol{w}}_n$, *any* $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}^n|}$ *and* $\boldsymbol{b} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{z}^n_\exists|}$, *we have* $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}}_n(\boldsymbol{b}, \boldsymbol{a})$ *iff there is a homomorphism* $h\colon \boldsymbol{q}_n \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ *such that*

$$h(x) = \boldsymbol{a}(x), \quad \text{for } x \in \boldsymbol{x}^n, \qquad \text{and} \quad h(z) = \boldsymbol{b}(z)\boldsymbol{w}(z), \quad \text{for } z \in \boldsymbol{z}^n_\exists. \tag{8}$$

PROOF. The proof is by induction on n.

For the base case ($n = M$), first suppose that we have $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}}_M(\boldsymbol{b}, \boldsymbol{a})$. The only rule in $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ with head predicate $G^{\boldsymbol{w}}_M$ is $G^{\boldsymbol{w}}_M(\boldsymbol{z}^M_\exists, \boldsymbol{x}^M) \leftarrow \mathsf{At}^{\boldsymbol{w}}(\boldsymbol{z}^M)$ with $\boldsymbol{z}^M = \boldsymbol{z}^M_\exists \uplus \boldsymbol{x}^M$, which is equivalent to

$$G^{\boldsymbol{w}}_M(\boldsymbol{z}^M_\exists, \boldsymbol{x}^M) \leftarrow \bigwedge_{z \in \boldsymbol{z}^M} \Big(\bigwedge_{\substack{A(z) \in \boldsymbol{q} \\ \boldsymbol{w}(z) = \varepsilon}} A(z) \;\wedge \bigwedge_{\substack{P(z,z) \in \boldsymbol{q} \\ \boldsymbol{w}(z) = \varepsilon}} P(z,z) \;\wedge \bigwedge_{\boldsymbol{w}(z) = \varrho w} A_\varrho(z) \Big). \tag{9}$$

So the body of this rule must be satisfied when $\boldsymbol{b}$ and $\boldsymbol{a}$ are substituted for $\boldsymbol{z}^M_\exists$ and $\boldsymbol{x}^M$ respectively. Moreover, by local compatibility of $\boldsymbol{w}$ with $\boldsymbol{z}^M$, we know that $\boldsymbol{w}(x) = \varepsilon$ for every $x \in \boldsymbol{x}^M$. It follows that

- $A(\boldsymbol{a}(x)) \in \mathcal{A}$ for every $A(x) \in \boldsymbol{q}$ such that $x \in \boldsymbol{x}^M$;
- $A(\boldsymbol{b}(z)) \in \mathcal{A}$ for every $A(z) \in \boldsymbol{q}$ such that $z \in \boldsymbol{z}^M_\exists$ and $\boldsymbol{w}(z) = \varepsilon$;
- $P(\boldsymbol{a}(x), \boldsymbol{a}(x)) \in \mathcal{A}$ for every $P(x,x) \in \boldsymbol{q}$ such that $x \in \boldsymbol{x}^M$;
- $P(\boldsymbol{b}(z), \boldsymbol{b}(z)) \in \mathcal{A}$ for every $P(z,z) \in \boldsymbol{q}$ such that $z \in \boldsymbol{z}^M_\exists$ and $\boldsymbol{w}(z) = \varepsilon$;
- $A_\varrho(z) \in \mathcal{A}$ for every $z \in \boldsymbol{z}^M$ with $\boldsymbol{w}(z) = \varrho w$.

Now let h^M be the unique mapping from $\boldsymbol{z}^M$ to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ satisfying (8). First note that h^M is well-defined, since by the last item, if $\boldsymbol{w}(z) = \varrho w$, then we have $A_\varrho(z) \in \mathcal{A}$

and $\varrho w \in \boldsymbol{W}_{\mathcal{T}}$, so $\boldsymbol{b}(z)\varrho w$ belongs to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. To show that h^M is a homomorphism of $\boldsymbol{q}_M$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, first recall that the atoms of $\boldsymbol{q}_M$ are of two types: $A(z)$ or $P(z,z)$, with $z \in \boldsymbol{z}^M$. Take some $A(z) \in \boldsymbol{q}_M$. If $\boldsymbol{w}(z) = \varepsilon$, then we immediately obtain either $A(h^M(z)) = A(\boldsymbol{a}(z)) \in \mathcal{A}$ or $A(h^M(z)) = A(\boldsymbol{b}(z)) \in \mathcal{A}$, depending on whether $z \in \boldsymbol{z}_\exists^M$ or in $\boldsymbol{x}^M$. Otherwise, if $\boldsymbol{w}(z) \neq \varepsilon$, then the local compatibility of $\boldsymbol{w}$ with $\boldsymbol{z}^M$ means that the final letter ϱ in $\boldsymbol{w}(z)$ is such that $\mathcal{T} \models \exists y\, \varrho(y,x) \to A(x)$, hence $h^M(z) = \boldsymbol{b}(z)\boldsymbol{w}(z) \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. Finally, suppose that $P(z,z) \in \boldsymbol{q}$. The local compatibility of $\boldsymbol{w}$ with $\boldsymbol{z}^M$ ensures that either $\boldsymbol{w}(z) = \varepsilon$ or $\mathcal{T} \models P(x,x)$. In the former case, we have either $P(\boldsymbol{a}(z), \boldsymbol{a}(z)) \in \mathcal{A}$ or $P(\boldsymbol{b}(z), \boldsymbol{b}(z)) \in \mathcal{A}$, depending again on whether $z \in \boldsymbol{z}_\exists^M$ or $z \in \boldsymbol{x}^M$. In the latter case, $(h^M(z), h^M(z)) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

For the other direction, $(\Leftarrow)$, of the base case, suppose that the mapping h^M given by (8) defines a homomorphism from $\boldsymbol{q}_M$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. We therefore have:

- $\boldsymbol{a}(x) \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ for every $A(x) \in \boldsymbol{q}$ with $x \in \boldsymbol{x}^M$;
- $\boldsymbol{b}(z)\boldsymbol{w}(z) \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ for every $A(z) \in \boldsymbol{q}$ with $z \in \boldsymbol{z}_\exists^M$;
- $(\boldsymbol{a}(x), \boldsymbol{a}(x)) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ for every $P(x,x) \in \boldsymbol{q}$ such that $x \in \boldsymbol{x}^M$;
- $(\boldsymbol{b}(z), \boldsymbol{b}(z)) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ for every $P(z,z) \in \boldsymbol{q}$ such that $z \in \boldsymbol{z}_\exists^M$;
- $\mathcal{T}, \mathcal{A} \models \exists y\, \varrho(\boldsymbol{b}(z), y)$ for every $z \in \boldsymbol{z}_\exists^M$ with $\boldsymbol{w}(z) = \varrho w$ (for otherwise $\boldsymbol{b}(z)\boldsymbol{w}(z)$ would not belong to the domain of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$).

The first two items, together with completeness of the data instance $\mathcal{A}$, ensure that all atoms in

$$\{A(z) \mid A(z) \in \boldsymbol{q}, z \in \boldsymbol{z}^M, \boldsymbol{w}(z) = \varepsilon\}$$

are present in $\mathcal{A}$ when $\boldsymbol{b}$ and $\boldsymbol{a}$ substituted for $\boldsymbol{z}_\exists^M$ and $\boldsymbol{x}^M$, respectively. The third and fourth items, again together with completeness of $\mathcal{A}$, ensure the presence of the atoms in

$$\{P(z,z) \mid P(z,z) \in \boldsymbol{q}, z \in \boldsymbol{z}^M, \boldsymbol{w}(z) = \varepsilon\}.$$

Finally, the fifth item plus completeness of $\mathcal{A}$ ensure that $\mathcal{A}$ contains all atoms in

$$\{A_\varrho(z) \mid z \in \boldsymbol{z}^M, \boldsymbol{w}(z) = \varrho w\}.$$

It follows that the body of the unique rule for $G_M^{\boldsymbol{w}}$ is satisfied when $\boldsymbol{b}$ and $\boldsymbol{a}$ are substituted for $\boldsymbol{z}_\exists^M$ and $\boldsymbol{x}^M$ respectively, and thus $\Pi_{\boldsymbol{Q}}^{\text{LIN}}, \mathcal{A} \models G_M^{\boldsymbol{w}}(\boldsymbol{b}, \boldsymbol{a})$.

For the induction step, assume that the statement has been shown to hold for all $n \leq k+1 \leq M$, and let us show that it holds when $n = k$. For the first direction, $(\Rightarrow)$, suppose $\Pi_{\boldsymbol{Q}}^{\text{LIN}}, \mathcal{A} \models G_k^{\boldsymbol{w}}(\boldsymbol{b}, \boldsymbol{a})$. It follows that there exists a pair of types $(\boldsymbol{w}, \boldsymbol{s})$ compatible with $(\boldsymbol{z}^k, \boldsymbol{z}^{k+1})$ and an assignment $\boldsymbol{c}$ of individuals from $\mathcal{A}$ to the variables in $\boldsymbol{z}^k \cup \boldsymbol{z}^{k+1}$ such that $\boldsymbol{c}(x) = \boldsymbol{a}(x)$ for all $x \in (\boldsymbol{z}^k \cup \boldsymbol{z}^{k+1}) \cap \boldsymbol{x}$, and $\boldsymbol{c}(z) = \boldsymbol{b}(z)$ for all $z \in \boldsymbol{z}_\exists^k$, and such that every atom in the body of the clause

$$G_k^{\boldsymbol{w}}(\boldsymbol{z}_\exists^k, \boldsymbol{x}^k) \leftarrow \mathsf{At}^{\boldsymbol{w} \cup \boldsymbol{s}}(\boldsymbol{z}^k, \boldsymbol{z}^{k+1}) \wedge G_{k+1}^{\boldsymbol{s}}(\boldsymbol{z}_\exists^{k+1}, \boldsymbol{x}^{k+1})$$

is entailed from $\Pi_{\boldsymbol{Q}}^{\text{LIN}}, \mathcal{A}$ when the individuals in $\boldsymbol{c}$ are substituted for $\boldsymbol{z}^k \cup \boldsymbol{z}^{k+1}$. Recall that $\mathsf{At}^{\boldsymbol{w} \cup \boldsymbol{s}}(\boldsymbol{z}^k, \boldsymbol{z}^{k+1})$ is the conjunction of the following atoms, for $z, z' \in \boldsymbol{z}^k \cup \boldsymbol{z}^{k+1}$:

- $A(z)$, if $A(z) \in \boldsymbol{q}$ and $(\boldsymbol{w} \cup \boldsymbol{s})(z) = \varepsilon$,
- $P(z,z')$, if $P(z,z') \in \boldsymbol{q}$ and $(\boldsymbol{w} \cup \boldsymbol{s})(z) = (\boldsymbol{w} \cup \boldsymbol{s})(z') = \varepsilon$,
- $z = z'$, if $P(z,z') \in \boldsymbol{q}$ and either $(\boldsymbol{w} \cup \boldsymbol{s})(z) \neq \varepsilon$ or $(\boldsymbol{w} \cup \boldsymbol{s})(z') \neq \varepsilon$,
- $A_\varrho(z)$, if $(\boldsymbol{w} \cup \boldsymbol{s})(z)$ is of the form ϱw.

In particular, we have $\Pi_{\boldsymbol{Q}}^{\text{LIN}}, \mathcal{A} \models G_{k+1}^{\boldsymbol{s}}(\boldsymbol{c}(\boldsymbol{z}_\exists^{k+1}), \boldsymbol{c}(\boldsymbol{x}^{k+1}))$. By the induction hypothesis, there exists a homomorphism $h^{k+1} \colon \boldsymbol{q}_{k+1} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h^{k+1}(z) = \boldsymbol{c}(z)\boldsymbol{s}(z)$ for every $z \in \boldsymbol{z}_\exists^{k+1} \cup \boldsymbol{x}^{k+1}$. Define a mapping h^k from $\mathsf{var}(\boldsymbol{q}_k)$ to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ by setting $h^k(z) = h^{k+1}(z)$ for every variable $z \in \mathsf{var}(\boldsymbol{q}_{k+1})$, setting $h^k(x) = \boldsymbol{a}(x)$ for every $x \in \boldsymbol{z}^k \cap \boldsymbol{x}$, and setting $h^k(z) = \boldsymbol{b}(z)\boldsymbol{w}(z)$ for every $z \in \boldsymbol{z}^k$. Using the same argument as was used in the base case, we can show that h^k is well-defined. For atoms from $\boldsymbol{q}_k$ involving only variables from $\boldsymbol{q}_{k+1}$, we can use the induction hypothesis to conclude that they are satisfied under h^k, and for atoms only involving variables from $\boldsymbol{z}^k$, we can argue as in the base case. It thus remains to handle role atoms that contain one variable from $\boldsymbol{z}^k$ and one variable from $\boldsymbol{z}^{k+1}$. Consider such an atom $P(z,z') \in \boldsymbol{q}_k$, for $z \in \boldsymbol{z}^k$ and $z' \in \boldsymbol{z}^{k+1}$. If $\boldsymbol{w}(z) = \boldsymbol{s}(z') = \varepsilon$, then the atom $P(z,z')$ appears in the body of the clause we are considering. It follows that $\Pi_{\boldsymbol{Q}}^{\text{LIN}}, \mathcal{A} \models P(\boldsymbol{c}(z), \boldsymbol{c}(z'))$, hence $(\boldsymbol{c}(z), \boldsymbol{c}(z')) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. It then suffices to note that $\boldsymbol{c}$ agrees with $\boldsymbol{a}$ and $\boldsymbol{b}$ on the variables in $\boldsymbol{z}^k$. Next suppose that either $\boldsymbol{w}(z) \neq \varepsilon$ or $\boldsymbol{s}(z') \neq \varepsilon$. It follows that the clause body contains $z = z'$, hence $\boldsymbol{c}(z) = \boldsymbol{c}(z')$. As $(\boldsymbol{w}, \boldsymbol{s})$ is compatible with $(\boldsymbol{z}^k, \boldsymbol{z}^{k+1})$, one of the following must hold: either

(a) $\boldsymbol{s}(z') = \boldsymbol{w}(z)$ and $\mathcal{T} \models P(x,x)$

(b) or $\mathcal{T} \models \varrho(x,y) \to P(x,y)$ and either $\boldsymbol{s}(z') = \boldsymbol{w}(z)\varrho$ or $\boldsymbol{w}(z) = \boldsymbol{s}(z')\varrho^-$.

We give the argument in the case where $z \in \boldsymbol{z}_\exists^k$ (the argument is entirely similar if $z \in \boldsymbol{x}^k$). If (a) holds, then

$$(h^k(z), h^k(z')) = (\boldsymbol{b}(z)\boldsymbol{w}(z), \boldsymbol{c}(z')\boldsymbol{s}(z')) = (\boldsymbol{b}(z)\boldsymbol{w}(z), \boldsymbol{c}(z')\boldsymbol{w}(z)) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$$

since $\mathcal{T} \models P(x,x)$ and $\boldsymbol{c}(z') = \boldsymbol{c}(z) = \boldsymbol{b}(z)$. If the first option of (b) holds, then

$$(h^k(z), h^k(z')) = (\boldsymbol{b}(z)\boldsymbol{w}(z), \boldsymbol{c}(z')\boldsymbol{s}(z')) = (\boldsymbol{b}(z)\boldsymbol{w}(z), \boldsymbol{c}(z')\boldsymbol{w}(z)\varrho) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$$

since $\mathcal{T} \models \varrho(x,y) \to P(x,y)$ and $\boldsymbol{c}(z') = \boldsymbol{c}(z) = \boldsymbol{b}(z)$. If the second option of (b) holds, then

$$(h^k(z), h^k(z')) = (\boldsymbol{b}(z)\boldsymbol{w}(z), \boldsymbol{c}(z')\boldsymbol{s}(z')) = (\boldsymbol{b}(z)\boldsymbol{s}(z')\varrho^-, \boldsymbol{c}(z')\boldsymbol{s}(z')) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$$

since $\mathcal{T} \models \varrho(x, y) \to P(x, y)$.

For the converse direction, ($\Leftarrow$), of the induction step, let $\boldsymbol{w}$ be a type that is locally compatible with $\boldsymbol{z}^k$, let $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}^k|}$, $\boldsymbol{b} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{z}^k_\exists|}$, and let $h^k \colon \boldsymbol{q}_k \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ be a homomorphism satisfying

$$h^k(x) = \boldsymbol{a}(x), \qquad \text{for } x \in \boldsymbol{x}^k,$$
$$\text{and} \qquad h^k(z) = \boldsymbol{b}(z)\boldsymbol{w}(z), \qquad \text{for } z \in \boldsymbol{z}^k_\exists. \tag{10}$$

We let $\boldsymbol{c}$ for $\boldsymbol{z}^{k+1}$ be defined by setting $\boldsymbol{c}(z)$ equal to the unique individual c such that $h(z)$ is of the form cw (for some $w \in \boldsymbol{W}_\mathcal{T}$), and let $\boldsymbol{s}$ be the unique type for $\boldsymbol{z}^{k+1}$ satisfying $h(z) = \boldsymbol{c}(z)\boldsymbol{s}(z)$ for every $z \in \boldsymbol{z}^{k+1}$; in other words, we obtain $\boldsymbol{s}(z)$ from $h(z)$ by omitting the initial individual name $\boldsymbol{c}(z)$. Note that since $\boldsymbol{x}^{k+1} \subseteq \boldsymbol{x}^k$, we have $\boldsymbol{a}(x) = \boldsymbol{c}(x)$ for every $x \in \boldsymbol{x}^{k+1}$. It follows from the fact that h^k is a homomorphism that $\boldsymbol{s}$ is locally compatible with $\boldsymbol{z}^{k+1}$ and that, for every role atom $P(z, z') \in \boldsymbol{q}_k$ with $z \in \boldsymbol{z}^k$ and $z' \in \boldsymbol{z}^{k+1}$, one of the following holds: (*i*) $\boldsymbol{w}(z) = \boldsymbol{s}(z') = \varepsilon$, (*ii*) $\boldsymbol{w}(z) = \boldsymbol{s}(z')$ and $\mathcal{T} \models P(x, x)$, (*iii*) $\mathcal{T} \models \varrho(x, y) \to P(x, y)$ and either $\boldsymbol{s}(z') = \boldsymbol{w}(z)\varrho$ or $\boldsymbol{w}(z) = \boldsymbol{s}(z')\varrho^-$. Thus, the pair of types $(\boldsymbol{w}, \boldsymbol{s})$ is compatible with $(\boldsymbol{z}^k, \boldsymbol{z}^{k+1})$, and so the following rule appears in $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$:

$$G^{\boldsymbol{w}}_k(\boldsymbol{z}^k_\exists, \boldsymbol{x}^k) \leftarrow \mathsf{At}^{\boldsymbol{w} \cup \boldsymbol{s}}(\boldsymbol{z}^k, \boldsymbol{z}^{k+1}) \wedge G^{\boldsymbol{s}}_{k+1}(\boldsymbol{z}^{k+1}_\exists, \boldsymbol{x}^{k+1}),$$

where we recall that $\mathsf{At}^{\boldsymbol{w} \cup \boldsymbol{s}}(\boldsymbol{z}^k, \boldsymbol{z}^{k+1})$ is the conjunction of the following atoms, for $z, z' \in \boldsymbol{z}^k \cup \boldsymbol{z}^{k+1}$:

– $A(z)$, if $A(z) \in \boldsymbol{q}$ and $(\boldsymbol{w} \cup \boldsymbol{s})(z) = \varepsilon$,

– $P(z, z')$, if $P(z, z') \in \boldsymbol{q}$ and $(\boldsymbol{w} \cup \boldsymbol{s})(z) = (\boldsymbol{w} \cup \boldsymbol{s})(z') = \varepsilon$,

– $z = z'$, if $P(z, z') \in \boldsymbol{q}$ and either $(\boldsymbol{w} \cup \boldsymbol{s})(z) \neq \varepsilon$ or $(\boldsymbol{w} \cup \boldsymbol{s})(z') \neq \varepsilon$,

– $A_\varrho(z)$, if $(\boldsymbol{w} \cup \boldsymbol{s})(z)$ is of the form ϱw.

It follows from Equation (10) and the fact that h^k is a homomorphism that each of the ground atoms obtained by taking an atom from $\mathsf{At}^{\boldsymbol{w} \cup \boldsymbol{s}}(\boldsymbol{z}^k, \boldsymbol{z}^{k+1})$ and substituting $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ for $\boldsymbol{x}^k$, $\boldsymbol{z}^k_\exists$ and $\boldsymbol{z}^{k+1}$, respectively, is present in $\mathcal{A}$. By applying the induction hypothesis to the predicate $G^{\boldsymbol{s}}_{k+1}$ and the homomorphism $h^{k+1} \colon \boldsymbol{q}_{k+1} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ obtained by restricting h^k to $\mathsf{var}(\boldsymbol{q}_{k+1})$, we obtain that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{s}}_{k+1}(\boldsymbol{c}(\boldsymbol{z}^{k+1}_\exists), \boldsymbol{a}(\boldsymbol{x}^{k+1}))$. Since for the considered substitution, all body atoms are entailed, we can conclude that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}, \mathcal{A} \models G^{\boldsymbol{w}}_k(\boldsymbol{b}, \boldsymbol{a})$. ❑

It follows that answering OMQs $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathcal{T}$ of finite depth $\boldsymbol{d}$ and tree-shaped $\boldsymbol{q}$ with $\boldsymbol{\ell}$ leaves over any data instance $\mathcal{A}$ can be done in time

$$\mathit{poly}(|\mathcal{T}|^{\boldsymbol{d\ell}}, |\boldsymbol{q}|, |\mathcal{A}|^{\boldsymbol{\ell}}). \tag{5}$$

Indeed, $(\Pi^{\text{LIN}}_{\boldsymbol{Q}}, G(\boldsymbol{x}))$ can be evaluated in time polynomial in $|\Pi^{\text{LIN}}_{\boldsymbol{Q}}|$ and $|\mathcal{A}|^{\mathsf{w}(\Pi^{\text{LIN}}_{\boldsymbol{Q}}, G)}$, which are bounded by a polynomial in $|\mathcal{T}|^{2\boldsymbol{d\ell}}$, $|\boldsymbol{q}|$ and $|\mathcal{A}|^{2\boldsymbol{\ell}}$.

A.5 Tw-rewritings

Lemma 25. *For any OMQ $\boldsymbol{Q}(\boldsymbol{x}_0) = (\mathcal{T}, \boldsymbol{q}_0(\boldsymbol{x}_0))$ with a tree-shaped CQ, any complete data instance $\mathcal{A}$, any $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ and $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}|}$, we have $\Pi^{\text{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}}(\boldsymbol{a})$ iff there exists a homomorphism $h \colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(\boldsymbol{x}) = \boldsymbol{a}$.*

Proof. An inspection of the definition of the set $\mathfrak{Q}$ shows that every $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ is a tree-shaped query having at least one answer variable, with the possible exception of the original query $\boldsymbol{q}_0(\boldsymbol{x}_0)$, which may be Boolean.

Just as we did for subtrees in Section 3.2, we associate a binary relation on the queries in $\mathfrak{Q}$ by setting $\boldsymbol{q}'(\boldsymbol{x}') \prec \boldsymbol{q}(\boldsymbol{x})$ whenever $\boldsymbol{q}'(\boldsymbol{x}')$ was introduced when applying one of the two decomposition conditions on p. to $\boldsymbol{q}(\boldsymbol{x})$. The proof is by induction on the subqueries in $\mathfrak{Q}$, according to $\prec$. We will start by establishing the statement for all queries in $\mathfrak{Q}$ other than $\boldsymbol{q}_0(\boldsymbol{x}_0)$, and afterwards, we will complete the proof by giving an argument for $\boldsymbol{q}_0(\boldsymbol{x}_0)$.

For the basis of induction, take some $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ that is minimal in the ordering induced by $\prec$, which means that $\mathsf{var}(\boldsymbol{q}) = \boldsymbol{x}$. Indeed, if there is an existentially quantified variable, then the first decomposition rule will give rise to a 'smaller' query (in particular, if $|\mathsf{var}(\boldsymbol{q})| = 2$, then although the 'smaller' query may have the same atoms, the selected existential variable will become an answer variable). For the first direction, ($\Rightarrow$), suppose that $\Pi^{\text{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}}(\boldsymbol{a})$. By definition, $G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$ is the only clause with head predicate $G_{\boldsymbol{q}}$. Thus, all atoms in the ground CQ $\boldsymbol{q}(\boldsymbol{a})$ are present in $\mathcal{A}$, and hence the desired homomorphism exists. For the converse direction, ($\Leftarrow$), suppose there is a homomorphism $h \colon \boldsymbol{q}(\boldsymbol{x}) \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(\boldsymbol{x}) = \boldsymbol{a}$. It follows that every atom in the ground CQ $\boldsymbol{q}(\boldsymbol{a})$ is entailed from $\mathcal{T}, \mathcal{A}$. Completeness of $\mathcal{A}$ ensures that all of the ground atoms in $\boldsymbol{q}(\boldsymbol{a})$ are present in $\mathcal{A}$, and thus we can apply the clause $G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$ to derive $G_{\boldsymbol{q}}(\boldsymbol{a})$.

For the induction step, let $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ with $\mathsf{var}(\boldsymbol{q}) \neq \boldsymbol{x}$ and suppose that the claim holds for all $\boldsymbol{q}'(\boldsymbol{x}') \in \mathfrak{Q}$ with $\boldsymbol{q}'(\boldsymbol{x}') \prec \boldsymbol{q}(\boldsymbol{x})$. For the first direction, ($\Rightarrow$), suppose $\Pi^{\text{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}}(\boldsymbol{a})$. There are two cases, depending on which type of clause was used to derive $G_{\boldsymbol{q}}(\boldsymbol{a})$.

- Case 1: $G_{\boldsymbol{q}}(\boldsymbol{a})$ was derived by an application of the following clause:

  $$G_{\boldsymbol{q}}(\boldsymbol{z}) \leftarrow \bigwedge_{A(z_{\boldsymbol{q}}) \in \boldsymbol{q}} A(z_{\boldsymbol{q}}) \wedge \bigwedge_{P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \in \boldsymbol{q}} P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \wedge \bigwedge_{1 \le i \le n} G_{\boldsymbol{q}_i}(\boldsymbol{x}_i),$$

  where $\boldsymbol{q}_1(\boldsymbol{x}_1), \dots, \boldsymbol{q}_n(\boldsymbol{x}_n)$ are the subqueries induced by the neighbours of $z_{\boldsymbol{q}}$ in the Gaifman graph $\mathcal{G}$ of $\boldsymbol{q}$. Then there exists a substitution $\boldsymbol{c}$ for the variables in the body of this rule that coincides with $\boldsymbol{a}$ on $\boldsymbol{z}$ and is such that the ground atoms obtained by applying $\boldsymbol{c}$ to the variables in the body are all entailed from $\Pi^{\text{Tw}}_{\boldsymbol{Q}}, \mathcal{A}$. In particular, $\Pi^{\text{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models$

$G_{\boldsymbol{q}_i}(\boldsymbol{c}(\boldsymbol{x}_i))$ for every $1 \leq i \leq n$. We can apply the induction hypothesis to the $\boldsymbol{q}_i(\boldsymbol{x}_i)$ to obtain homomorphisms $h_i \colon \boldsymbol{q}_i \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h_i(\boldsymbol{x}_i) = \boldsymbol{c}(\boldsymbol{x}_i)$. Let h be the mapping from $\mathsf{var}(\boldsymbol{q})$ to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ defined by taking $h(z) = h_i(z)$, for $z \in \mathsf{var}(\boldsymbol{q}_i)$. Note that h is well-defined since $\mathsf{var}(\boldsymbol{q}) = \bigcup_{i=1}^n \mathsf{var}(\boldsymbol{q}_i)$, and the $\boldsymbol{q}_i$ have no variable in common other than $z_{\boldsymbol{q}}$, which is sent to $\boldsymbol{c}(z_{\boldsymbol{q}})$ by every h_i. To see why h is a homomorphism from $\boldsymbol{q}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, observe that

$$\boldsymbol{q} \;=\; \bigcup_{i=1}^n \boldsymbol{q}_i \;\cup\; \big\{A(z_{\boldsymbol{q}}) \in \boldsymbol{q}\big\} \;\cup\; \big\{P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \in \boldsymbol{q}\big\}.$$

By the definition of h, all atoms in $\bigcup_{i=1}^n \boldsymbol{q}_i$ hold under h. If $A(z_{\boldsymbol{q}}) \in \boldsymbol{q}$, then $A(\boldsymbol{c}(z_{\boldsymbol{q}}))$ is entailed from $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A}$, and hence is present in $\mathcal{A}$. Similarly, we can show that for every $P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \in \boldsymbol{q}$, the ground atom $P(\boldsymbol{c}(z_{\boldsymbol{q}}), \boldsymbol{c}(z_{\boldsymbol{q}}))$ belongs to $\mathcal{A}$. It follows that all of these atoms hold in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ under h. Finally, we recall that $\boldsymbol{c}$ coincides with $\boldsymbol{a}$ on $\boldsymbol{x}$, so we have $h(\boldsymbol{x}) = \boldsymbol{a}$, as required.

- Case 2: $G_{\boldsymbol{q}}(\boldsymbol{a})$ was derived by an application of the following clause, for a tree witness $\mathfrak{t}$ for $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ generated by ϱ with $\mathfrak{t}_\mathsf{r} \neq \emptyset$ and $z_{\boldsymbol{q}} \in \mathfrak{t}_\mathsf{i}$:

$$G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow A_\varrho(z_0) \wedge \bigwedge_{z \in \mathfrak{t}_\mathsf{r} \setminus \{z_0\}} (z = z_0) \;\wedge\; \bigwedge_{1 \leq i \leq k} G_{\boldsymbol{q}^\mathfrak{t}_i}(\boldsymbol{x}^\mathfrak{t}_i),$$

where $\boldsymbol{q}^\mathfrak{t}_1, \dots, \boldsymbol{q}^\mathfrak{t}_k$ are the connected components of $\boldsymbol{q}$ without $\boldsymbol{q}_\mathfrak{t}$ and z_0 is some variable in $\mathfrak{t}_\mathsf{r}$. There must exist a substitution $\boldsymbol{c}$ for the variables in the body of this rule that coincides with $\boldsymbol{a}$ on $\boldsymbol{x}$ and is such that the ground atoms obtained by applying $\boldsymbol{c}$ to the variables in the body are all entailed from $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A}$. In particular, for every $1 \leq i \leq k$, we have $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}^\mathfrak{t}_i}(\boldsymbol{c}(\boldsymbol{x}^\mathfrak{t}_i))$. We can apply the induction hypothesis to the $\boldsymbol{q}^\mathfrak{t}_i(\boldsymbol{z}^\mathfrak{t}_i)$ to find homomorphisms $h_1, \dots, h_k$ of $\boldsymbol{q}^\mathfrak{t}_1, \dots, \boldsymbol{q}^\mathfrak{t}_k$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h_i(\boldsymbol{x}^\mathfrak{t}_i) = \boldsymbol{c}(\boldsymbol{x}^\mathfrak{t}_i)$. Since $\mathfrak{t}$ is a tree witness for $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ generated by ϱ, there exists a homomorphism $h_\mathfrak{t}$ of $\boldsymbol{q}_\mathfrak{t}$ into $\mathcal{C}_{\mathcal{T},\{A_\varrho(a)\}}$ with $\mathfrak{t}_\mathsf{r} = h_\mathfrak{t}^{-1}(a)$ and such that $h_\mathfrak{t}(z)$ begins by $a\varrho$ for every $z \in \mathfrak{t}_\mathsf{i}$. Now take $z_0 \in \mathfrak{t}_\mathsf{r}$ such that $A_\varrho(z_0)$ is the atom in the clause body (recall that $\mathfrak{t}_\mathsf{r} \neq \emptyset$), and so $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models A_\varrho(\boldsymbol{c}(z_0))$, which means that $A_\varrho(\boldsymbol{c}(z_0))$ must appear in $\mathcal{A}$. It follows that for every element in $\mathcal{C}_{\mathcal{T},\{A_\varrho(a)\}}$ of the form $a\varrho w$, there exists a corresponding element $\boldsymbol{c}(z_0)\varrho w$ in $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. We now define a mapping h from $\mathsf{var}(\boldsymbol{q})$ to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ as follows:

$$h(z) = \begin{cases} h_i(z), & \text{for every } z \in \mathsf{var}(\boldsymbol{q}^\mathfrak{t}_i),\\ \boldsymbol{c}(z_0)\varrho w, & \text{if } z \in \mathfrak{t}_\mathsf{i} \text{ and } h_\mathfrak{t}(z) = a\varrho w,\\ \boldsymbol{c}(z_0) & \text{if } z \in \mathfrak{t}_\mathsf{r}.\end{cases}$$

Every variable in $\mathsf{var}(\boldsymbol{q})$ occurs in $\mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}$ or in exactly one of the $\boldsymbol{q}^\mathfrak{t}_i$, and so is assigned a unique value by h. Note that although $\mathfrak{t}_\mathsf{r} \cap \mathsf{var}(\boldsymbol{q}^\mathfrak{t}_i)$ is not necessarily empty, due to the equality atoms, we have $h(z) = h(z')$, for all $z, z' \in \mathfrak{t}_\mathsf{r}$, and so the function is well-defined. We claim that h is a homomorphism from $\boldsymbol{q}$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Clearly, the atoms occurring in some $\boldsymbol{q}^\mathfrak{t}_i$ are preserved under h. Now consider some unary atom $A(z)$ with $z \in \mathfrak{t}_\mathsf{i}$. Then $h(z) = \boldsymbol{c}(z_0)\varrho w$, where $h_\mathfrak{t}(z) = a\varrho w$. Since $h_\mathfrak{t}$ is a homomorphism, we know that w ends with a role σ such that $\mathcal{T} \models \exists y\, \sigma(y, x) \to A(x)$. It follows that $h(z)$ also ends with σ, and thus $h(z) \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. Next, consider a binary atom $P(z, z')$, where at least one of z and z' belongs to $\mathfrak{t}_\mathsf{i}$. As $h_\mathfrak{t}$ is a homomorphism, either
  - $\mathcal{T} \models \sigma(x, y) \to P(x, y)$, for some σ, such that $h_\mathfrak{t}(z') = h_\mathfrak{t}(z)\sigma$ or $h_\mathfrak{t}(z) = h_\mathfrak{t}(z')\sigma^-$,
  - or $\mathcal{T} \models P(x, x)$ and $h_\mathfrak{t}(z') = h_\mathfrak{t}(z)$.

  We also know that $\boldsymbol{c}(z) = \boldsymbol{c}(z_0)$ for all $z \in \mathfrak{t}_\mathsf{r}$, hence $h(z) = h(z_0)$ for all $z \in \mathfrak{t}_\mathsf{r}$. It follows that in the former case we have $h(z') = h(z)\sigma$ or $h(z) = h(z')\sigma^-$ with $\mathcal{T} \models \sigma(x, y) \to P(x, y)$. In the latter case, we have $h(z') = h(z)$ with $\mathcal{T} \models P(x, x)$. Thus, $P(z, z')$ is preserved under h. Finally, since $\boldsymbol{c}$ coincides with $\boldsymbol{a}$ on $\boldsymbol{x}$, we have $h(\boldsymbol{x}) = \boldsymbol{a}$.

For the converse direction, ($\Leftarrow$), of the induction step, suppose that h is a homomorphism of $\boldsymbol{q}$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(\boldsymbol{x}) = \boldsymbol{a}$. There are two cases to consider, depending on where h maps the 'splitting' variable $z_{\boldsymbol{q}}$.

- Case 1: $h(z_{\boldsymbol{q}}) \in \mathsf{ind}(\mathcal{A})$. Let $\boldsymbol{q}_1(\boldsymbol{x}_1), \dots, \boldsymbol{q}_n(\boldsymbol{x}_n)$ be the subqueries of $\boldsymbol{q}(\boldsymbol{x})$ induced by the neighbours of $z_{\boldsymbol{q}}$ in $\mathcal{G}$. Recall that $\boldsymbol{x}_i$ consists of $z_{\boldsymbol{q}}$ and the variables in $\mathsf{var}(\boldsymbol{q}_i) \cap \boldsymbol{x}$. By restricting h to $\mathsf{var}(\boldsymbol{q}_i)$, we obtain, for each $1 \leq i \leq n$, a homomorphism of $\boldsymbol{q}_i(\boldsymbol{x}_i)$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ that maps $z_{\boldsymbol{q}}$ to $h(z_{\boldsymbol{q}})$ and $\mathsf{var}(\boldsymbol{q}_i) \cap \boldsymbol{x}$ to $\boldsymbol{a}(\mathsf{var}(\boldsymbol{q}_i) \cap \boldsymbol{x})$. Consider $\boldsymbol{a}^*$ defined by taking $\boldsymbol{a}^*(x) = \boldsymbol{a}(x)$ for every $x \in \mathsf{var}(\boldsymbol{q}_i) \cap \boldsymbol{x}$ and $\boldsymbol{a}^*(z_{\boldsymbol{q}}) = h(z_{\boldsymbol{q}})$. By the induction hypothesis, for every $1 \leq i \leq n$, we have $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}_i}(\boldsymbol{a}^*(\boldsymbol{x}_i))$. Next, since h is a homomorphism, we must have $h(z_{\boldsymbol{q}}) \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ whenever $A(z_{\boldsymbol{q}}) \in \boldsymbol{q}$ and $(h(z_{\boldsymbol{q}}), h(z_{\boldsymbol{q}})) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ whenever $P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \in \boldsymbol{q}$. Since $\mathcal{A}$ is a complete data instance, $A(h(z_{\boldsymbol{q}})) \in \mathcal{A}$ for every $A(z_{\boldsymbol{q}}) \in \boldsymbol{q}$ and $P(h(z_{\boldsymbol{q}}), h(z_{\boldsymbol{q}}))$ for every $P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \in \boldsymbol{q}$. We have thus shown that, under the substitution $\boldsymbol{a}^*$, every atom in the body of the clause

$$G_{\boldsymbol{q}}(\boldsymbol{z}) \leftarrow \bigwedge_{A(z_{\boldsymbol{q}}) \in \boldsymbol{q}} A(z_{\boldsymbol{q}}) \;\wedge\; \bigwedge_{P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \in \boldsymbol{q}} P(z_{\boldsymbol{q}}, z_{\boldsymbol{q}}) \;\wedge\; \bigwedge_{1 \leq i \leq n} G_{\boldsymbol{q}_i}(\boldsymbol{x}_i),$$

  is entailed from $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A}$. It follows that we must also have $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}}(\boldsymbol{a})$.

- Case 2: $h(z_{\boldsymbol{q}}) \notin \mathsf{ind}(\mathcal{A})$. Then $h(z_{\boldsymbol{q}})$ is of the form $b\varrho w$, for some ϱ. Let V be the smallest subset of $\mathsf{var}(\boldsymbol{q})$ that contains $z_{\boldsymbol{q}}$ and satisfies the following closure property:
  - if $z \in V$, $h(z) \notin \mathsf{ind}(\mathcal{A})$ and $\boldsymbol{q}$ contains an atom with z and z', then $z' \in V$.

Let V' consist of all variables z in V such that $h(z) \notin \mathsf{ind}(\mathcal{A})$. We observe that $h(z)$ begins by $b\varrho$ for every $z \in V'$ and $h(z) = b$ for every $z \in V \setminus V'$. Define $\boldsymbol{q}_V$ as the CQ comprising all atoms in $\boldsymbol{q}$ whose variables are in V and which contain at least one variable from V'; the answer variables of $\boldsymbol{q}_V$ are $V \setminus V'$. By replacing the initial b by a in the mapping h, we obtain a homomorphism h_V of $\boldsymbol{q}_V$ into $\mathcal{C}_{\mathcal{T},\{A_\varrho(a)\}}$ with $V \setminus V' = h_V^{-1}(a)$. It follows that $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ with $\mathfrak{t}_\mathsf{r} = V \setminus V'$ and $\mathfrak{t}_\mathsf{i} = V'$ is a tree witness for $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ generated by ϱ (and $\boldsymbol{q}_\mathfrak{t} = \boldsymbol{q}_V$). Moreover, $\mathfrak{t}_\mathsf{r} \neq \emptyset$ because $\boldsymbol{q}$ has at least one answer variable. This means that the program $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}$ contains the following clause

$$G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow A_\varrho(z_0) \wedge \bigwedge_{z \in \mathfrak{t}_\mathsf{r} \setminus \{z_0\}} (z = z_0) \ \wedge \bigwedge_{1 \le i \le k} G_{\boldsymbol{q}_i^\mathfrak{t}}(\boldsymbol{x}_i^\mathfrak{t}),$$

where $\boldsymbol{q}_1^\mathfrak{t}, \dots, \boldsymbol{q}_k^\mathfrak{t}$ are the connected components of $\boldsymbol{q}$ without $\boldsymbol{q}_\mathfrak{t}$ and $z_0 \in \mathfrak{t}_\mathsf{r}$. Recall that the query $\boldsymbol{q}_i^\mathfrak{t}$ has answer variables $\boldsymbol{x}_i^\mathfrak{t} = \mathsf{var}(\boldsymbol{q}_i^\mathfrak{t}) \cap (\boldsymbol{x} \cup \mathfrak{t}_\mathsf{r})$. Let $\boldsymbol{a}^*$ be the substitution for $\boldsymbol{x} \cup \mathfrak{t}_\mathsf{r}$ such that $\boldsymbol{a}^*(x) = \boldsymbol{a}(x)$ for $x \in \boldsymbol{x}$ and $\boldsymbol{a}^*(z) = h(z)$ for $z \in \mathfrak{t}_\mathsf{r}$. Then, for every $1 \le i \le k$, there exists a homomorphism h_i from $\boldsymbol{q}_i^\mathfrak{t}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h_i(x) = \boldsymbol{a}^*(x)$ for every $x \in \boldsymbol{x}_i^\mathfrak{t}$. By the induction hypothesis, we obtain $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}_i^\mathfrak{t}}(\boldsymbol{a}^*(\boldsymbol{x}_i^\mathfrak{t}))$. Next, since $h(z) = b$ for every $z \in \mathfrak{t}_\mathsf{r}$, we have $\boldsymbol{a}^*(z) = \boldsymbol{a}^*(z')$ for every $z, z' \in \mathfrak{t}_\mathsf{r}$. Moreover, the presence of the element $b\varrho$ in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ means that $\mathcal{T}, \mathcal{A} \models A_\varrho(b)$. Since $\mathcal{A}$ is a complete data instance, we have $A_\varrho(b) \in \mathcal{A}$. It follows that under the substitution $\boldsymbol{a}^*$, all atoms in the body of the clause under consideration are entailed by $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A}$. Therefore, we must also have $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}}(\boldsymbol{a})$.

We have thus shown the lemma for all queries $\mathfrak{Q}$ other than $\boldsymbol{q}_0(\boldsymbol{x}_0)$. Let us now turn to $\boldsymbol{q}_0(\boldsymbol{x}_0)$.

For the first direction, ($\Rightarrow$), suppose $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}_0}(\boldsymbol{a})$. There are four cases, depending on which type of clause was used to derive $G_{\boldsymbol{q}_0}(\boldsymbol{a})$. We skip the first three cases, which are identical to those considered in the base case and induction step, and focus instead on the case in which $G_{\boldsymbol{q}_0}(\boldsymbol{a})$ was derived using a clause of the form $G_{\boldsymbol{q}_0} \leftarrow A(x)$ with A a unary predicate such that $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}_0$. In this case, there must exist some $b \in \mathsf{ind}(\mathcal{A})$ such that $\mathcal{T}, \mathcal{A} \models A(b)$. By completeness of $\mathcal{A}$, we obtain $A(b) \in \mathcal{A}$. Since $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}_0$, we get $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}_0$, which implies the existence of a homomorphism from $\boldsymbol{q}_0$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$.

For the converse direction, ($\Leftarrow$), suppose that there is a homomorphism $h\colon \boldsymbol{q}_0 \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(\boldsymbol{x}_0) = \boldsymbol{a}$. We focus on the case in which $\boldsymbol{q}_0$ is Boolean ($\boldsymbol{x}_0 = \emptyset$) and none of the variables in $\boldsymbol{q}_0$ is mapped to an individual constant (the other cases can be handled exactly as in the induction basis and induction step). In this case, there must exist an individual constant b and some ϱ such that $h(z)$ begins by $b\varrho$ for every $z \in \mathsf{var}(\boldsymbol{q}_0)$. It follows that $\mathcal{T}, \{A_\varrho(a)\} \models \boldsymbol{q}_0$, since the mapping h' defined by setting $h'(z) = a\varrho w$ whenever $h(z) = b\varrho w$ is a homomorphism from $\boldsymbol{q}_0$ to $\mathcal{C}_{\mathcal{T},\{A_\varrho(a)\}}$. It follows that $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}$ contains the clause $G_{\boldsymbol{q}_0} \leftarrow A_\varrho(x)$. Since $b\varrho$ occurs in $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, we have $\mathcal{T}, \mathcal{A} \models A_\varrho(b)$. By completeness of $\mathcal{A}$, $A_\varrho(b) \in \mathcal{A}$, and so by applying the clause $G_{\boldsymbol{q}_0} \leftarrow A_\varrho(x)$, we obtain $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}_0}$. ❑

A.6 Rewritings Zoo

In this section, we put together the rewritings from Sections 3.2–3.4 for the OMQ given in Examples 8 and 11.

Consider the CQ $\boldsymbol{q}(x_0, x_7)$ depicted below (black nodes represent answer variables)

$$x_0 \xrightarrow{R} x_1 \xrightarrow{S} x_2 \xrightarrow{R} x_3 \xrightarrow{R} x_4 \xrightarrow{S} x_5 \xrightarrow{R} x_6 \xrightarrow{R} x_7$$

and the following ontology $\mathcal{T}$ in normal form:

$$P(x, y) \to S(x, y), \qquad P(x, y) \to R(y, x),$$

$$A_P(x) \leftrightarrow \exists y\, P(x, y), \qquad A_{P^-}(x) \leftrightarrow \exists y\, P(y, x),$$
$$A_R(x) \leftrightarrow \exists y\, R(x, y), \qquad A_{R^-}(x) \leftrightarrow \exists y\, R(y, x),$$
$$A_S(x) \leftrightarrow \exists y\, S(x, y) \qquad A_{S^-}(x) \leftrightarrow \exists y\, S(y, x).$$

A.6.1 UCQ rewriting

The 9 CQs below form a UCQ rewriting of the OMQ $\boldsymbol{Q}(x_0, x_7) = (\mathcal{T}, \boldsymbol{q}(x_0, x_7))$ over complete data instances given as an NDL program with goal predicate G:

$$G(x_0, x_7) \leftarrow [R(x_0, x_1) \wedge S(x_1, x_2) \wedge R(x_2, x_3)] \wedge [R(x_3, x_4) \wedge S(x_4, x_5) \wedge R(x_5, x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [A_{P^-}(x_0) \wedge R(x_0, x_3)] \wedge [R(x_3, x_4) \wedge S(x_4, x_5) \wedge R(x_5, x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [R(x_0, x_3) \wedge A_P(x_3)] \wedge [R(x_3, x_4) \wedge S(x_4, x_5) \wedge R(x_5, x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [R(x_0, x_1) \wedge S(x_1, x_2) \wedge R(x_2, x_3)] \wedge [A_{P^-}(x_3) \wedge R(x_3, x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [R(x_0, x_1) \wedge S(x_1, x_2) \wedge R(x_2, x_3)] \wedge [R(x_3, x_6) \wedge A_P(x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [A_{P^-}(x_0) \wedge R(x_0, x_3)] \wedge [A_{P^-}(x_3) \wedge R(x_3, x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [A_{P^-}(x_0) \wedge R(x_0, x_3)] \wedge [R(x_3, x_6) \wedge A_P(x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [R(x_0, x_3) \wedge A_P(x_3)] \wedge [A_{P^-}(x_3) \wedge R(x_3, x_6)] \wedge R(x_6, x_7),$$
$$G(x_0, x_7) \leftarrow [R(x_0, x_3) \wedge A_P(x_3)] \wedge [R(x_3, x_6) \wedge A_P(x_6)] \wedge R(x_6, x_7).$$

We note that a UCQ rewriting over all data instances would in addition contain variants of the CQs above with each of the predicates R and S replaced by P (with arguments swapped appropriately).

The UCQ rewriting above can be obtained by transforming the following PE-formula into UCQ form:

$$\begin{aligned}
&\big[\big(R(x_0,x_1)\land S(x_1,x_2)\land R(x_2,x_3)\big)\\
&\quad\lor\big(A_{P^-}(x_0)\land R(x_0,x_3)\big)\lor\big(R(x_0,x_3)\land A_P(x_3)\big)\big]\\
\land\quad&\big[\big(R(x_3,x_4)\land S(x_4,x_5)\land R(x_5,x_6)\big)\\
&\quad\lor\big(A_{P^-}(x_3)\land R(x_5,x_6)\big)\lor\big(R(x_3,x_6)\land A_P(x_6)\big)\big]\\
\land\quad&R(x_6,x_7).
\end{aligned}$$

(Intuitively, each of the two sequences RSR in the query can be derived in three possible ways: from RSR, from $A_{P^-}R$ and from RA_P).

A.6.2 Log-*rewriting*

As explained in Example 11, we split T into D_1 and D_2 and obtain two rules:

$$G_T^{\boldsymbol{\varepsilon}}(x_0,x_7)\leftarrow G_{D_1}^{x_3\mapsto\varepsilon}(x_3,x_0)\land R(x_3,x_4)\land G_{D_2}^{x_4\mapsto\varepsilon}(x_4,x_7),$$

$$G_T^{\boldsymbol{\varepsilon}}(x_0,x_7)\leftarrow G_{D_1}^{x_3\mapsto\varepsilon}(x_3,x_0)\land A_{P^-}(x_4)\land(x_3=x_4)\land G_{D_2}^{x_4\mapsto P^-}(x_4,x_7).$$

Next, we split each of D_1 and D_2 into single-atom subqueries, which yields the following rules:

$$G_{D_1}^{x_3\mapsto\varepsilon}(x_3,x_0)\leftarrow(x_0=x_1)\land A_{P^-}(x_1)\land(x_1=x_2)\land R(x_2,x_3),$$

$$G_{D_1}^{x_3\mapsto\varepsilon}(x_3,x_0)\leftarrow R(x_0,x_1)\land(x_1=x_2)\land A_P(x_2)\land(x_2=x_3),$$

$$G_{D_1}^{x_3\mapsto\varepsilon}(x_3,x_0)\leftarrow R(x_0,x_1)\land S(x_1,x_2)\land R(x_2,x_3),$$

$$G_{D_2}^{x_4\mapsto\varepsilon}(x_4,x_7)\leftarrow(x_4=x_5)\land A_P(x_5)\land(x_5=x_6)\land R(x_6,x_7),$$

$$G_{D_2}^{x_4\mapsto\varepsilon}(x_4,x_7)\leftarrow S(x_4,x_5)\land R(x_5,x_6)\land R(x_6,x_7),$$

$$G_{D_2}^{x_4\mapsto P^-}(x_4,x_7)\leftarrow A_{P^-}(x_4)\land(x_4=x_5)\land R(x_5,x_6)\land R(x_6,x_7).$$

Note that in each case we consider only those types that give rise to predicates that have definitions in the rewriting. The resulting NDL rewriting with goal $G_T^{\boldsymbol{\varepsilon}}$ consists of 8 rules. Note, however, that the rewriting illustrated above is a slight simplification of the definition given in Section 3.2: here, for the leaves of the tree decomposition, we directly use the atoms $\mathsf{At}^{\boldsymbol{s}}$ instead of including a rule $G_D^{\boldsymbol{w}}(\partial D,\boldsymbol{x}_D)\leftarrow\mathsf{At}^{\boldsymbol{s}}$ in the rewriting. This simplification clearly does not affect the width of the NDL query or the choice of weight function.

A.6.3 Lin-*rewriting*

We assume that x_0 is the root, which makes x_7 the only leaf of the query. (Note that we could have chosen another variable, say x_3, as the root, with x_0 and x_7 the two leaves.) So, the top-level rule is

$$G(x_0,x_7)\leftarrow G_0^{x_0\mapsto\varepsilon}(x_0,x_7).$$

We then move along the query and consider the variables x_1, x_2 and x_3. The possible ways of mapping these variables to the canonical model give rise to the following 7 rules:

$$\begin{aligned}
G_0^{x_0\mapsto\varepsilon}(x_0,x_7)&\leftarrow R(x_0,x_1)\land P_1^{x_1\mapsto\varepsilon}(x_1,x_7),\\
G_0^{x_0\mapsto\varepsilon}(x_0,x_7)&\leftarrow(x_0=x_1)\land A_{P^-}(x_1)\land G_1^{x_1\mapsto P^-}(x_1,x_7),\\[4pt]
G_1^{x_1\mapsto\varepsilon}(x_1,x_7)&\leftarrow S(x_1,x_2)\land G_2^{x_2\mapsto\varepsilon}(x_2,x_7),\\
G_1^{x_1\mapsto\varepsilon}(x_1,x_7)&\leftarrow(x_1=x_2)\land A_P(x_2)\land G_2^{x_2\mapsto P}(x_2,x_7),\\
G_1^{x_1\mapsto P^-}(x_1,x_7)&\leftarrow A_{P^-}(x_1)\land(x_1=x_2)\land G_2^{x_2\mapsto\varepsilon}(x_2,x_7),\\[4pt]
G_2^{x_2\mapsto\varepsilon}(x_2,x_7)&\leftarrow R(x_2,x_3)\land G_3^{x_3\mapsto\varepsilon}(x_3,x_7),\\
G_2^{x_2\mapsto P}(x_2,x_7)&\leftarrow A_P(x_2)\land(x_2=x_3)\land G_3^{x_3\mapsto\varepsilon}(x_3,x_7).
\end{aligned}$$

Next, we move to the variables x_4, x_5 and x_6, which give similar 7 rules:

$$\begin{aligned}
G_3^{x_3\mapsto\varepsilon}(x_3,x_7)&\leftarrow R(x_3,x_4)\land P_4^{x_4\mapsto\varepsilon}(x_4,x_7),\\
G_3^{x_3\mapsto\varepsilon}(x_3,x_7)&\leftarrow(x_3=x_4)\land A_{P^-}(x_4)\land G_4^{x_4\mapsto P^-}(x_4,x_7),\\[4pt]
G_4^{x_4\mapsto\varepsilon}(x_4,x_7)&\leftarrow S(x_4,x_5)\land G_5^{x_5\mapsto\varepsilon}(x_5,x_7),\\
G_4^{x_4\mapsto\varepsilon}(x_4,x_7)&\leftarrow(x_4=x_5)\land A_P(x_5)\land G_5^{x_5\mapsto P}(x_5,x_7),\\
G_4^{x_4\mapsto P^-}(x_4,x_7)&\leftarrow A_{P^-}(x_4)\land(x_4=x_5)\land G_5^{x_5\mapsto\varepsilon}(x_5,x_7),\\[4pt]
G_5^{x_5\mapsto\varepsilon}(x_5,x_7)&\leftarrow R(x_5,x_6)\land G_6^{x_6\mapsto\varepsilon}(x_6,x_7),\\
G_5^{x_5\mapsto P}(x_5,x_7)&\leftarrow A_P(x_2)\land(x_5=x_6)\land G_6^{x_6\mapsto\varepsilon}(x_6,x_7).
\end{aligned}$$

Finally, the last variable can only be mapped to a constant in the data instance, which yields a single rule:

$$G_6^{x_6\mapsto\varepsilon}(x_6,x_7)\leftarrow R(x_6,x_7).$$

Note that, like in the previous case, we consider only those types that give rise to predicates with definitions (and ignore the dead-ends in the construction).

A.6.4 Tw-*rewriting*

We begin by splitting the query roughly in the middle, that is, we choose x_3 and consider two subqueries:

$$\boldsymbol{q}_{03}(x_0,x_3)=\exists x_1x_2\,\big(R(x_0,x_1)\land S(x_1,x_2)\land R(x_2,x_3)\big)$$

and

$$\boldsymbol{q}_{37}(x_3,x_7)=\exists x_4x_5x_6\,\big(R(x_3,x_4)\land S(x_4,x_5)\land R(x_5,x_6)\land R(x_6,x_7)\big).$$

Since there is no tree witness $\mathfrak{t}$ for $(\mathcal{T},\boldsymbol{q}(x_0,x_7))$ that contains x_3 in $\mathfrak{t}_{\mathsf{i}}$, we have only one top-level rule:

$$G_{07}(x,y)\leftarrow G_{03}(x_0,x_3)\land G_{37}(x_3,x_7).$$

Next, we focus on $\boldsymbol{q}_{03}$ and choose x_1 as the splitting variable. In this case, there is a tree witness $\mathfrak{t}^1$ with $\mathfrak{t}^1_{\mathsf{i}}=\{x_1\}$ and $\mathfrak{t}^1_{\mathsf{r}}=\{x_0,x_2\}$, and so we obtain two rules for G_{03}:

$$G_{03}(x_0,x_3)\leftarrow R(x_0,x_1)\land G_{13}(x_1,x_3),$$

$$G_{03}(x_0, x_3) \leftarrow A_{P^-}(x_0) \wedge (x_0 = x_2) \wedge R(x_2, x_3).$$

The subquery $\boldsymbol{q}_{13}(x_1, x_3) = \exists x_2 \big(S(x_1, x_2) \wedge R(x_2, x_3)\big)$ contains two atoms and is split at x_2. Since there is a tree witness $\mathfrak{t}^2$ for $(\mathcal{T}, \boldsymbol{q}_{13}(x_1, x_3))$ with $\mathfrak{t}^2_\mathsf{i} = \{x_2\}$ and $\mathfrak{t}^2_\mathsf{r} = \{x_1, x_3\}$, we obtain two rules:

$$G_{13}(x_1, x_3) \leftarrow S(x_1, x_2) \wedge R(x_2, x_3),$$
$$G_{13}(x_1, x_3) \leftarrow A_P(x_1) \wedge (x_1 = x_3).$$

By applying the same procedure to $\boldsymbol{q}_{37}(x_3, x_7)$, we get the following five rules:

$$G_{37}(x_3, x_7) \leftarrow G_{35}(x_3, x_5) \wedge G_{57}(x_5, x_7),$$
$$G_{37}(x_5, x_7) \leftarrow R(x_3, x_4) \wedge A_P(x_4) \wedge (x_4 = x_6) \wedge R(x_6, x_7),$$
$$G_{35}(x_3, x_5) \leftarrow R(x_3, x_5) \wedge S(x_5, x_7),$$
$$G_{35}(x_3, x_5) \leftarrow A_{P^-}(x_3) \wedge (x_3 = x_5),$$
$$G_{57}(x_3, x_5) \leftarrow R(x_3, x_4) \wedge R(x_4, x_7).$$

Note that the rewriting illustrated above is slightly simpler than the definition in Section 3.4: here, we directly use the atoms of $\boldsymbol{q}(\boldsymbol{x})$ instead of including a rule $G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$, for each $\boldsymbol{q}(\boldsymbol{x})$ without existentially quantified variables. This simplification clearly does not affect the width of the NDL query and the choice of weight function.

B. PROOFS FOR SECTION 4

B.1 Theorem 15

THEOREM 15. *pDepth-*TREEOMQ *is* $W[2]$*-hard.*

PROOF. We show that $\mathcal{T}_H^k, \{V_0^0(a)\} \models \boldsymbol{q}_H^k$ iff H has a hitting set of size k. Denote by $\mathcal{C}$ the canonical model of $(\mathcal{T}_H^k, \{V_0^0(a)\})$. For convenience of reference to the points of the canonical model we assume that $\mathcal{T}_H^k$ contains the following axioms:

$$\begin{aligned}
V_i^{l-1}(x) &\to \exists z\, v_{i'}^l(x, z) \text{ and} \\
v_{i'}^l(x, z) &\to P(z, x) \wedge V_{i'}^l(z), && \text{for } 0 \leq i < i' \leq n, \\
V_i^l(x) &\to E_j^l(z), && \text{for } v_i \in e_j,\ e_j \in E, \\
E_j^l(x) &\to \exists z\, \eta_j^l(x, z) \text{ and} \\
\eta_j^l(x, z) &\to P(x, z) \wedge E_j^{l-1}(z), && \text{for } 1 \leq j \leq m.
\end{aligned}$$

We show that $\mathcal{C} \models \boldsymbol{q}_H^k$ iff H has a hitting set of size k.

($\Rightarrow$) Suppose $h\colon \boldsymbol{q}_H^k \to \mathcal{C}$ is a homomorphism. Note that $\mathcal{C}$ satisfies the following properties: (i) $w \in E_j^0$ iff $w = av_{i_1}^1 v_{i_2}^2 \dots v_{i_s}^s \eta_j^s \eta_j^{s-1} \dots \eta_j^1$ where $v_{j_s} \in e_j$ and (ii) all points in $\Delta^{\mathcal{C}}$ have at most one P-predecessor. By starting with some E_j^0 atom and applying first (i) and then iterating (ii), we conclude that $h(y) = av_{i_1}^1 \dots v_{i_k}^k$ for some $1 \leq i_1 < i_2 < \dots i_k \leq n$. We claim that $\{v_{i_1}, v_{i_2}, \dots, v_{i_k}\}$ is a hitting set in H. Indeed, for every branch j of $\boldsymbol{q}_H^k$, there is $1 \leq s \leq k$ such that this branch is mapped on $\mathcal{C}$ in the following way:

$$\begin{aligned}
h(z_j^l) &= av_{i_1}^1 v_{i_2}^2 \dots v_{i_l}^l, && s \leq l \leq k-1, \\
h(z_j^l) &= av_{i_1}^1 v_{i_2}^2 \dots v_{i_s}^s \eta_j^s \eta_j^{s-1} \dots \eta_j^{l+1}, && 0 \leq l < s,
\end{aligned}$$

with $v_{i_s} \in e_j$. This can be shown by induction on l from 0 to $k-1$ using (i) to prove the base of induction and (ii) to prove the induction step. Therefore, for every j, there exists s such that $v_{i_s} \in e_j$.

($\Leftarrow$) Suppose $\{v_{i_1}, v_{i_2}, \dots, v_{i_k}\}$ is a hitting set in H. We construct a homomorphism h from $\boldsymbol{q}_H^k$ to $\mathcal{C}$. First, we set $h(y) = av_{i_1}^1 \dots v_{i_k}^k$. Then, for each $1 \leq j \leq m$, we find s such $v_{i_s} \in e_j$ and define h as follows:

$$\begin{aligned}
h(z_j^l) &= av_{i_1}^1 v_{i_2}^2 \dots v_{i_l}^l, && s \leq l \leq k-1, \\
h(z_j^l) &= av_{i_1}^1 v_{i_2}^2 \dots v_{i_s}^s \eta_j^s \eta_j^{s-1} \dots \eta_j^{l+1}, && 0 \leq l < s.
\end{aligned}$$

It should be clear that h is indeed a homomorphism. ❑

B.2 Theorem 16

THEOREM 16. *pLeaves-*TREEOMQ *is* $W[1]$*-hard.*

PROOF. We prove that $\mathcal{T}_G, \{A(a)\} \models \boldsymbol{q}_G$ iff G has a clique containing one vertex from each set V_i.

We start with some preliminaries. First note we assume that the final axiom in $\mathcal{T}_G$ (which uses the syntactic sugar $\wedge$) is actually given by the following three axioms (where P is a fresh binary predicate):

$$\begin{aligned}
B(x) &\to \exists y\, P(x, y), \\
P(x, y) &\to U(x, y), \\
P(x, y) &\to U(y, x).
\end{aligned}$$

To simplify notation, we will abbreviate $\mathcal{C}_{\mathcal{T}_G, \{A(a)\}}$ by $\mathcal{C}$, and for every $1 \leq j \leq M$, we let $\mathsf{w}(v_j) = L_j^1 L_j^2 \dots L_j^{2M}$. Observe that for every $v_{j_1} \in V_1, v_{j_2} \in V_2, \dots, v_{j_p} \in V_p$, the element $a\mathsf{w}(v_{j_1})\mathsf{w}(v_{j_2}) \dots \mathsf{w}(v_{j_p})$ belongs to $\Delta^{\mathcal{C}}$. Further, observe that if $aw \in \Delta^{\mathcal{C}}$ with $|w| = 2M \cdot p$, then there exist $v_{j_1} \in V_1, v_{j_2} \in V_2, \dots, v_{j_p} \in V_p$ such that $w = \mathsf{w}(v_{j_1})\mathsf{w}(v_{j_2}) \dots \mathsf{w}(v_{j_p})$.

($\Rightarrow$) Suppose that $\mathcal{T}_G, \{A(a)\} \models \boldsymbol{q}_G$, and let h be a homomorphism of $\boldsymbol{q}_G$ into $\mathcal{C}$. Note that because of the atom $B(y)$, the variable y must be sent by h to an element occurring at the end of the pth block. As noted above, every such element takes the form

$$a\mathsf{w}(v_{j_1})\mathsf{w}(v_{j_2}) \dots \mathsf{w}(v_{j_p})$$

where $v_{j_1} \in V_1, v_{j_2} \in V_2, \dots, v_{j_p} \in V_p$. We claim that $\{v_{j_1}, \dots, v_{j_p}\}$ is a clique in G. To see why, consider the ith branch of $\boldsymbol{q}_G$, compactly represented as follows:

$$\big(U^{2M-2} \cdot (YY \cdot U^{2M-2})^i \cdot SS\big)(y, z_i)$$

By examining the axioms, we see that starting from the first occurrence of YY, every U and Y atom takes us one step closer to a (prior to the first YY, we may go back and forth on the extra P-edge leaving from $h(y)$). It follows that SS must be mapped within the p-ith block of the selected branch, and since S is present only at positions $2j_{p-i}$ and $2j_{p-i}+1$ of the block, we must have $h(z_i) = a\mathsf{w}(v_{j_1}) \dots \mathsf{w}(v_{j_{p-i-1}}) L_{j_{p-i}}^1 \dots L_{j_{p-i}}^{2j_{p-i}-1}$. As the distance between consecutive occurrences of YY (and between the final YY and the SS) is $2M-2$, it follows that all YY blocks occur at positions $2j_{p-i}$ and $2j_{p-i}+1$ of blocks $p-i+1,\dots,p$, which implies that $v_{j_{p-i+1}},\dots,v_{j_p}$ are neighbours of v_{j_i} in G. Since $\boldsymbol{q}_G$ contains branches for every $1 \le i < p$, the selected vertices $v_{j_1},\dots,v_{j_p}$ are all neighbours in G, and G contains a clique with the required properties.

($\Leftarrow$) Suppose that $v_{j_1} \in V_1,\dots,v_{j_p} \in V_p$ form a clique. We construct a homomorphism h of $\boldsymbol{q}_G$ into $\mathcal{C}$. First, set $h(y) = aw$ where $w = \mathsf{w}(v_{j_1})\mathsf{w}(v_{j_2})\dots\mathsf{w}(v_{j_p})$ and observe that the atom $B(y)$ is satisfied by this assignment. We will use $w[\ell,\ell']$ to denote the subword of w beginning with the ℓth symbol of w and ending with the ℓ'th symbol (note that $w = |2M\cdot p|$, so $w = w[1, 2M\cdot p]$). Next, consider the ith branch of the query, which connects y to z_i, and let $y_0, y_1, \dots y_{2M(i+1)}$ be the variables lying between y and z_i with $y_0 = y$ and $z_i = y_{2M(i+1)}$. For $0 \le k \le 2j_{p-i}$, we set $h(y_k) = h(y)$ if k is even, and set $h(y_k) = h(y)P$ otherwise. Observe that because P is included in both U and U^-, we satisfy all binary atoms between variables from $\{y_0,\dots,y_{2j_{p-i}}\}$. For $2j_{p-i} < k \le 2M(i+1)$, we set

$$h(y_k) = aw[1, 2M\cdot p - (k - 2j_{p-i})].$$

Note that, in particular, this yields

$$\begin{aligned}
h(y_{2M(i+1)-2}) &= aw[1, 2M(p-i-1) + 2j_{p-i} + 2],\\
h(y_{2M(i+1)-1}) &= aw[1, 2M(p-i-1) + 2j_{p-i} + 1],\\
h(y_{2M(i+1)}) &= aw[1, 2M(p-i-1) + 2j_{p-i}],
\end{aligned}$$

so the final two S-atoms in the branch are satisfied by h. It is easy to see that all U-atoms between variables from $y_{2j_{p-i}},\dots,y_{2M(i+1)}$ are also satisfied. Finally, using the fact that vertices $v_{j_{p-i+1}},\dots,v_{j_p}$ are neighbours of $v_{j_{p-i}}$, we can show that all of the Y-atoms in the ith branch are satisfied by h. As we have constructed a homomorphism from $\boldsymbol{q}_G$ into $\mathcal{C}$, we can conclude $\mathcal{T}_G, \{A(a)\} \models \boldsymbol{q}_G$. ❑

C. PROOFS FOR SECTION 5

C.1 Theorem 17

THEOREM 17. *There is an ontology $\mathcal{T}_\dagger$ such that answering OMQs of the form $(\mathcal{T}_\dagger, \boldsymbol{q})$ with Boolean tree-shaped CQs $\boldsymbol{q}$ is* NP*-hard for query complexity.*

PROOF. We assume that $\mathcal{T}_\dagger$ consists of the following axioms:

$$\begin{aligned}
&A(x) \to \exists y\, \upsilon_+(x,y)\\
&\upsilon_+(x,y) \to P_+(y,x) \land P_0(y,x) \land B_-(y) \land A(y),\\
&\quad B_-(x) \to \exists y\, \eta_-(x,y)\\
&\quad \eta_-(x,y) \to P_-(x,y) \land B_0(y),\\
&A(x) \to \exists y\, \upsilon_-(x,y)\\
&\upsilon_-(x,y) \to P_-(y,x) \land P_0(y,x) \land B_+(y) \land A(y),\\
&\quad B_+(x) \to \exists y\, \eta_+(x,y)\\
&\quad \eta_+(x,y) \to P_+(x,y) \land B_0(y),\\
&\quad B_0(x) \to \exists y\, \eta_0(x,y)\\
&\quad \eta_0(x,y) \to P_+(x,y) \land P_-(x,y) \land P_0(x,y) \land B_0(y).
\end{aligned}$$

Let $\mathcal{C}$ be the canonical model of $(\mathcal{T}_\dagger, \{A(a)\})$. We prove that $\mathcal{C} \models \boldsymbol{q}_\varphi$ iff φ is satisfiable.

($\Rightarrow$) Suppose h is a homomorphism from $\boldsymbol{q}_\varphi$ to $\mathcal{C}$ and $h(z_j^k) = h(y) = a\varrho_1\dots\varrho_n$, for some roles ϱ_l. Since $A(y) \in \boldsymbol{q}_\varphi$, it follows that $\varrho_l \in \{\upsilon_+, \upsilon_-\}$. Moreover, because of the structure of $\mathcal{C}$, without any loss of generality we may assume that $n = k$. Define a valuation $\nu\colon \{p_1,\dots,p_k\} \to \{\mathsf{t},\mathsf{f}\}$ by taking $\nu(p_l) = \mathsf{t}$ if $\varrho_l = \upsilon_-$, $\nu(p_l) = \mathsf{f}$, if $\varrho_l = \upsilon_+$. We claim that ν makes φ true. To verify that the clause χ_j is satisfied, consider a number $1 \le s \le k$, such that the jth branch of the query is mapped on $\mathcal{C}$ in the following way:

$$\begin{aligned}
h(z_j^l) &= a\varrho_1\dots\varrho_l, && s \le l \le k,\\
h(z_j^l) &= a\varrho_1\dots\varrho_s\gamma_1\dots\gamma_{s-l}, && 0 \le l < s,
\end{aligned}$$

for some roles $\gamma_1\dots\gamma_{s-l}$ with $\gamma_1 \in \{\eta_-, \eta_+\}$ and $\gamma_i = \eta_0$ for $2 \le i \le s-l$. Such s and the roles γ_i exist, because the P-atoms in $\mathcal{C}$ are directed towards the root if they cover υ-atoms, and away from the root if they cover η-atoms ($s \ge 1$ since $B_0(z_j^0) \in \boldsymbol{q}_\varphi$). Clearly, $\mathcal{T}_\dagger \models \gamma_1(x,y) \to P_+(x,y)$ iff $\rho_s = \upsilon_-$ iff $\nu(p_s) = \mathsf{t}$ and $\mathcal{T}_\dagger \models \gamma_1(x,y) \to P_-(x,y)$ iff $\rho_s = \upsilon_+$ iff $\nu(p_s) = \mathsf{f}$. It follows that either $P_+(z_j^s, z_j^{s-1}) \in \boldsymbol{q}_\varphi$ and $\nu(p_s) = \mathsf{t}$, or $P_-(z_j^s, z_j^{s-1}) \in \boldsymbol{q}_\varphi$ and $\nu(p_s) = \mathsf{f}$. In either case, χ_j contains a literal with p_s satisfied by ν.

($\Leftarrow$) Suppose a valuation $\nu\colon \{p_1,\dots,p_k\} \to \{\mathsf{t},\mathsf{f}\}$ satisfies φ. Consider the sequence of roles $\varrho_1\dots\varrho_k$, such that for $1 \le l \le k$ we have $\varrho_l = \upsilon_+$, if $\nu(p_l) = \mathsf{f}$, and $\varrho_l = \upsilon_-$, if $\nu(p_l) = \mathsf{t}$. We claim that there exists a homomorphism h from q_φ to $\mathcal{C}$. First, let $h(y) = a\varrho_1\dots\varrho_k$. To map the jth branch of the query, consider the maximal $1 \le s \le k$, such that a p_s-literal (positive or negative) makes χ_j true. Set

$$\begin{aligned}
h(z_j^l) &= a\varrho_1\dots\varrho_l, && s \le l \le k-1,\\
h(z_j^l) &= a\varrho_1\dots\varrho_s\gamma_1\dots\gamma_{s-l}, && 0 \le l < s,
\end{aligned}$$

where $\gamma_1 = \eta_+$ if p_s occurs positively, $\gamma_1 = \eta_-$ if p_s occurs negatively and $\gamma_i = \eta_0$ for $i \ge 2$. That z_j^l, for $s \le l \le k-1$, are mapped correctly follows from the maximality of s. That z_j^l is mapped correctly for $l = s-1$ follows from the fact that p_s occurs in χ_j positively iff $P_+(z_j^s, z_j^{s-1}) \in \boldsymbol{q}_\varphi$ iff $\nu(p_s) = \mathsf{t}$ iff $\varrho_s = \upsilon_-$ iff $\gamma_1 = \eta_+$ (similarly for negative p_s). Finally, z_j^l is mapped correctly for $0 \le l < s-1$ since the sequence of roles $\gamma_2\dots\gamma_{s-l}$ can embed any P_+, P_-, or P_0 roles, and B_0 concept. Thus, h is a homomorphism from $\boldsymbol{q}_\varphi$ to $\mathcal{C}$. ❑

C.2 Theorem 20

We need several intermediate results and definitions before we present the proof in the end of the section. Suppose φ is a propositional formula in CNF having

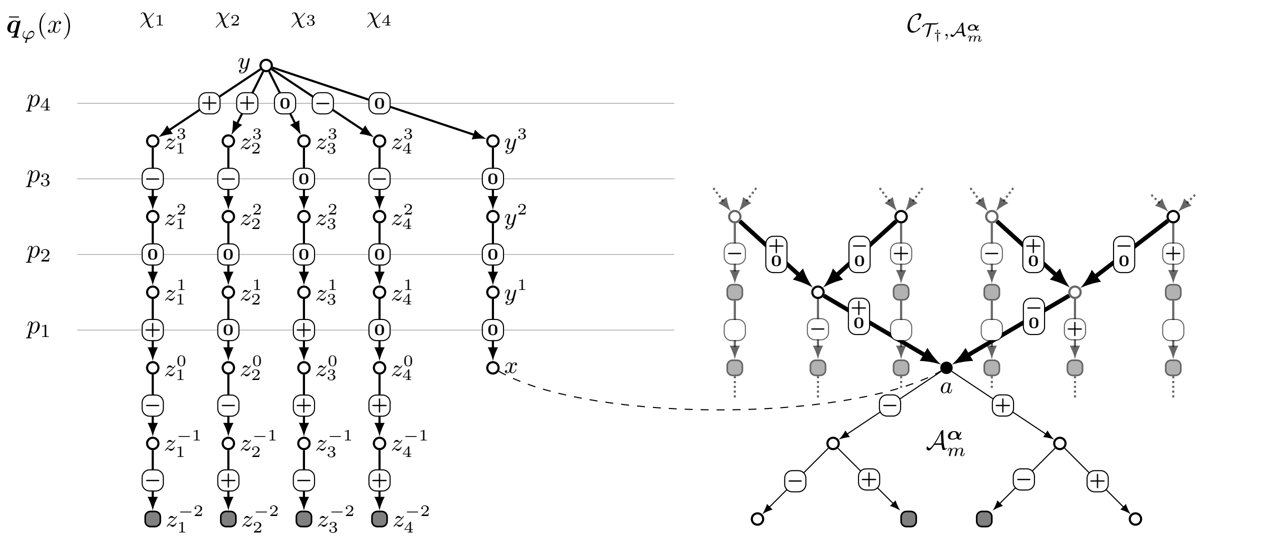


Figure 3: Example of $\bar{\boldsymbol{q}}_\varphi(x)$ and $\mathcal{C}_{\mathcal{T}_\dagger,\mathcal{A}_m^{\boldsymbol{\alpha}}}$ for $\varphi = \chi_1 \land \dots \land \chi_4$ with $\chi_1 = (p_1 \lor \neg p_3 \lor p_4)$, $\chi_2 = (\neg p_3 \land p_4)$, $\chi_3 = p_1$, $\chi_4 = (\neg p_3 \lor \neg p_4)$ and $\boldsymbol{\alpha} = (0,1,1,0)$

k variables $p_1, \dots, p_k$ and m clauses $\chi_1, \dots, \chi_m$. We assume that $m = 2^\ell$. We associate with every such φ a CQ $\bar{\boldsymbol{q}}_\varphi(x)$ with one answer variable x and the following atoms, where $1 \le j \le m$, $1 \le l \le k$, and $z_j^k = y^k$:

$$P_0(y^1, x), \dots, P_0(y^k, y^{k-1}),$$

$$P_+(z_j^l, z_j^{l-1}) \qquad \text{if } \chi_j \text{ contains } p_l,$$

$$P_-(z_j^l, z_j^{l-1}), \qquad \text{if } \chi_j \text{ contains } \neg p_l,$$

$$P_0(z_j^l, z_j^{l-1}), \qquad \text{if } \chi_j \text{ contains no occurrence of } p_l.$$

Then, for $0 \le l \le \ell - 1$,

$$P_-(z_j^{-l}, z_j^{-l-1}), \qquad \text{if the } l\text{th bit of } (j-1)_2 \text{ is } 0,$$

$$P_+(z_j^{-l}, z_j^{-l-1}), \qquad \text{if the } l\text{th bit of } (j-1)_2 \text{ is } 1,$$

$$B_0(z_j^{-\ell}).$$

See an example in Fig. 3. For any $\boldsymbol{\alpha} \in \{0,1\}^m$, define a data instance $\mathcal{A}_m^{\boldsymbol{\alpha}}$ as the full binary tree of depth ℓ (and so $m = 2^\ell$ leaves) on the binary predicates P_- (for the left child) and P_+ (for the right child); $\mathcal{A}_m^{\boldsymbol{\alpha}}$ contains $A(a)$ for the root a of the tree and, for every ith leaf b_i of the tree, $B_0(b_i) \in \mathcal{A}_m^{\boldsymbol{\alpha}}$ iff $\boldsymbol{\alpha}_i = 1$.

Denote by $f_\varphi \colon \{0,1\}^m \to \{0,1\}$ the *monotone* function such that $f_\varphi(\boldsymbol{\alpha}) = 1$ iff the CNF $\varphi^{-\boldsymbol{\alpha}}$, which is obtained from φ by removing all conjuncts χ_i with $\boldsymbol{\alpha}_i = 1$, is satisfiable. It is readily checked that we have

LEMMA 26. *For any $\boldsymbol{\alpha} \in \{0,1\}^m$,*

$$\mathcal{T}_\dagger, \mathcal{A}_m^{\boldsymbol{\alpha}} \models \bar{\boldsymbol{q}}_\varphi(a) \quad \textit{iff} \quad f_\varphi(\boldsymbol{\alpha}) = 1.$$

Let $\mathcal{QL}$ be any query language such that, for any $\mathcal{QL}$-query $\Phi(x)$ and any $\mathcal{A}_m^{\boldsymbol{\alpha}}$, the answer to $\Phi(a)$ over $\mathcal{A}_m^{\boldsymbol{\alpha}}$ can be computed in time *poly*$(|\Phi|, m)$.

THEOREM 27. *The OMQ $(\mathcal{T}_\dagger, \bar{\boldsymbol{q}}_\varphi(x))$ does not have a polynomial-size rewriting in $\mathcal{QL}$ unless* NP $\subseteq$ P/poly.

PROOF. Take any sequence of CNFs φ_n of polynomial size in n such that f_{φ_n} is NP-hard [23, Sec. 3]. Suppose there is a $\mathcal{QL}$-rewriting Φ_n of $(\mathcal{T}_\dagger, \bar{\boldsymbol{q}}_\varphi(x))$ of polynomial size. By adapting the proof of P $\subseteq$ P/poly [3, Theorem 6.6] to the algorithm that checks $\mathcal{A}_m^{\boldsymbol{\alpha}} \models \Phi_n(a)$, we obtain a sequence of polynomial-size circuits computing f_{φ_n}, from which NP $\subseteq$ P/poly. ❑

C.3 Theorem 21

THEOREM 21. *Evaluating PE-queries over trees in $\mathfrak{T}$ is* NP*-hard.*

More precisely, we are going to prove:

THEOREM 28. *The evaluation problem for PE-queries over data instances of the form $\mathcal{A}_m^{\boldsymbol{\alpha}}$ is* NP*-hard.*

PROOF. Let φ_k, $k \ge 1$, be the 3-CNF with all possible $m = O(k^3)$ clauses of k variables. Without loss of generality, we will assume that the number of clauses in φ_k is actually $m = 2^\ell$, for some ℓ. We construct a PE-query $\boldsymbol{q}_m(x)$ such that, for any $\boldsymbol{\alpha} \in \{0,1\}^m$, we have $\mathcal{A}_m^{\boldsymbol{\alpha}} \models \boldsymbol{q}_m(a)$ iff the CNF $\varphi_k^{-\boldsymbol{\alpha}}$ is satisfiable, and the size of $\boldsymbol{q}_m$ is polynomial in m (and k).

The query $\boldsymbol{q}_m(x)$ takes the form

$$\boldsymbol{q}_m(x) = \exists \boldsymbol{z}\, \big(\boldsymbol{r}(x, \boldsymbol{z}) \land \boldsymbol{s}(x, \boldsymbol{z}) \land \boldsymbol{t}(x, \boldsymbol{z})\big),$$

where the subqueries (without quantified variables) $\boldsymbol{r}$, $\boldsymbol{s}$ and $\boldsymbol{t}$ and the variables $\boldsymbol{z}$ are defined as follows. Among the variables $\boldsymbol{z}$, there are variables $z_1, \dots, z_m$ corresponding to the leaves of $\mathcal{A}_m^{\boldsymbol{\alpha}}$, variables $x_1, \dots, x_k$ corresponding to the propositional variables of φ_k, and variables $x_1', \dots, x_k'$ corresponding to their negations (there are other auxiliary variables which will be introduced later on).

Now we will describe the subqueries $\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t}$ of $\boldsymbol{q}_m$. The subquery $\boldsymbol{r}$ expresses that the variables $z_1, \dots, z_m$ indeed correspond to the clauses of φ_k; it takes the form $\boldsymbol{r} = \bigwedge_{i=1}^{m} \boldsymbol{r}_i$. Each $\boldsymbol{r}_i$ corresponds to a leaf of $\mathcal{A}_m^{\boldsymbol{\alpha}}$. Consider a path from the root a to this ith leaf. Let $P_1, \dots, P_\ell$ be the sequence of labels on the edges of this path, that is, each P_i is either P_- or P_+. Then

$$\boldsymbol{r}_i = P_1(x, y_i^1) \wedge P_2(y_i^1, y_i^2) \wedge \ldots \wedge P_\ell(y_i^{\ell-1}, z_i),$$

where $y_i^1, \dots, y_i^{\ell-1}$ are variables among $\boldsymbol{z}$.

The subquery $\boldsymbol{s}$ encodes that the variables $x_1 \dots, x_k$ and $x'_1, \dots, x'_k$ correspond to an arbitrary Boolean assignment. It is of the form $\boldsymbol{s} = \bigwedge_{i=1}^{k} \boldsymbol{s}_i$, and each $\boldsymbol{s}_i$ is the following:

$$\begin{aligned} &P_\pm(x, u_i^1) \wedge P_\pm(u_i^1, u_i^2) \wedge \cdots \wedge P_\pm(u_i^{\ell-2}, u_i^{\ell-1}) \wedge \\ &\quad \big[\big(P_\pm(u_i^{\ell-1}, x_i) \wedge P_\pm(x'_i, u_i^{\ell-1}) \wedge B_0(x_i)\big) \vee \\ &\qquad \big(P_\pm(u_i^{\ell-1}, x'_i) \wedge P_\pm(x_i, u_i^{\ell-1}) \wedge B_0(x'_i)\big) \big], \end{aligned}$$

where $u_i^1, \dots, u_i^{\ell-1}$ are variables among $\boldsymbol{z}$ and $P_\pm(x, y) = P_-(x, y) \vee P_+(x, y)$.

The last subquery $\boldsymbol{t}$ encodes that the assignment given by $x_1, \dots, x_k$ and $x'_1, \dots, x'_k$ satisfies the CNF given by $z_1, \dots, z_m$. The formula $\boldsymbol{t}$ has the following form: $\boldsymbol{t} = \bigwedge_{i=1}^{m} \boldsymbol{t}_i$. Suppose the clause z_i is a disjunction of literals $l_{i,1}, l_{i,2}$ and $l_{i,3}$, where each $l_{i,n}$ is among $x_1, \dots, x_k$ and $x'_1, \dots, x'_k$. Then

$$\boldsymbol{t}_i = B_0(z_i) \vee B_0(l_{i,1}) \vee B_0(l_{i,2}) \vee B_0(l_{i,3}).$$

It is easy to see that $\boldsymbol{q}_m$ is satisfiable over a given $\mathcal{A}_m^{\boldsymbol{\alpha}}$ iff $\mathcal{A}_m^{\boldsymbol{\alpha}}$ corresponds to a satisfiable 3-CNF $\varphi_k^{-\boldsymbol{\alpha}}$. Thus we have reduced the 3-SAT problem to the problem of evaluating $\boldsymbol{q}_m$ over $\mathcal{A}_m^{\boldsymbol{\alpha}}$. Since 3-SAT is NP-complete, we thus have shown NP-hardness of our query evaluation problem. ❑

C.4 Theorem 22

THEOREM 22. *There is an ontology $\mathcal{T}_\ddagger$ such that answering OMQs of the form $(\mathcal{T}_\ddagger, \boldsymbol{q})$ with Boolean linear CQs $\boldsymbol{q}$ is LOGCFL-hard for query complexity.*

PROOF. Our proof encodes the hardest LOGCFL language $\mathcal{L}$ [26] as formulated in [52]. The language $\mathcal{L}$ enjoys the following property: for every language $\mathcal{L}'$ over the alphabet Σ' in LOGCFL, there exists a logspace transducer τ converting words over Σ' to the words over the alphabet Σ of $\mathcal{L}$ in the sense that $w \in \mathcal{L}'$ iff $\tau(w) \in \mathcal{L}$. We construct an ontology $\mathcal{T}_\ddagger$ and a logspace transducer that converts the words $w \in \Sigma^*$ to linear Boolean CQs $\boldsymbol{q}_w$ such that

$$w \in \mathcal{L} \quad \text{iff} \quad \mathcal{T}_\ddagger, \{A(a)\} \models \boldsymbol{q}_w.$$

To explain the construction, we begin with a simpler context-free language. Let $\Sigma_0 = \{a_1, b_1, a_2, b_2\}$ be an alphabet and B_0 be the context-free language generated by the following grammar:

$$\begin{array}{ll} S \to SS, & S \to \epsilon, \\ S \to a_1 S b_1, & S \to a_2 S b_2. \end{array}$$

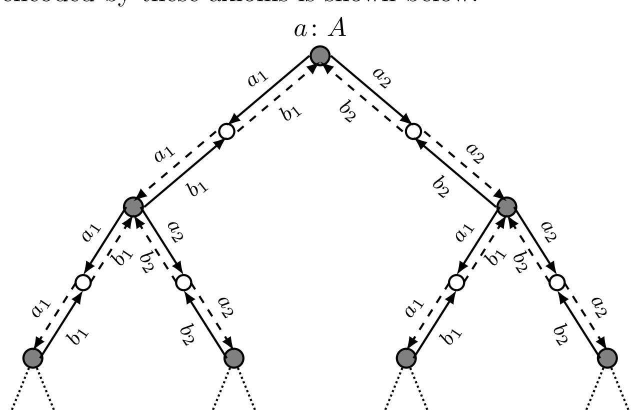
With each word $w = c_0 \dots c_n$ over Σ_0 we associate conjunction $\gamma_w(u_0, v_0, \dots, u_n, v_n, u_{n+1})$ of the following atoms:

$$\begin{aligned} R_{c_0}(u_0, v_0), S_{c_0}(v_0, u_1), R_{c_1}(u_1, v_1), S_{c_1}(v_1, u_2), \dots,& \\ R_{c_n}(u_n, v_n), S_{c_n}(v_n, u_{n+1}),& \end{aligned}$$

where R_c and S_c are binary predicates, for $c \in \Sigma_0$. Let $\mathcal{T}_0$ contain the following axioms, for $i = 1, 2$:

$$\begin{aligned} D(x) \to \exists y \,\big(R_{a_i}(x, y) \wedge S_{b_i}(y, x) \wedge& \\ \exists z \,\big(S_{a_i}(y, z) \wedge R_{b_i}(z, y) \wedge D(z)\big)\big).& \end{aligned} \tag{11}$$

An initial part of the canonical model of $(\mathcal{T}_0, \{A(a), D(a)\})$ encoded by these axioms is shown below:



(each large gray node belongs to D, each solid arrow with label c belongs to R_c and each dashed arrow with label c to S_c, for $c \in \Sigma_0$). Let $\boldsymbol{q}_w^A$ be the following linear Boolean CQ:

$$A(u_0) \wedge \gamma_w(u_0, v_0, \dots, u_n, v_n, u_{n+1}) \wedge A(u_{n+1}).$$

The following claim can readily be verified:

PROPOSITION 29. *For every $w \in \Sigma_0^*$, we have $w \in B_0$ iff $\mathcal{T}_0, \{A(a), D(a)\} \models \boldsymbol{q}_w^A$.*

The language B_0 is, however, not LOGCFL-hard. We now reproduce the definition of the hardest LOGCFL language $\mathcal{L}$ from [52], which uses B_0 as a basis of the construction. Let $\Sigma = \Sigma_0 \cup \{[,], \#\}$, for distinct symbols [,], and # not in Σ_0. Then set

$$\begin{aligned} \mathcal{L} = \big\{ &[x_1 y_1 z_1][x_2 y_2 z_2] \dots [x_k y_k z_k] \mid k \geq 1, \\ &x_i \in (\Sigma_0 \cup \{\#\})^* \{\#\} \cup \{\epsilon\} \text{ and} \\ &z_i \in \{\epsilon\} \cup \{\#\} (\Sigma_0 \cup \{\#\})^*, \text{ for all } i \leq k, \\ &\qquad\qquad \text{and } y_1 y_2 \dots y_k \in B_0 \big\}. \end{aligned}$$

To explain the intuition, following [52], let a string of symbols of the form $[w_1 \# w_2 \# \dots \# w_n]$, where $w_i \in \Sigma^*$ for all i, be called a *block* and let each of the substrings w_i be called a *choice*. Then, $\mathcal{L}$ is the set of all strings of blocks such that there exists a sequence of choices, one from each block, which is in the base language B_0. The reader should notice that a choice (possibly of the empty string) must be made from each block. For example,

$$[a_1 a_2 \# b_2 b_1] \notin \mathcal{L}, \tag{12}$$

$$[a_1a_2\#b_2b_1][b_2b_1] \in \mathcal{L}, \tag{13}$$

$$[a_1a_2\#b_2b_1][a_1b_1] \notin \mathcal{L}, \tag{14}$$

$$[\#a_1a_2\#b_2b_1][a_1b_1] \in \mathcal{L}. \tag{15}$$

We say that a word w over Σ is *block-formed* if the following conditions are satisfied:

– the word begins with [and ends with],

– after each [there is no [before];

– each non-final] is followed immediately by [;

– between each pair of matching [and] there is at least one symbol.

With these definitions at hand, we first describe a log-space transducer that, given a word w over Σ, returns a linear Boolean CQ $\boldsymbol{q}_w$ with binary predicates R_c and S_c, for $c \in \Sigma$, and unary predicates A and E. If the word $w = c_0 \dots c_n$ is block-formed, then $\boldsymbol{q}_w$ consists of the following atoms:

$$A(u_0) \wedge \gamma_w(u_0, v_0, \dots, u_n, v_n, u_{n+1}) \wedge A(u_{n+1}).$$

Otherwise, the transducer returns a query that consists of a prefix of $A(u_0) \wedge \gamma_w(u_0, v_0, \dots, u_n, v_n, u_{n+1})$ and ends in $E(u_i)$, for some i, which will indicate an error (as all queries containing E will be false in $\mathcal{T}_\ddagger, \{A(a)\}$). It is straightforward to verify that the required transducer can be implemented in L.

Let $\mathcal{T}_\ddagger$ contain the two axioms (11) and the following axioms:

$$A(x) \to D(x), \tag{16}$$

$$D(x) \to \exists y \big(R_{[}(x,y) \wedge S_{[}(y,x)\big), \tag{17}$$

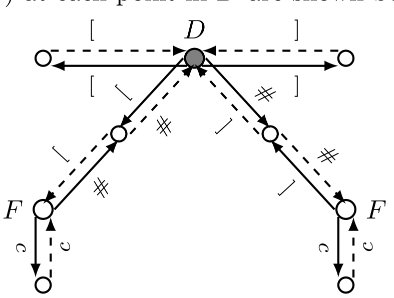
$$D(x) \to \exists y \big(R_{[}(x,y) \wedge S_{\#}(y,x) \wedge \exists z \, \big(S_{[}(y,z) \wedge R_{\#}(z,y) \wedge F(z)\big)\big), \tag{18}$$

$$D(x) \to \exists y \big(R_{]}(x,y) \wedge S_{]}(y,x)\big), \tag{19}$$

$$D(x) \to \exists y \big(R_{\#}(x,y) \wedge S_{]}(y,x) \wedge \exists z \, \big(S_{\#}(y,z) \wedge R_{]}(z,y) \wedge F(z)\big)\big), \tag{20}$$

$$F(x) \to \exists y \, \big(R_c(x,y) \wedge S_c(y,x)\big), \quad \text{for } c \in \Sigma_0 \cup \{\#\}. \tag{21}$$

The four additional branches of the canonical model of $(\mathcal{T}_\ddagger, \{A(a)\})$ at each point in D are shown below:



(the labels D and F are indicated next to the nodes, and, as before, each solid arrow with label c belongs to R_c and each dashed arrow with label c to S_c, for $c \in \Sigma_0$; to avoid clutter, only one pair of c-arrows is shown at the bottom).

Let $\boldsymbol{q}_w^D$ be defined identically to $\boldsymbol{q}_w^A$ except that the two occurrences of A are replaced by D. The following property is established similarly to Proposition 29:

Proposition 30. *For any block-formed word* $w] \in \Sigma^*$,

$$w = [x, \text{ for } x \in (\Sigma_0 \cup \{\#\})^* \{\#\} \cup \{\epsilon\}, \quad \textit{iff} \quad \{(17), (18), (21)\}, \{D(d)\} \models \boldsymbol{q}_w^D.$$

For any block-formed word $[w \in \Sigma^*$,

$$w = z], \text{ for } z \in \{\epsilon\} \cup \{\#\}(\Sigma_0 \cup \{\#\})^*, \quad \textit{iff} \quad \{(19), (20), (21)\}, \{D(d)\} \models \boldsymbol{q}_w^D.$$

With these properties established, it can readily be verified that $\mathcal{T}_\ddagger, \{A(a)\} \models \boldsymbol{q}_w$ iff $w \in \mathcal{L}$. Consider a block-formed word $w \in \Sigma^*$. Let $[w_1\#w_2\# \dots \#w_n]$ be its m-th block and $w_j = y_m$ (that is, w_j is the segment of the B_0-word in this block). By Proposition 30, the subtree generated by (18) matches the (translation of) $[w_1\# \dots \#w_{j-1}\#$, whereas the subtree generated by (20) matches $\#w_{j+1}\# \dots \#w_n]$. By Proposition 29, the w_j itself is mapped into the main tree generated by (11). Note that (17) and (19) are needed for the case when $j = 1$ and $j = n$, respectively. Finally, observe that (the translation of) w has to be mapped starting from a (the root of the tree) and ending at a, and that the tree of the canonical model does not contain concept E, so only a block-formed w can be mapped to the canonical model. In particular, $\mathcal{T}_\ddagger, \{A(a)\} \not\models \boldsymbol{q}_w$ for w of (12) and (14), and $\mathcal{T}_\ddagger, \{A(a)\} \models \boldsymbol{q}_w$ for w of (13) and (15). ❑

D. EXPERIMENTS

D.1 Computing rewritings

We computed 6 types of rewritings for linear queries similar to those in Example 8 and a fixed ontology from Example 11. The first three rewritings were obtained by running executables of Rapid [13], Clipper [18] and Presto [49] with a 15 minute timeout on a desktop machine. The other three rewritings are rewritings Lin, Log and Tw described in Sections 3.3, 3.2 and 3.4 respectively.

We considered the following three sequences:

$$RRSRSRSRRSRRSSR, \quad \text{(Sequence 1)}$$

$$SRRRRRSRSRRRRRR, \quad \text{(Sequence 2)}$$

$$SRRSSRSRSRRSRRS. \quad \text{(Sequence 3)}$$

For each of the three sequences, we consider the line-shaped queries with 1–15 atoms formed by their prefixes. Table 1 presents the sizes of the different types of rewritings.

Table 1: The size (number of clauses) of different types of rewritings for the three sequences of queries (– indicates timeout after 15 minutes)

no. of atoms	Sequence 1 *RRSRSRSRRSRRSSR*						Sequence 2 *SRRRRRSRSRRRRRR*						Sequence 3 *SRRSSRSRSRRSRRS*					
	Rapid	Clipper	Presto	Lin	Log	Tw	Rapid	Clipper	Presto	Lin	Log	Tw	Rapid	Clipper	Presto	Lin	Log	Tw
1	1	1	5	2	1	1	1	1	5	2	1	1	1	1	5	2	1	1
2	1	1	5	5	2	0	2	2	14	5	4	2	2	2	14	5	4	2
3	2	2	14	8	5	3	2	2	14	8	5	3	2	2	14	8	5	3
4	3	3	19	11	8	4	2	2	14	11	6	3	4	4	23	11	8	5
5	5	5	24	14	12	6	2	2	14	14	8	4	4	4	23	14	10	6
6	7	7	33	17	16	10	2	2	14	17	10	4	8	8	39	17	15	7
7	10	11	49	20	20	10	4	4	23	20	13	7	11	11	57	20	18	14
8	13	16	77	23	24	14	6	7	29	23	16	7	18	24	96	23	21	8
9	13	16	77	26	27	15	10	13	50	26	22	10	24	35	183	26	27	10
10	26	44	203	29	32	16	14	26	83	29	27	11	34	63	356	29	33	17
11	39	72	329	32	36	16	14	26	83	32	29	14	43	100	356	32	37	20
12	39	126	329	35	40	21	14	26	83	35	33	18	56	302	1028	35	42	23
13	–	241	959	38	45	24	–	30	83	38	35	20	–	–	1712	38	46	25
14	–	–	959	41	47	25	–	31	83	41	36	16	–	–	1712	41	51	27
15	–	–	2723	44	51	22	–	30	83	44	37	15	–	–	5108	44	52	29

Table 2: Generated datasets

dataset	V	p	q	avg. degree of vertices	no. of atoms
1.ttl	1 000	0.050	0.050	50	61 498
2.ttl	5 000	0.002	0.004	10	64 157
3.ttl	10 000	0.002	0.004	20	256 804
4.ttl	20 000	0.002	0.010	40	1 027 028

D.2 Datasets

We used Erdös-Rènyi random graphs with independent parameters V (number of vertices), p (probability of an R-edge) and q (probability of concepts A and B at a given vertex). Note that we intentionally did not introduce any S-edges. The last parameter, the average degree of a vertex, is $V \cdot p$. Table 2 summarises the parameters of the datasets.

D.3 Evaluating rewritings

We evaluated all obtained rewritings on the datasets in Section D.2 using RDFox triplestore [42] with 999-second timeout. The materialisation time and other relevant statistics are given in Tables 3, 4, and 5.

D.4 Discussion

Note that the three types of rewritings suggested in this paper give rise to three different rewriting strategies for linear queries. Let us compare how the execution time depends on the exact rewriting strategy. We see in Table 3 that for most queries in Sequence 1 the Lin rewriting shows the best performance, while for Sequences 2 and 3 algorithms Log and Tw* are the winners (Tables 4 and 5). Note also that even within a single sequence the results may vary with the number of atoms.

All three rewriting algorithms are based upon a common idea: given a query, pick a point (or a set of points) that would split the query into subqueries, then rewrite these subqueries recursively, and then include rules that join the results into the rewriting of the initial query. However, there is a liberty in the choice of this point, and our rewritings are essentially different in this strategy. Thus, different rewritings generate NDL programs which are related to each other like different execution plans for CQs. Taking into account that we use highly unbalanced data (empty S versus dense R) and that RDFox just materialises all of the predicates of the program without using magic sets or optimising the program before executions, the performance naturally depends on how we split the query into subqueries in the rewriting algorithm.

In the paper, we described three simple complexity-motivated splitting strategies. Our experiments show that none of them is always the best and the execution time may be dramatically improved by using an 'adaptable' splitting strategy which would work similarly to a query execution planner in database management systems and use statistical information about the data to generate a quickly executable NDL program.

The difference in performance between different types of optimal rewritings made us investigate its causes. For example, we noticed that the Tw-rewriting of the query with 3 atoms of Sequence 3

$$G(x, y) \leftarrow S(x, z) \wedge P_{13}(z, y),$$
$$P_{13}(x, y) \leftarrow R(x, z) \wedge R(z, y),$$

Table 3: Evaluating rewritings on RDFox - 1

data-set	query size	evaluation time (sec)							no. of answers	no. of generated tuples						
		Rapid	Clipper	Presto	Lin	Log	Tw	Tw*		Rapid	Clipper	Presto	Lin	Log	Tw	Tw*
	1	0.021	0.019	0.034	0.049	0.017	0.016	0.01	61390	61390	61390	122780	61449	61390	61390	61390
	2	0.675	0.694	0.706	0.898	0.505	0.652	0.698	976789	976789	976789	1038179	1041822	1038179	976789	976789
	3	0.058	0.053	0.125	0.013	0.112	0.01	0.012	2956	2956	2956	64394	3054	64394	3004	3004
	4	0.204	0.201	0.314	0.087	0.675	0.76	0.12	212213	212213	212213	273710	283409	1314797	1189061	212272
	5	0.12	0.114	0.314	0.014	0.576	0.696	0.064	2956	2956	2956	64453	3150	1105636	976837	3004
	6	0.266	0.248	0.685	0.093	0.266	0.768	0.124	212213	212213	212213	273710	292815	337479	1198455	218710
	7	0.271	0.242	1.11	0.008	0.243	0.687	0.05	2 956	2 956	2 956	64453	3 246	125 361	982797	3148
	8	0.412	0.377	1.406	0.084	0.904	0.944	0.186	212 213	212 213	212 213	273710	302 221	1 659 409	1410727	431100
	9	3.117	3.337	12.713	3.376	2.941	2.405	1.633	998 945	998 945	998 945	1060442	2 927 979	2 684 359	2435551	1455913
1.ttl	10	1.079	1.102	18.432	0.012	0.607	0.76	0.166	8 374	8 374	10 760	69871	12 573	1 178 714	1203649	224057
	11	2.246	1.984	48.311	0.385	0.945	1.075	0.371	436 000	436 000	436 000	497497	836 876	1 618 743	1663534	664174
	12	13.693	30.032	>999	8.129	6.867	5.922	5.28	999 998	999 998	1 000 000	–	5 311 314	4 439 352	3217262	2241208
	13	–	6.810	560.206	0.027	0.616	0.946	0.274	20 985	–	24 839	82482	38 200	553 821	1234421	254888
	14	–	–	913.387	0.013	0.358	0.819	0.27	0	–	–	61497	48	312 723	1201459	228307
	15	–	–	>999	0.032	0.394	0.994	0.33	2 000	–	–	–	70 277	376 313	1417786	442579
	1	0.02	0.022	0.039	0.02	0.019	0.017	0.008	64103	64103	64103	128206	64125	64103	64103	64103
	2	0.273	0.305	0.321	0.29	0.297	0.275	0.466	809731	809731	809731	873834	874112	873834	809731	809731
	3	0.03	0.028	0.06	0.011	0.058	0.01	0.013	427	427	427	64561	489	64561	458	458
	4	0.057	0.054	0.103	0.032	0.448	0.315	0.035	8778	8778	8778	72934	74004	947164	818531	8800
	5	0.05	0.046	0.128	0.014	0.423	0.301	0.03	427	427	427	64583	551	938875	809762	458
	6	0.08	0.074	0.27	0.035	0.084	0.316	0.038	8778	8778	8778	72934	75103	77253	819648	9490
	7	0.089	0.080	0.378	0.008	0.078	0.295	0.024	427	427	427	64583	613	68 546	810647	551
	8	0.136	0.125	0.467	0.029	0.434	0.322	0.037	8 778	8 778	8 778	72934	76 202	1 085 362	828448	18334
	9	0.202	0.254	1.179	0.369	0.554	0.391	0.102	105 853	105 853	105 853	170009	1 020 363	1 190 249	933295	123190
2.ttl	10	0.174	0.204	2.341	0.011	0.461	0.321	0.052	11	11	438	64167	506	943 097	819428	9354
	11	0.192	0.259	4.726	0.036	0.473	0.336	0.053	651	651	9 396	64807	74 922	944 210	820354	11271
	12	0.244	0.699	24.778	0.396	1.034	0.509	0.15	8 058	8 058	113 179	72214	1 004 735	1 940 300	934269	124420
	13	–	0.629	20.555	0.015	0.244	0.458	0.084	0	–	438	64156	502	209 915	820373	10321
	14	–	–	25.243	0.014	0.153	0.350	0.081	0	–	–	64156	31	200 962	820106	10722
	15	–	–	66.916	0.032	0.172	0.335	0.072	0	–	–	64156	64 543	265 087	828884	19522
	1	0.131	0.094	0.225	0.101	0.096	0.14	0.032	256699	256699	256699	513398	256756	256699	256699	256699
	2	2.933	2.946	3.017	2.955	3.053	2.929	3.039	6379932	6379932	6379932	6636631	6638150	6636631	6379932	6379932
	3	0.206	0.175	0.519	0.03	0.499	0.029	0.034	1217	1217	1217	257963	1311	257963	1264	1264
	4	0.399	0.424	0.927	0.171	4.003	3.419	0.231	67022	67022	67022	323825	327716	6961626	6447011	67079
	5	0.36	0.357	1.112	0.036	4.133	3.396	0.179	1217	1217	1217	258020	1405	6895915	6379979	1264
	6	0.632	0.57	1.806	0.169	0.836	3.425	0.228	67022	67022	67022	323825	331647	363640	6450931	69782
	7	0.631	0.581	2.981	0.035	0.756	3.255	0.156	1 217	1 217	1 217	258020	1 499	296 711	6382460	1405
	8	0.925	0.876	3.739	0.159	4.377	3.405	0.278	67 022	67 022	67 022	323825	335 578	7 546 184	6518010	136975
	9	1.949	2.275	14.564	4.063	5.251	4.169	1.169	1 678 668	1 678 668	1 678 668	1935471	8 613 829	9 225 201	8196944	1815899
3.ttl	10	1.24	1.377	35.109	0.049	4.731	3.571	0.342	60	60	1 277	256863	1 389	6 936 178	6449555	68557
	11	1.403	1.798	60.858	0.249	4.846	3.607	0.343	11 498	11 498	77 811	268301	341 459	6 949 160	6462905	85267
	12	1.697	5.413	572.53	4.355	10.128	6.693	1.645	305 640	305 640	1 951 654	562443	8 780 232	15 626 926	8438115	2058532
	13	–	4.382	484.969	0.082	1.762	4.926	0.599	0	–	1 277	256803	1 377	917 117	6453717	72776
	14	–	–	575.487	0.063	1.115	3.972	0.584	0	–	–	256803	47	850 309	6452195	73900
	15	–	–	>999	0.177	1.011	3.585	0.501	0	–	–	–	257 974	1 107 065	6519217	140979
	1	0.433	0.451	1.037	0.495	0.439	0.456	0.165	1026526	1026526	1026526	2053052	1026774	1026526	1026526	1026526
	2	27.549	28.088	28.329	27.011	29.532	32.331	31.34	49364886	49364886	49364886	50391412	50404311	50391412	49364886	49364886
	3	2.067	2.409	3.657	0.159	4.087	0.161	0.162	13103	13103	13103	1039882	13613	1039882	13356	13356
	4	4.866	5.438	9.511	1.37	38.919	31.188	2.746	1286991	1286991	1286991	2314018	2353661	52718280	50652125	1287239
	5	4.061	4.032	10.374	0.209	42.943	33.064	2.142	13103	13103	13103	1040130	14119	51444898	49365139	13356
	6	6.909	7.133	16.249	1.443	7.767	36.268	2.782	1286991	1286991	1286991	2314018	2393145	2952225	50691250	1313261
	7	6.614	6.277	23.7	0.243	8.586	29.098	2.02	13 103	13 103	13 103	1040130	14 625	1 665 376	49391598	14115
	8	11.441	10.923	29.1	1.880	54.813	29.426	3.669	1 286 991	1 286 991	1 286 991	2314018	2 432 629	56 098 445	51978489	2600996
	9	46.704	50.668	193	76.169	102.055	66.464	33.63	58 753 514	58 753 514	58 753 514	59780541	114 973 160	114 837 395	110717131	61339643
4.ttl	10	14.348	15.503	462	0.375	43.347	30.008	4.694	19 966	19 966	33 014	1046993	35 359	52 103 362	50698955	1321716
	11	19.593	20.907	821	2.843	44.410	31.061	5.319	1 872 159	1 872 159	3 051 184	2899186	4 397 556	53 986 724	52602849	3224788
	12	71.354	182.499	>999	172.822	237.478	179.12	90.04	79 939 048	79 939 048	120 229 590	–	199 083 489	242 500 074	189429768	140064931
	13	–	54.497	>999	0.562	22.345	44.427	7.105	22 474	–	53 717	–	58 826	5 686 759	50759705	1382714
	14	–	–	>999	0.550	12.462	36.259	7.493	0	–	–	–	253	4 356 739	50704606	1353393
	15	–	–	>999	1.211	11.315	30.709	7.028	12 165	–	–	–	1 064 542	5 395 902	52014512	2652797

Table 4: Evaluating rewritings on RDFox - 2

data-set	query size	evaluation time (sec)							no. of answers	no. of generated tuples						
		Rapid	Clipper	Presto	Lin	Log	Tw	Tw*		Rapid	Clipper	Presto	Lin	Log	Tw	Tw*
	1	0.009	0.005	0.005	0.005	0.005	0.005	0.007	0	0	0	0	48	0	0	0
	2	0.009	0.008	0.021	0.05	0.012	0.008	0.007	59	59	59	61508	64406	118	59	59
	3	0.083	0.058	0.077	0.9	0.093	0.732	0.058	3584	3584	3584	65033	1092161	65033	980373	3584
	4	2.363	4.049	2.301	8.32	0.11	0.723	0.073	57571	57571	57571	119020	2204964	119079	1034419	57630
	5	97	92	102	13.599	2	14.272	2.718	59000	59000	59000	120449	3265393	1097297	2035848	59059
	6	>999	>999	>999	17.882	19	13.881	42.914	59000	–	–	–	4324393	1162212	2039373	62584
	7	129	122	>999	0.384	0.25	0.749	0.344	2832	2832	2832	–	156824	132259	1030122	6464
	8	>999	>999	>999	10.963	2	1.82	21.399	55991	–	–	–	3352724	304347	1302623	268322
	9	162	158	>999	0.395	0.21	0.722	0.344	2832	2832	2832	–	156920	187155	1040255	5895
1.ttl	10	>999	>999	>999	11.118	2	12.928	39.21	55991	–	–	–	3362130	1220806	2104667	68937
	11	>999	>999	>999	20.217	4	14.611	>999	59000	–	–	–	5920653	2251570	2342243	–
	12	>999	>999	>999	31.648	21	19.079	>999	59000	–	–	–	8714382	3361965	4165789	–
	13	–	>999	>999	34.395	46	193.512	>999	59000	–	–	–	9783393	3429574	4198870	–
	14	–	>999	>999	39.818	223	190.334	>999	59000	–	–	–	10842393	1509563	4130571	–
	15	–	>999	>999	49.391	232	226.827	>999	59000	–	–	–	11901393	1594164	4420495	–
	1	0.007	0.007	0.005	0.007	0.007	0.005	0.004	0	0	0	0	31	0	0	0
	2	0.01	0.01	0.028	0.027	0.011	0.008	0.008	22	22	22	64147	64543	44	22	22
	3	0.025	0.025	0.041	0.345	0.046	0.313	0.024	256	256	256	64381	879372	64381	809987	256
	4	0.135	0.136	0.169	4.798	0.055	0.297	0.023	3300	3300	3300	67425	9329702	67447	813053	3322
	5	1.314	1.278	1.824	39.195	0.513	4.714	0.122	34474	34474	34474	98599	33935400	908352	9240858	34496
	6	13.597	13.652	19.52	119.212	0.698	4.606	0.178	106742	106742	106742	170867	59117304	1044957	9313360	106998
	7	1.396	1.34	18.91	0.116	0.102	0.326	0.028	248	248	248	64404	214761	129190	815625	535
	8	1.572	1.987	20.58	2.518	0.095	0.364	0.069	3478	3478	3478	67634	2968573	199843	825309	12300
	9	1.397	1.554	35.15	0.118	0.076	0.333	0.033	248	248	248	64404	214823	132187	813759	728
2.ttl	10	1.636	2.634	233	2.591	0.639	4.45	0.069	3478	3478	3478	67634	2969672	976875	9245685	4871
	11	1.677	12.024	895	30.575	0.98	4.434	0.66	35382	35382	35382	99538	26328037	1823608	9285127	44313
	12	2.009	143	>999	128.532	1.756	5.666	7.999	106895	106895	106895	–	71017728	2184441	10358119	1010563
	13	–	>999	>999	243.656	2.559	47.098	5.121	110000	–	–	–	115653199	2742932	34483363	145486
	14	–	>999	>999	325.755	2.866	50.997	12.028	110000	–	–	–	151038934	1448087	35282112	111224
	15	–	>999	>999	433.438	26.903	54.518	133.512	110000	–	–	–	176515562	9102348	35442252	118515
	1	0.009	0.01	0.009	0.011	0.009	0.011	0.009	0	0	0	0	47	0	0	0
	2	0.023	0.02	0.115	0.145	0.022	0.019	0.019	57	57	57	256813	257974	114	57	57
	3	0.123	0.127	0.249	3.364	0.315	3.212	0.136	1462	1462	1462	258218	6668549	258218	6381394	1462
	4	1.992	1.93	3.072	85.844	0.345	3.21	0.122	36260	36260	36260	293016	86686553	293073	6416249	36317
	5	47	56	76.8	967	7.09	70.117	1.898	452502	452502	452502	709258	187656175	7089247	86439255	452559
	6	>999	>999	>999	>999	9.996	73.99	3.965	570000	-	–	–	–	7464849	86558158	571462
	7	47	51	>999	1.591	0.736	3.47	0.181	2125	2125	2125	–	883690	518306	6413768	3634
	8	77	99	>999	60.365	0.667	3.601	1.842	53191	53191	53191	–	22657990	862422	6536462	120327
	9	50	56	>999	1.885	0.473	3.496	0.223	2125	2125	2125	–	883784	553583	6419638	3446
3.ttl	10	79	142	>999	59.019	7.999	67.145	2.083	53191	53191	53191	–	22661921	7401781	86497805	58664
	11	81	>999	>999	>999	10.862	68.956	50.812	516631	516631	–	–	–	14275796	87027128	587987
	12	116	>999	>999	>999	26.218	112.098	306.304	570000	570000	–	–	–	16280643	95308112	8298971
	13	–	>999	>999	>999	45.19	>999	785.247	570000	–	–	–	–	27255415	–	1026838
	14	–	>999	>999	>999	74.691	>999	>999	570000	–	–	–	–	9092721	–	–
	15	–	>999	>999	>999	>999	>999	>999	–	–	–	–	–	–	–	–
	1	0.026	0.027	0.027	0.035	0.026	0.047	0.029	0	0	0	0	253	0	0	0
	2	0.068	0.067	0.5	0.543	0.078	0.069	0.07	248	248	248	1027022	1040241	496	248	248
	3	0.992	0.99	1.483	33.62	1.98	30.768	0.976	12651	12651	12651	1039425	51050537	1039425	49377537	12651
	4	60.836	69.126	65.671	M	2.175	30.532	1.272	609193	609193	609193	1635967	–	1636215	49974327	609441
	5	>999	>999	>999	>999	85	>999	60.335	4947136	–	–	–	–	55339044	–	4947384
	6	>999	>999	>999	>999	287	>999	261.562	4960000	–	–	–	–	56390837	–	4972651
	7	>999	>999	>999	63	5	31.839	3.118	62572	–	–	–	10949093	2141879	50070886	75476
	8	>999	>999	>999	>999	13	37.121	273.336	2435666	–	–	–	–	6151203	53696984	3723153
	9	>999	>999	>999	61	5	31.899	5.725	62572	–	–	–	10949599	2739031	50050255	76176
4.ttl	10	>999	>999	>999	>999	131	>999	319.902	2435666	–	–	–	–	58829172	–	2487953
	11	>999	>999	>999	M	214	>999	–	4960000	–	–	–	–	111363802	–	–
	12	>999	>999	>999	M	>999	>999	–	–	–	–	–	–	–	–	–
	13	–	>999	>999	M	>999	>999	–	–	–	–	–	–	–	–	–
	14	–	>999	>999	M	>999	>999	–	–	–	–	–	–	–	–	–
	15	–	>999	>999	M	>999	>999	–	–	–	–	–	–	–	–	–

Table 5: Evaluating rewritings on RDFox - 3

data-set	query size	evaluation time (sec)							no. of answers	no. of generated tuples						
		Rapid	Clipper	Presto	Lin	Log	Tw	Tw*		Rapid	Clipper	Presto	Lin	Log	Tw	Tw*
	1	0.004	0.003	0.003	0.004	0.003	0.021	0.003	0	0	0	0	48	0	0	0
	2	0.006	0.006	0.017	0.022	0.008	0.014	0.005	59	59	59	61508	64406	118	59	59
	3	0.053	0.06	0.065	0.849	0.087	0.69	0.053	3584	3584	3584	65033	1092161	65033	980373	3584
	4	0.012	0.01	0.074	0.01	0.009	0.008	0.008	2	2	2	61499	3176	168	109	109
	5	0.011	0.009	0.07	0.008	0.009	0.008	0.009	0	0	0	61497	48	166	59	59
	6	0.018	0.015	0.139	0.023	0.09	0.677	0.055	2	2	2	61499	64560	65203	980434	3704
	7	0.017	0.015	0.145	0.01	0.087	0.68	0.057	0	0	0	61497	144	65190	980480	3691
1.ttl	8	0.025	0.034	0.339	0.026	0.044	0.009	0.008	2	2	135	61499	73966	129565	170	286
	9	0.025	0.034	0.433	0.01	0.034	0.009	0.008	0	0	2	61497	240	65530	109	214
	10	0.035	0.086	0.549	0.029	0.026	0.015	0.015	2	2	135	61499	83372	67690	12950	13114
	11	0.034	0.086	4.445	1.164	0.54	0.765	0.221	133	0	2	61630	1684864	1095576	1227962	251278
	12	0.048	0.223	4.877	0.013	0.137	0.699	0.062	2	2	135	61499	4082	192211	983694	4115
	13	-	-	13.007	0.141	0.153	0.79	0.175	133	-	-	61630	380205	226874	1228297	229396
	14	-	-	382.922	3.738	0.878	1.166	0.318	1967	-	-	63464	3842746	1282299	1809813	270081
	15	-	-	307.184	0.017	0.36	0.771	0.224	11	-	-	61508	16542	242156	1228610	252720
	1	0.004	0.004	0.004	0.004	0.004	0.004	0.003	0	0	0	0	31	0	0	0
	2	0.006	0.006	0.02	0.023	0.009	0.006	0.006	22	22	22	64147	64543	44	22	22
	3	0.022	0.019	0.04	0.339	0.045	0.29	0.019	256	256	256	64381	879372	64381	809987	256
	4	0.013	0.011	0.047	0.01	0.01	0.009	0.009	0	0	0	64156	490	75	53	53
	5	0.012	0.011	0.044	0.008	0.009	0.01	0.009	0	0	0	64156	31	75	22	22
	6	0.02	0.016	0.081	0.027	0.042	0.304	0.021	0	0	0	64156	64543	64456	810009	300
2.ttl	7	0.018	0.015	0.094	0.011	0.041	0.297	0.024	0	0	0	64156	93	64465	810040	309
	8	0.025	0.036	0.182	0.027	0.053	0.01	0.009	0	0	0	64156	65642	129037	75	119
	9	0.026	0.037	0.215	0.012	0.03	0.009	0.009	0	0	0	64156	155	64611	53	106
	10	0.038	0.091	0.327	0.028	0.029	0.014	0.013	0	0	0	64156	66741	65120	1393	1468
	11	0.036	0.09	1.467	0.345	0.358	0.314	0.055	0	0	0	64156	906286	879949	818896	9218
	12	0.052	0.268	1.868	0.014	0.106	0.294	0.03	0	0	0	64156	494	193096	810468	385
	13	-	-	4.579	0.032	0.119	0.359	0.051	0	-	-	64156	74216	193944	819532	10495
	14	-	-	26.213	0.38	0.454	0.37	0.123	0	-	-	64156	995998	1008466	819319	9523
	15	-	-	26.689	0.017	0.209	0.352	0.063	0	-	-	64156	502	198540	819067	9293
	1	0.009	0.009	0.009	0.01	0.007	0.008	0.008	0	0	0	0	47	0	0 0	0
	2	0.019	0.017	0.104	0.111	0.02	0.017	0.016	57	57	57	256813	257974	114	57	57
	3	0.11	0.135	0.233	3.244	0.274	3.109	0.113	1462	1462	1462	258218	6668549	258218	6381394	1462
	4	0.038	0.034	0.277	0.034	0.026	0.027	0.028	0	0	0	256803	1314	161	104	104
	5	0.036	0.036	0.275	0.025	0.024	0.031	0.027	0	0	0	256803	47	161	57	57
	6	0.063	0.056	0.663	0.128	0.298	3.122	0.133	0	0	0	256803	257974	258379	6381451	1576
3.ttl	7	0.061	0.062	0.709	0.032	0.287	3.101	0.132	0	0	0	256803	141	258369	6381498	1566
	8	0.094	0.153	1.425	0.138	0.297	0.03	0.027	0	0	0	256803	261905	516433	161	275
	9	0.098	0.15	1.819	0.037	0.156	0.03	0.027	0	0	0	256803	235	258660	104	208
	10	0.143	0.399	2.478	0.15	0.148	0.049	0.048	0	0	0	256803	265836	259504	5473	5634
	11	0.141	0.368	12.374	3.343	3.315	3.397	0.384	0	0	0	256803	6866425	6670079	6452693	72865
	12	0.21	1.136	15.915	0.051	0.576	3.133	0.171	0	0	0	256803	1326	773580	6382718	1730
	13	-	-	35.05	0.194	0.623	3.521	0.341	0	-	-	256803	329484	776449	6451652	74135
	14	-	-	399.257	3.948	3.982	3.344	0.558	0	-	-	256803	8461907	7190771	6463879	74581
	15	-	-	388.289	0.06	1.34	3.213	0.378	0	-	-	256803	1377	803755	6452448	73026
	1	0.026	0.025	0.025	0.039	0.025	0.024	0.025	0	0	0	0	253	0	0	0
	2	0.064	0.069	0.471	0.522	0.064	0.07	0.064	248	248	248	1027022	1040241	496	248	248
	3	0.929	0.938	1.404	28.325	1.857	28.103	0.945	12651	12651	12651	1039425	51050537	1039425	49377537	12651
	4	0.198	0.173	1.617	0.157	0.095	0.129	0.135	4	4	4	1027031	13800	753	505	505
	5	0.182	0.174	1.617	0.144	0.094	0.143	0.138	0	0	0	1027027	253	749	248	248
	6	0.327	0.312	4.729	0.64	1.913	28.148	1	4	4	4	1027031	1040479	1040182	49377789	13151
4.ttl	7	0.308	0.325	4.721	0.222	1.98	27.908	1.106	0	0	0	1027027	759	1040183	49378038	13152
	8	0.504	0.778	9.217	0.675	1.278	0.158	0.129	4	4	236	1027031	1079963	2080575	757	1249
	9	0.522	0.835	12.456	0.266	0.705	0.14	0.131	0	0	4	1027027	1265	1041493	505	1002
	10	0.782	2.174	15.698	0.738	0.66	0.288	0.253	4	4	236	1027031	1119447	1055223	52295	53040
	11	0.76	2.077	93.286	30.477	30.641	29.507	3.476	232	0	4	1027259	54927712	51065747	50689528	1325139
	12	1.083	6.03	114.063	0.354	3.362	28.046	1.329	4	4	236	1027031	15222	3107857	49391554	14314
	13	-	-	253.131	1.64	3.442	30.217	3.913	232	-	-	1027259	2499217	3173640	50730474	1353321
	14	-	-	>999	74.607	35.483	30.531	5.52	10972	-	-	-	117902759	53931133	52556376	1368984
	15	-	-	>999	0.454	10.929	29.497	3.763	1	-	-	-	35953	3754770	50690218	1326126

$$G(x, y) \leftarrow A_P(x) \wedge R(x, y)$$

takes as long as 28 seconds to execute on the fourth dataset because it needs so much time to materialise P_{13}, which has around $6 \cdot 10^6$ triples. On the other hand, if we remove this predicate by substituting its definition into the first rule, we obtain the rewriting

$$G(x, y) \leftarrow S(x, z) \wedge R(x, v) \wedge R(v, y),$$
$$G(x, y) \leftarrow A_P(x) \wedge R(x, y),$$

which is executed in 0.945 seconds. This substitution could be done automatically by a clever NDL engine, but not performed by RDFox. Thus, we made an attempt to 'improve' the Tw-rewriting by getting rid in this fashion of all predicates that are defined by a single rule and occur not more than twice in the bodies of the rules. However, though the rewriting Tw* thus obtained shows a much better performance on Sequences 1 and 3 (see Tables 3 and 5), it is not always so on Sequence 2 (Table 4). This observation suggests that our rewriting could be executed faster on a more advanced NDL engine than RDFox which would carry out such substitutions depending on the cardinality of EDBs.