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

Meghyn Bienvenu
CNRS & University of Montpellier, France
meghyn@lirmm.fr

Stanislav Kikot
Birkbeck University of London, UK
staskikotx@gmail.com

Roman Kontchakov
Birkbeck University of London, UK
roman@dcs.bbk.ac.uk

Vladimir V. Podolskii
Steklov Mathematical Institute & National Research University Higher School of Economics, Moscow, Russia
podolskii@mi.ras.ru

Vladislav Ryzhikov
Free University of Bozen-Bolzano, Italy
ryzhikov@inf.unibz.it

Michael Zakharyaschev
Birkbeck University of London, UK
michael@dcs.bbk.ac.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, more generally, 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, and even in NL if their CQs are tree-shaped with a bounded number of leaves. Thus, such OMQs incur no overhead in complexity-theoretic terms. For OMQs with arbitrary ontologies and bounded-leaf tree-shaped CQs, NDL-rewritings are constructed and evaluated in LOGCFL. We experimentally demonstrate feasibility and scalability of our rewritings compared to previously proposed 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 CQs and LOGCFL-complete for bounded-leaf CQs.

Keywords

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

1. INTRODUCTION

The main aim of ontology-based data access (OBDA, for short) [50, 43] 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 schemas of data sources by a mapping $\mathcal{M}$. The instance $\mathcal{M}(\mathcal{D})$ that can be 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 [9], data integration [20], and data exchange [2].

In the classical approach to OBDA [13, 50], 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 every dataset $\mathcal{D}$ and mapping $\mathcal{M}$,

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

where $\mathcal{I}_{\mathcal{M}(\mathcal{D})}$ is the FO-structure comprised of the atoms in $\mathcal{M}(\mathcal{D})$; note that the rewriting is interpreted in $\mathcal{I}_{\mathcal{M}(\mathcal{D})}$ under the closed-world semantics. When the form of $\mathcal{M}$ is appropriately restricted, e.g., to GAV mappings, in which case the ontology predicates are defined as views over the data sources, 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 [13] 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) [10, 11], and the *OWL 2 QL* profile [45] of the W3C-standardised Web Ontology Language *OWL 2*, which extends *DL-Lite* and is the focus of this work. Like many other ontology languages originating from description logics, *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 aforementioned ontology languages [50, 47, 41, 54, 15, 21, 53, 38, 28, 44, 40], and a few mature OBDA systems have emerged, including pioneering MASTRO [12], commercial Stardog [48]

PODS'17, May 14-19, 2017, Raleigh, NC, USA

DOI: http://dx.doi.org/10.1145/3034786.3034791

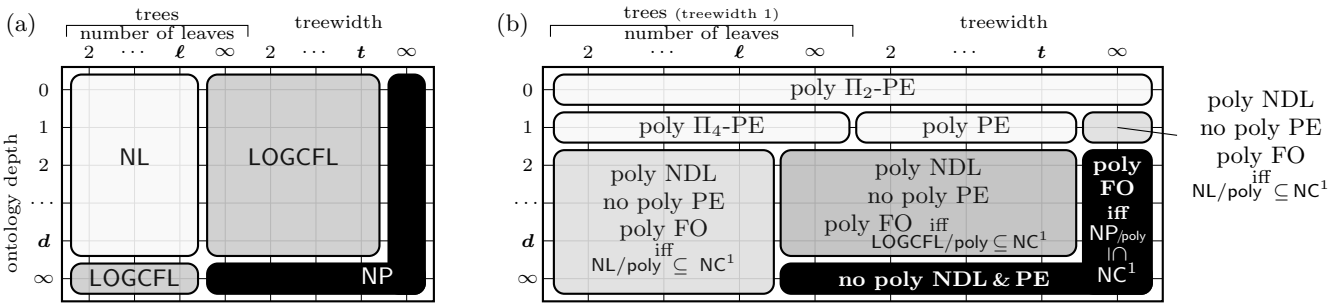


Figure 1: OMQ answering in *OWL 2 QL* (a) combined complexity and (b) the size of rewritings.

and Ultrawrap [55], and the Optique platform [24] with the query answering engine Ontop [51, 42].

Our concern in this paper is the overhead of OMQ answering—that is, 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 is NP-complete for combined complexity [13], which for both measures 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) [37], in sharp contrast to the well-known LOGCFL-completeness of evaluating bounded treewidth CQs [62, 14, 27]. 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—that is, sets of unary and binary ground atoms—and $\mathcal{M}$ is the identity) [13, 37, 35, 5], indicates three tractable cases:

$\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$**:** ontologies of depth $\leq \boldsymbol{d}$ coupled with CQs of treewidth $\leq \boldsymbol{t}$ (for fixed $\boldsymbol{d}, \boldsymbol{t}$);

$\mathsf{OMQ}(\boldsymbol{d}, 1, \boldsymbol{\ell})$**:** ontologies of depth $\leq \boldsymbol{d}$ with tree-shaped CQs with $\leq \boldsymbol{\ell}$ leaves (for fixed $\boldsymbol{d}, \boldsymbol{\ell}$);

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

Observe, in particular, that when the depth of ontologies is bounded by a fixed constant, the complexity of OMQ answering is precisely the same as of evaluating the underlying CQs. If we place no restriction on the ontology, then tractability for 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 [50], 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 [36, 26, 35, 5]. Thus, we see, for example, that PE-rewritings for OMQs from $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$ can be of superpolynomial 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 [21], Presto [54] and Rapid [15], 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, of course, $\mathsf{LOGCFL} = \mathsf{NP}$.) 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 lower bounds were established using sequences $(\mathcal{T}_n, \boldsymbol{q}_n)$ of OMQs, where the depth of $\mathcal{T}_n$ grows with n [37, 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 because a typical OBDA scenario has users posing different queries over the same ontology. Our third main result is that this is surprisingly 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 CQs $\boldsymbol{q}$ and $(\mathcal{T}_\ddagger, \boldsymbol{q})$ with linear CQs $\boldsymbol{q}$ is NP- and LOGCFL-hard for query complexity, respectively. We also show that, unless $\mathsf{P} = \mathsf{NP}$, no polynomial-time algorithm can

construct FO-rewritings of the OMQs $(\mathcal{T}_\dagger, \boldsymbol{q})$, 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 that 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 ($W[1]$-hardness) when the ontology depth (respectively, number of leaves) is taken as the parameter. For the infinite depth case, we show in Section 5 that NP-hardness holds even for a fixed ontology. The final section of the paper presents preliminary experiments that compare our new NDL-rewritings to those produced by existing rewriting engines and also discusses possible directions for future work. Concrete examples of our three types of NDL-rewriting are provided in the appendix. All omitted proofs and details of the experiments can be found in the long version of this paper [4].

2. PRELIMINARIES

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

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

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 $\mathcal{T}$ (where, as usual, $\varphi \leftrightarrow \psi$ is an abbreviation for $\varphi \to \psi$ and $\psi \to \varphi$). 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 $\boldsymbol{x} \cup \boldsymbol{y}$. We assume, without loss of generality, that CQs contain no constants. We often regard a CQ as the set of its atoms; in particular, $|\boldsymbol{q}|$ is the number of atoms in $\boldsymbol{q}$. 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; $\boldsymbol{q}$ is *tree-shaped* if $\mathcal{G}$ is a tree, and *linear* if $\mathcal{G}$ is a tree with two leaves.

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

By an *ontology-mediated query* (OMQ) we understand 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*. The *size* of $\boldsymbol{Q}$ is $|\boldsymbol{Q}| = |\mathcal{T}| + |\boldsymbol{q}|$, where $|\mathcal{T}|$ is the number of symbols in $\mathcal{T}$.

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 such that $\mathcal{T} \models \forall xy\, P(x,y)$. 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 consisting of the empty word ε and all 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 $\mathcal{T} \models \exists y\, \varrho(y,x) \to A(x)$;
- $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models P(u,v)$ iff one of the three 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 $u = v\varrho^-$.

We say that $\mathcal{T}$ is *of depth* 0 if it does not contain any axioms with $\exists$ on the right-hand side, excepting the normalisation axioms[2]. Otherwise, we say that $\mathcal{T}$ is *of depth* $0 < \boldsymbol{d} < \infty$ if $\boldsymbol{d}$ is the maximum length of the words in $\boldsymbol{W}_\mathcal{T}$, and it is *of depth* ∞ if $\boldsymbol{W}_\mathcal{T}$ is infinite. (Note that the depth of $\mathcal{T}$ is computable in NL; cf. [25, 8] 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 (that is, an FO-formula with $\exists$, $\lor$ and $\land$ only), 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 a rewriting $\boldsymbol{q}'$ is the number of symbols in it.

[2]This somewhat awkward definition of depth 0 ontologies is due to the use of normalisation axioms, which may introduce unnecessary words on length 1 in $\boldsymbol{W}_\mathcal{T}$.

We also consider rewritings in the form of nonrecursive datalog queries. A *datalog program*, Π, is a finite set of Horn clauses $\forall \boldsymbol{z}\,(\gamma_0 \leftarrow \gamma_1 \wedge \dots \wedge \gamma_m)$, where each γ_i is an atom $Q(\boldsymbol{y})$ with $\boldsymbol{y} \subseteq \boldsymbol{z}$ or an equality $(z = z')$ with $z, z' \in \boldsymbol{z}$. (As usual, we omit $\forall \boldsymbol{z}$ from clauses.) The atom γ_0 is the *head* of the clause, and $\gamma_1, \dots, \gamma_m$ its *body*. All variables in the head must occur in the body, and $=$ can only occur in the body. The predicates in the heads of clauses in Π are *IDB predicates*, the rest (including $=$) *EDB predicates*. A predicate Q *depends* on P in Π if Π has a clause with Q in the head and P in the body. Π is a *nonrecursive datalog* (NDL) *program* if the (directed) *dependence graph* of the dependence relation is acyclic. The size $|\Pi|$ of Π is the number of symbols in it.

An *NDL query* is a pair $(\Pi, G(\boldsymbol{x}))$, where Π is an NDL program and $G(\boldsymbol{x})$ 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 *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 called 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 [33]. 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, without loss of generality, we can (and will) assume that our ontologies $\mathcal{T}$ do not contain $\bot$. Indeed, we can always incorporate into rewritings subqueries that check whether the left-hand side of an axiom with $\bot$ holds and output all tuples of constants if this is the case [10].

3. OPTIMAL NDL-REWRITINGS

In order to construct theoretically optimal NDL-rewritings for OMQs from 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 $(\Pi, G(\boldsymbol{x}))$ is called *ordered* if each of its IDB predicates Q comes with a fixed list of variables $\boldsymbol{x}_Q \subseteq \boldsymbol{x}$, called the *parameters of* Q, such that

(i) in each occurrence of Q in Π, the parameters occupy the last $|\boldsymbol{x}_Q|$ positions: that is, $Q(\boldsymbol{z}, \boldsymbol{x}_Q)$; parameters can, however, occur in other positions too;

(ii) the parameters of G are $\boldsymbol{x}$; and

(iii) in every clause, the parameters of the head include all the parameters of the predicates in the body.

Observe that Boolean NDL queries are trivially ordered (the lists of parameters are empty for all IDBs).

The *width* $\mathsf{w}(\Pi, G)$ of an ordered $(\Pi, G(\boldsymbol{x}))$ is the maximal number of non-parameter variables in a clause of Π.

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

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

is ordered with parameter x and has width 1 (the conditions do not restrict the EDB predicate R). Replacing $Q(x)$ by $Q(y)$ in the first clause yields an NDL query that is not ordered in view of (i). A further change of $Q(x)$ in the second clause to $Q(y)$ 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 [18] 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 checking whether $\Pi, \mathcal{A} \models G(\boldsymbol{a})$ can be done 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 every* $\mathsf{w} > 0$*, evaluation of linear NDL queries of width at most* 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 vertex in a certain set X in the grounding graph $\mathfrak{G}$, which is 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 $Q'(\boldsymbol{c}') \leftarrow Q(\boldsymbol{c}) \wedge S_1(\boldsymbol{c}_1) \wedge \dots \wedge S_k(\boldsymbol{c}_k)$ with $S_i(\boldsymbol{c}_i) \in \mathcal{A}$, for $1 \le i \le k$ (we assume that $\mathcal{A}$ 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 $Q(\boldsymbol{c}) \leftarrow S_1(\boldsymbol{c}_1) \wedge \dots \wedge S_k(\boldsymbol{c}_k)$ in $\Pi^{\boldsymbol{a}}_{\mathcal{A}}$ such that $S_i(\boldsymbol{c}_i) \in \mathcal{A}$, for $1 \le i \le k$. 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* $\mathsf{w}(\Pi', G) \le \mathsf{w} + 1$*.*

We note that a possible increase of the width by 1 is due to the 'replacement' of unary atoms $A(z)$ by binary atoms $\varrho(y,z)$ whenever $\mathcal{T} \models \exists y\, \varrho(y,z) \to A(z)$.

3.1.2 Skinny NDL in LOGCFL

The complexity class LOGCFL can be defined using *nondeterministic auxiliary pushdown automata* (NAuxPDAs) [16], which are nondeterministic Turing machines with an additional work tape constrained to operate as a pushdown store. Sudborough [58] 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 equivalently 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 $\boldsymbol{C}$ [61, pp. 392–397].

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

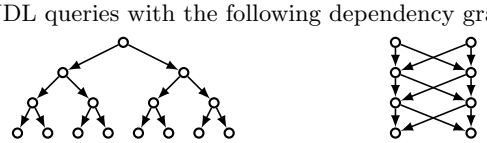
LEMMA 4. *For any skinny* $(\Pi, G(\boldsymbol{x}))$ *and any data instance* $\mathcal{A}$*, query evaluation can be done by an NAuxPDA in space* $\log|\Pi| + \mathsf{w}(\Pi, G) \cdot \log|\mathcal{A}|$ *and time* $2^{O(\mathsf{d}(\Pi,G))}$.

PROOF. Using the atoms of the grounding $\Pi^{\boldsymbol{a}}_{\mathcal{A}}$ as gates and inputs, we 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. Having observed 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. ❑

Observe that Lemma 4 holds for NDL queries with any *bounded* number of atoms, not only two. In the rewritings we propose in Sections 3.2 and 3.4, however, the number of atoms in the clauses is not bounded by a constant. We require the following notion to generalise skinny programs. A function ν from the predicate names in Π to $\mathbb{N}$ is called a *weight function* for an NDL query $(\Pi, G(\boldsymbol{x}))$ if

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

for any clause $Q(\boldsymbol{z}) \leftarrow P_1(\boldsymbol{z}_1) \wedge \dots \wedge P_k(\boldsymbol{z}_k)$ in Π. Note that $\nu(P)$ can be 0 for an EDB predicate P. To illustrate, we consider NDL queries with the following dependency graphs:



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

Let e_Π be the maximum number of EDB predicates in a clause of Π. The *skinny depth* $\mathsf{sd}(\Pi, G)$ of $(\Pi, G(\boldsymbol{x}))$ is the minimum value of $2\mathsf{d}(\Pi, G) + \log \nu(G) + \log \mathsf{e}_\Pi$ over possible weight functions ν. We show, using Huffman coding, that any NDL query $(\Pi, G(\boldsymbol{x}))$ can be transformed into an equivalent skinny NDL query of depth not exceeding $\mathsf{sd}(\Pi, G)$.

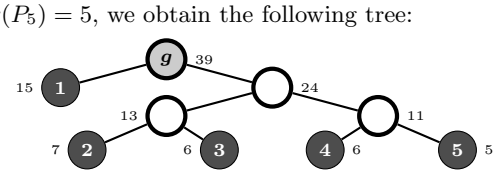
LEMMA 5. *Any NDL query* $(\Pi, G(\boldsymbol{x}))$ *is equivalent to a skinny NDL query* $(\Pi', G(\boldsymbol{x}))$ *such that* $|\Pi'| = O(|\Pi|^2)$, $\mathsf{d}(\Pi', G) \leq \mathsf{sd}(\Pi, G)$, *and* $\mathsf{w}(\Pi', G) \leq \mathsf{w}(\Pi, G)$.

PROOF. Suppose $\mathsf{sd}(\Pi, G) = 2\mathsf{d}(\Pi, G) + \log \nu(G) + \log \mathsf{e}_\Pi$. Without loss of generality, we assume that $\nu(E) = 0$ for EDB predicates E. First, we split clauses into EDB and IDB components by replacing each clause ψ of the form $Q(\boldsymbol{z}) \leftarrow \varphi(\boldsymbol{z}')$ with clauses $Q(\boldsymbol{z}) \leftarrow Q^\psi_E(\boldsymbol{z}_E) \wedge Q^\psi_I(\boldsymbol{z}_I)$ and $Q^\psi_\alpha(\boldsymbol{z}_\alpha) \leftarrow \varphi_\alpha(\boldsymbol{z}'_\alpha)$, where $\alpha \in \{E, I\}$, Q^ψ_E and Q^ψ_I are fresh predicates, and $\varphi_E(\boldsymbol{z}'_E)$ and $\varphi_I(\boldsymbol{z}'_I)$ are conjunctions of the EDB and IDB atoms in φ, respectively. The depth of the resulting NDL query $(\Pi_*, G(\boldsymbol{x}))$ is $2\mathsf{d}(\Pi, G)$. Now, each $Q^\psi_E(\boldsymbol{z}_E) \leftarrow \varphi_E(\boldsymbol{z}'_E)$ in Π_* is replaced by $\leq \mathsf{e}_\Pi - 1$ clauses with ≤ 2 atoms in the body, resulting in an NDL query of depth $\leq 2\mathsf{d}(\Pi, G) + \log \mathsf{e}_\Pi$. In the rest of the proof, we focus on the part $\Pi_\dagger$ of Π_* comprising clauses that have predicates Q and Q^ψ_I in their head (thus making the Q^ψ_E EDB predicates). The weight function for $(\Pi_\dagger, G(\boldsymbol{x}))$ is obtained by extending ν with $\nu(Q^\psi_I) = \nu(Q)$ and $\nu(Q^\psi_E) = 0$, for each clause ψ having Q as its head predicate.

Next, by induction on $\mathsf{d}(\Pi_\dagger, G)$, we show that there is an equivalent skinny NDL query $(\Pi'_\dagger, G(\boldsymbol{x}))$ of the required size and width such that $\mathsf{d}(\Pi'_\dagger, G) \leq \mathsf{d}(\Pi_\dagger, G) + \log \nu(G)$. We take $\Pi'_\dagger = \Pi_\dagger$ if $\mathsf{d}(\Pi_\dagger, G) = 0$. Otherwise, let ψ be a clause of the form $G(\boldsymbol{z}) \leftarrow P_1(\boldsymbol{z}_1) \wedge \dots \wedge P_k(\boldsymbol{z}_k)$ in $\Pi_\dagger$, for $k > 2$. By construction, all clauses in $\Pi_\dagger$ with EDB predicates are of the form $Q(\boldsymbol{z}) \leftarrow Q^\psi_E(\boldsymbol{z}_E) \wedge Q^\psi_I(\boldsymbol{z}'_I)$, with two atoms in the body. So, the P_i in ψ are IDB predicates and $\nu(G) \geq \nu(P_i) > 0$. Suppose that, for each i $(1 \leq i \leq k)$, we have an NDL query (Π'_i, P_i) equivalent to $(\Pi_\dagger, P_i)$ with

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

Construct the Huffman tree [31] for the alphabet $\{1, \dots, k\}$, where the frequency of i is $\nu(P_i)/\nu(G)$. 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 $1, \dots, k$, root g, and $k-2$ internal nodes such that the length of the path from g to any leaf i is $\leq \lceil \log(\nu(G)/\nu(P_i)) \rceil$. For each internal node v, 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 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 $(1 \leq i \leq k)$ with clauses $P_v(\boldsymbol{z}_v) \leftarrow P_{u_1}(\boldsymbol{z}_{u_1}) \wedge P_{u_2}(\boldsymbol{z}_{u_2})$, for each v with immediate successors u_1 and u_2. The number of the new clauses is $k - 1$. By (3), we have:

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

Let $\Pi'_\dagger$ be the result of applying this transformation to each clause in $\Pi_\dagger$ with head $G(\boldsymbol{z})$ and > 2 atoms in the body.

Finally, we add to $\Pi'_\dagger$ the clauses with the Q_E predicates and denote the result by Π'. 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:

THEOREM 6. *For every $c > 0$ and $\mathsf{w} > 0$, evaluation of NDL queries $(\Pi, G(\boldsymbol{x}))$ of width at most w and such that $\mathsf{sd}(\Pi, G) \leq c \log |\Pi|$ is in* LOGCFL *for combined complexity.*

We say that a class of OMQs is *skinny-reducible* if, for some fixed $c > 0$ and $\mathsf{w} > 0$, 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 such that $\mathsf{sd}(\Pi, G) \leq c \log |\Pi|$ and $\mathsf{w}(\Pi, G) \leq \mathsf{w}$. Theorem 6 and the transformation * give the following:

COROLLARY 7. *For any skinny-reducible class, the OMQ answering problem is in* LOGCFL *for combined complexity.*

In the following subsections, we will exploit the results obtained above to construct optimal NDL-rewritings for the three classes of tractable OMQs. Concrete examples of our rewritings are provided in the appendix of the present paper.

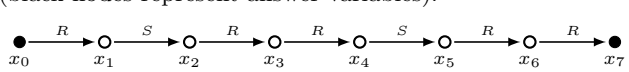
3.2 LOGCFL Rewritings for OMQ(d, t, ∞)

We begin by considering the case of bounded treewidth queries coupled with bounded depth ontologies. Recall (see, e.g., [23]) 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

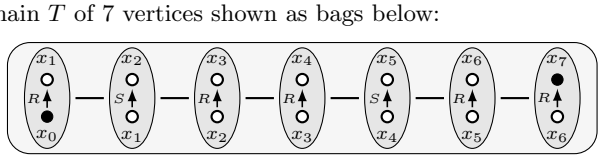
- for every $v \in V$, there exists a node t with $v \in \lambda(t)$;
- for every $e \in E$, there exists a node t with $e \subseteq \lambda(t)$;
- for every $v \in V$, the nodes $\{t \mid v \in \lambda(t)\}$ induce a connected subgraph of T (called a *subtree* of T).

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:



We now establish the following theorem, which, by the results of the preceding subsection, yields an NDL-rewriting with the desired LOGCFL complexity.

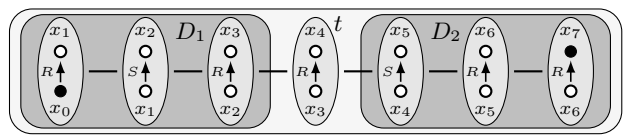
THEOREM 9. *For any fixed $\boldsymbol{d} \geq 0$ and $\boldsymbol{t} \geq 1$, the class* $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$ *is skinny-reducible.*

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 different from t belongs to exactly one the 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 $y, z \in \lambda(t) \cap \mathsf{dom}(\boldsymbol{w})$:

- 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)* $\boldsymbol{w}(y) = \boldsymbol{w}(z)$, $\mathcal{T} \models P(x, x)$; *(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 every $D \in \mathfrak{D}$ 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 not be disjoint). For each type $\boldsymbol{s}$ with $\mathsf{dom}(\boldsymbol{s}) = \lambda(\sigma(D))$ that is compatible with $\sigma(D)$ and agrees with $\boldsymbol{w}$ on their common domain, the NDL program $\Pi_{\boldsymbol{Q}}^{\text{LOG}}$ contains

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

where $(\boldsymbol{s} \cup \boldsymbol{w}) \restriction \partial D'$ 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 $(\Pi_{\boldsymbol{Q}}^{\text{LOG}}, G_T^{\boldsymbol{\varepsilon}})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$.

Lemma 12. *For any complete data instance $\mathcal{A}$, subtree $D \in \mathfrak{D}$, type $\boldsymbol{w}$ with $\mathsf{dom}(\boldsymbol{w}) = \partial D$, and any $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}_D|}$ and $\boldsymbol{b} \in \mathsf{ind}(\mathcal{A})^{|\partial D|}$, we have $\Pi_{\boldsymbol{Q}}^{\text{LOG}}, \mathcal{A} \models G_D^{\boldsymbol{w}}(\boldsymbol{b}, \boldsymbol{a})$ iff there is a homomorphism $h \colon \boldsymbol{q}_D \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(x) = \boldsymbol{a}(x)$, for $x \in \boldsymbol{x}_D$, and $h(z) = \boldsymbol{b}(z)\boldsymbol{w}(z)$, for $z \in \partial D$.*

Now fix $\boldsymbol{d}$ 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 without loss of generality that T has at most $|\boldsymbol{q}|$ nodes. We take the following weight function: $\nu(G_D^{\boldsymbol{w}}) = |D|$, where $|D|$ is the number of nodes in D. Clearly, $\nu(G_T^{\boldsymbol{\varepsilon}}) \leq |\boldsymbol{q}|$. By Lemma 10, we have

$$\mathsf{w}(\Pi_{\boldsymbol{Q}}^{\text{LOG}}, G_T^{\boldsymbol{\varepsilon}}) \leq \max\nolimits_D |\partial D \cup \lambda(\sigma(D))| \leq 3(\boldsymbol{t} + 1),$$
$$\mathsf{sd}(\Pi_{\boldsymbol{Q}}^{\text{LOG}}, G_T^{\boldsymbol{\varepsilon}}) \leq 4 \log |T| + 2 \log |\boldsymbol{q}| \leq 6 \log |\boldsymbol{q}| \leq 6 \log |\Pi_{\boldsymbol{Q}}^{\text{LOG}}|;$$

the last inequality is due to $\Pi_{\boldsymbol{Q}}^{\text{LOG}}$ containing every atom of $\boldsymbol{q}$ (with variables renamed). 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 $G_D^{\boldsymbol{w}}$, so $\Pi_{\boldsymbol{Q}}^{\text{LOG}}$ is of polynomial size. Finally, we note that a tree decomposition of treewidth $\leq \boldsymbol{t}$ can be computed using an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer [27], and so the NDL-rewriting can also be constructed by an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer. We have thus shown that the class $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$ is skinny-reducible, establishing Theorem 9.

The obtained NDL-rewriting shows 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

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

Indeed, we can evaluate $(\Pi_{\boldsymbol{Q}}^{\text{LOG}}, G_T^{\boldsymbol{\varepsilon}}(\boldsymbol{x}))$ in time polynomial in $|\Pi_{\boldsymbol{Q}}^{\text{LOG}}|$ and $|\mathcal{A}|^{\mathsf{w}(\Pi_{\boldsymbol{Q}}^{\text{LOG}}, G_T^{\boldsymbol{\varepsilon}})}$, which are bounded by a polynomial in $|\mathcal{T}|^{3\boldsymbol{d}(\boldsymbol{t}+1)}$, $|\boldsymbol{q}|$ and $|\mathcal{A}|^{3(\boldsymbol{t}+1)}$.

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

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

Theorem 13. *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 $\boldsymbol{w}(z) = \boldsymbol{s}(z')$, $\mathcal{T} \models P(x, x)$, or $\mathcal{T} \models \varrho(x, y) \to P(x, y)$ and either $\boldsymbol{s}(z') = \boldsymbol{w}(z)\varrho$ or $\boldsymbol{w}(z) = \boldsymbol{s}(z')\varrho^-$.

Consider the NDL program $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ defined as follows. For every $0 \le n < M$ 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 that $(\Pi^{\text{LIN}}_{\boldsymbol{Q}}, G(\boldsymbol{x}))$ is a rewriting of $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ over complete data instances.

Lemma 14. *For any complete data instance $\mathcal{A}$, any predicate $G^{\boldsymbol{w}}_n$, and 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 $h(x) = \boldsymbol{a}(x)$, for $x \in \boldsymbol{x}^n$, and $h(z) = \boldsymbol{b}(z)\boldsymbol{w}(z)$, for $z \in \boldsymbol{z}^n_\exists$.*

It should be clear that $\Pi^{\text{LIN}}_{\boldsymbol{Q}}$ is a linear NDL program of width $\le 2\boldsymbol{\ell}$ and containing $\le |\boldsymbol{q}| \cdot |\mathcal{T}|^{2\boldsymbol{d}\boldsymbol{\ell}}$ 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 shows that answering OMQs $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ 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

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

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 $|\mathcal{T}|^{2\boldsymbol{d}\boldsymbol{\ell}}$, $|\boldsymbol{q}|$ and $|\mathcal{A}|^{2\boldsymbol{\ell}}$.

3.4 LOGCFL Rewritings for OMQ(∞, 1, $\boldsymbol{\ell}$)

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

Theorem 15. *For any fixed number of leaves $\boldsymbol{\ell} \ge 2$, the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ is skinny-reducible.*

For OMQs with bounded-leaf CQs and ontologies of unbounded depth, our rewriting uses the notion of tree witness [38]. Consider an OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$. Suppose $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ is 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

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

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 drawn from $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ is based on the following observation:

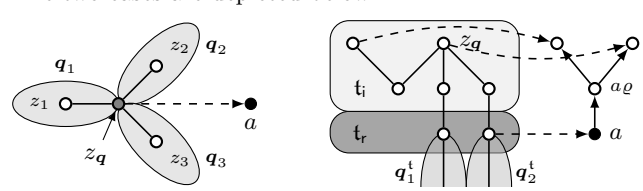
Lemma 16 ([35]). *Every tree T of size n has a node splitting it into subtrees of size $\le \lceil n/2 \rceil$.*

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 16 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 16; if $\boldsymbol{q}$ has two variables and at least one of them is existentially quantified, 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 $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ that has no existentially quantified variables, we include the clause $G_\boldsymbol{q}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$. For any $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ with existential variables, we include

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

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 $\mathcal{G}$, and, for any tree witness $\mathfrak{t}$ of $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathfrak{t}_\mathsf{r} \neq \emptyset$ and $z_\boldsymbol{q} \in \mathfrak{t}_\mathsf{i}$ and any ϱ generating $\mathfrak{t}$, the clause

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

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 [35]. By adapting the proof, we can show that the NDL query $(\Pi^{\text{TW}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}(\boldsymbol{x}_0))$ is indeed a rewriting.

Lemma 17. *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 query $\boldsymbol{q}(\boldsymbol{x}) \in \mathfrak{Q}$ and any $\boldsymbol{a} \in \mathsf{ind}(\mathcal{A})^{|\boldsymbol{x}|}$, we have $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, \mathcal{A} \models G_{\boldsymbol{q}}(\boldsymbol{a})$ iff there is a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ with $h(\boldsymbol{x}) = \boldsymbol{a}$.*

Now fix $\boldsymbol{\ell} > 1$ and consider $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}_0(\boldsymbol{x}))$ from the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$. The size of the program $\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}$ is polynomially bounded in $|\boldsymbol{Q}|$ since $\boldsymbol{q}_0$ has $O(|\boldsymbol{q}_0|^{\boldsymbol{\ell}})$ tree witnesses and tree-shaped subqueries. It is readily seen that the function ν defined by setting $\nu(G_{\boldsymbol{q}}) = |\boldsymbol{q}|$, for each $\boldsymbol{q} \in \mathfrak{Q}$, is a weight function for $(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}(\boldsymbol{x}))$ with $\nu(G_{\boldsymbol{q}_0}) \leq |\boldsymbol{Q}|$. Moreover, by Lemma 16, $\mathsf{d}(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}) \leq \log \nu(G_{\boldsymbol{q}_0}) + 1$; and clearly, $\mathsf{w}(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}) \leq \boldsymbol{\ell}+1$. Finally, we note that, since the number of leaves is bounded, it is in NL to decide whether a vertex satisfies the conditions of Lemma 16, and in LOGCFL to decide whether $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}(a)$, for bounded-leaf tree-shaped CQs $\boldsymbol{q}(x)$ [5], or whether a (logspace) representation of a possible tree witness is indeed a tree witness. This allows us to show that $(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G_{\boldsymbol{q}_0}(\boldsymbol{x}))$ can be generated by an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer. By Corollary 7, the obtained NDL-rewritings can be evaluated in LOGCFL.

It also follows that answering OMQs $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with a tree-shaped CQ with $\boldsymbol{\ell}$ leaves over any $\mathcal{A}$ can be done in time

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

Indeed, $(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G(\boldsymbol{x}))$ can be evaluated in time polynomial in $|\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}|$ and $|\mathcal{A}|^{\mathsf{w}(\Pi^{\mathrm{Tw}}_{\boldsymbol{Q}}, G)}$, which are bounded by polynomials in $|\mathcal{T}|$, $|\boldsymbol{q}|^{\boldsymbol{\ell}}$ and $|\mathcal{A}|^{\boldsymbol{\ell}+1}$, respectively.

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

Proof. The proof is by reduction of the parameterised problem p-HittingSet, known to be $W[2]$-complete [23]:

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 $e \in E$.

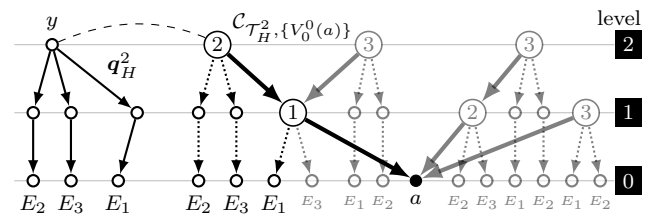
(Such a set A of vertices is called a *hitting set of size k*.) Suppose that $H = (V, E)$ is a hypergraph with vertices $V = \{v_1, \dots, v_n\}$ and hyperedges $E = \{e_1, \dots, e_m\}$. Let $\mathcal{T}^k_H$ be the (normal form of an) ontology with the following axioms, for $1 \leq l \leq k$:

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

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 $e_3 = \{1, 2\}$, are shown below:



Points $\textcircled{i}$ at level l belong to V^l_i. In the long version [4], 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 18 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 $\mathcal{T}, \{A(a)\} \models \boldsymbol{q}$.

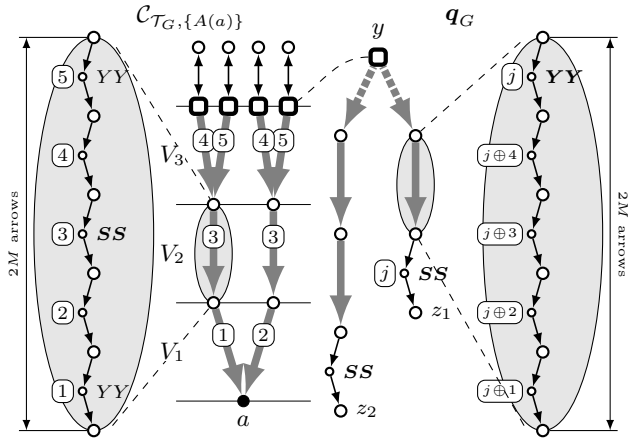
Theorem 19. *pLeaves*-TreeOMQ *is $W[1]$-hard.*

Proof. The proof is by reduction of the following $W[1]$-complete PartitionedClique problem [22]:

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:

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

It can be shown 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 the class $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$ can be answered (via NDL-rewriting) over a data instance $\mathcal{A}$ in time $poly(|\mathcal{T}|, |\boldsymbol{q}|^{\boldsymbol{\ell}}, |\mathcal{A}|^{\boldsymbol{\ell}})$. Theorem 19 shows that no algorithm can do this in time $f(\boldsymbol{\ell}) \cdot poly(|\mathcal{T}|, |\boldsymbol{q}|, |\mathcal{A}|)$, 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 was shown [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 [34], 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 ontology $\mathcal{T}$ 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 20. *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.*

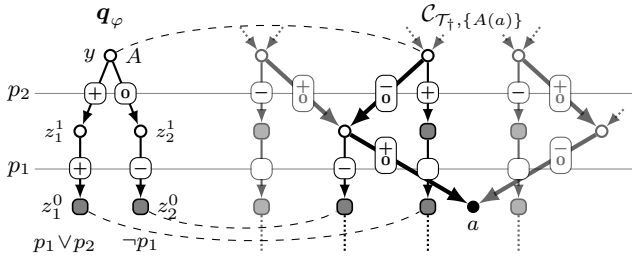
PROOF. The proof is by reduction of SAT. Given a CNF φ with variables $p_1, \dots, p_k$ and clauses $\chi_1, \dots, \chi_m$, take a Boolean CQ $\boldsymbol{q}_\varphi$ with $A(y)$ and, for $1 \le j \le m$, the following atoms with $z_j^k = y$:

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

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

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

Intuitively, $(\mathcal{T}_\dagger, \{A(a)\})$ generates an infinite binary tree, where 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 can show that $\mathcal{T}_\dagger, \{A(a)\} \models \boldsymbol{q}_\varphi$ iff φ is satisfiable. To illustrate, the CQ $\boldsymbol{q}_\varphi$ for $\varphi = (p_1 \vee p_2) \wedge \neg p_1$ 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 21. *No polynomial-time algorithm can construct FO- or NDL-rewritings of $\boldsymbol{Q}_\varphi$ 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 polytime by evaluating $\boldsymbol{q}'_\varphi$ over $\{A(a)\}$. ❑

Curiously enough, Corollary 21 can be complemented with

Theorem 22. *The $\boldsymbol{Q}_\varphi$ have polynomial FO-rewritings.*

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

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

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 [26, Corollary 14]. Recall that the proof of Theorem 20 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 [26]. ❑

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 can be found in [4]). 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 [39] or description logic instance queries [3].

Theorem 23. *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 23 is not applicable to PE.[3]

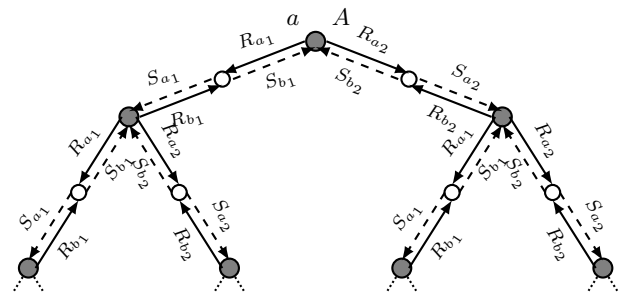
Theorem 24. *Evaluating PE-queries over trees in $\mathfrak{T}$ is* NP*-hard.*

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

[3]This result might be known but we could not find it in the literature, and so provide a proof in [4].

Theorem 25. *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 $\mathcal{L}$ [30, 57]. We construct an ontology $\mathcal{T}_\ddagger$ and a logspace transducer that converts the words w over the alphabet of $\mathcal{L}$ to linear CQs $\boldsymbol{q}_w$ such that $w \in \mathcal{L}$ iff $\mathcal{T}_\ddagger, \{A(a)\} \models \boldsymbol{q}_w$. To illustrate, take the context-free language $\mathcal{B}$ over the alphabet $\Sigma = \{a_1, b_1, a_2, b_2\}$ generated by the grammar $S \to SS$, $S \to \varepsilon$, $S \to a_1Sb_1$, $S \to a_2Sb_2$. Let $\mathcal{T}_\mathcal{B}$ be an ontology such that the canonical model of $(\mathcal{T}_\mathcal{B}, \{A(a)\})$ looks like in the picture below:



Then, for every word $w = c_0 \dots c_n$ over Σ, we have $w \in \mathcal{B}$ just in case $\mathcal{T}_\mathcal{B}, \{A(a)\} \models A(u_0) \wedge \gamma_w \wedge A(u_{n+1})$, where

$$\gamma_w = \bigwedge_{0 \le i \le n} \big(R_{c_i}(u_i, v_i), S_{c_i}(v_i, u_{i+1})\big).$$

Although $\mathcal{B}$ is not LOGCFL-hard, by adding axioms to $\mathcal{T}_\mathcal{B}$ that generate additional branches from the points ●, we can construct an ontology $\mathcal{T}_\ddagger$ recognising $\mathcal{L}$ (see [4] for details).

6. EXPERIMENTS & CONCLUSIONS

The main positive result of this paper is the development of theoretically optimal NDL-rewritings for OMQs belonging to one of the classes $\mathsf{OMQ}(\boldsymbol{d}, \boldsymbol{t}, \infty)$, $\mathsf{OMQ}(\boldsymbol{d}, 1, \boldsymbol{\ell})$, and $\mathsf{OMQ}(\infty, 1, \boldsymbol{\ell})$. 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., [49] for evidence in the RDF setting), and indeed, recent tools for query formulation over ontologies (like [56]) produce tree-shaped CQs. Moreover, the majority of important real-world *OWL 2* ontologies are of finite depth; see [17] for statistics. In the context of OBDA, *OWL 2 QL* ontologies are often built starting from the database schemas (bootstrapping [32]), 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 [50, 47, 15, 28, 44, 40], but is also true of more elaborate PE-rewriters (which use disjunctions inside conjunctions) [51, 59] whose rewritings in theory can be of superpolynomial size; see Fig. 1 (b). Surprisingly, even NDL-rewriters like Clipper [21], Presto [54],

[4]http://sws.ifi.uio.no/project/npd-v2/

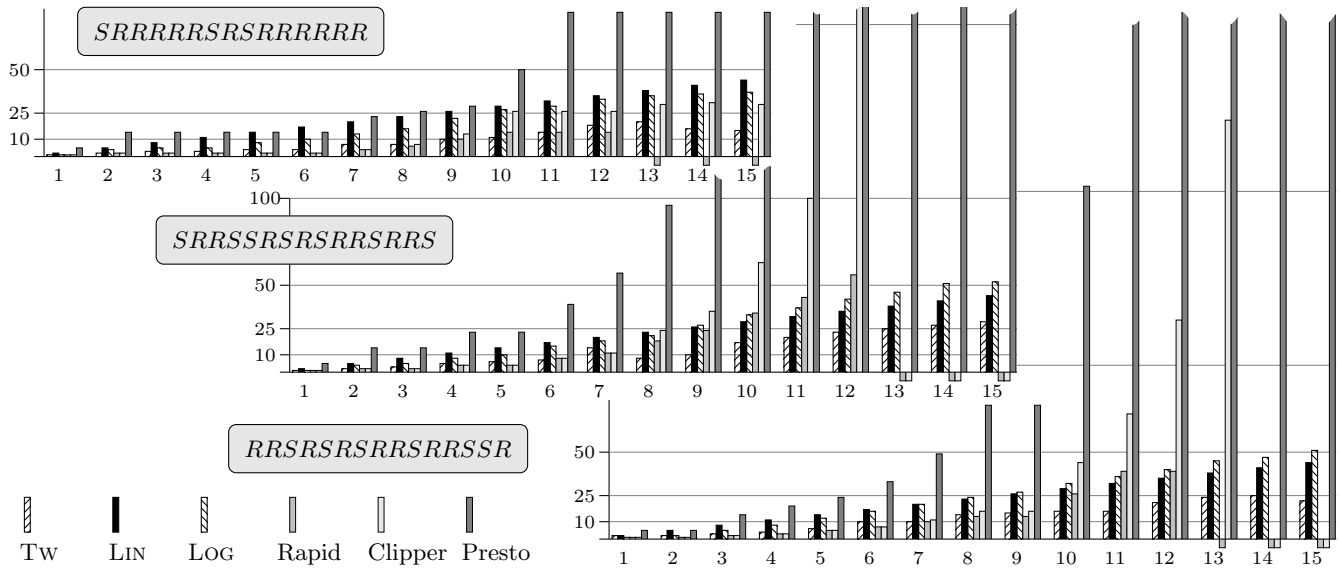


Figure 2: The size of NDL-rewritings produced by different algorithms.

and Rapid [15] 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 the classes $\mathsf{OMQ}(\boldsymbol{d},\boldsymbol{t},\infty)$, $\mathsf{OMQ}(\boldsymbol{d},1,\boldsymbol{\ell})$ 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 [46]. The experiments (detailed in [4]) 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 NL/LOGCFL rewritings, as it simply materialises all of 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. For example, one could first define a 'cost function' on some set of alternative rewritings that roughly estimates their evaluation time and then construct a rewriting minimising this function. Such a performance-oriented approach was introduced and exploited in [7], where the target language for OMQ rewritings was joins of UCQs (unions of CQs). Other optimisation techniques for removing redundant rules or sub-queries from rewritings [54, 51, 29, 40] or exploiting the emptiness of certain predicates [60] are also relevant here. In the context of OBDA with relational databases and mappings, integrity constraints [53, 52] and the structure of mappings [19] are particularly important for optimisation.

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.

Acknowledgments

This work was supported by the French ANR grant 12-JS02-007-01 'PAGODA: Practical Algorithms for Ontology-Based Data Access', the UK EPSRC grant EP/M012670 'iTract: Islands of Tractability in Ontology-Based Data Access', the Russian Foundation for Basic Research grant MK-7312.2016.1, and the Russian Academic Excellence Project 5-100. We thank the developers of Clipper and Rapid for making their systems freely available, and Riccardo Rosati for the opportunity to conduct experiments with Presto.

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] 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.

[4] M. Bienvenu, S. Kikot, R. Kontchakov, V. Podolskii, V. Ryzhikov, and M. Zakharyaschev. The complexity of ontology-based data access with OWL 2 QL and bounded treewidth queries. *CoRR*, 2017.

[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] D. Bursztyn, F. Goasdoué, and I. Manolescu. Teaching an RDBMS about ontological constraints. *PVLDB*, 9(12):1161–1172, 2016.

[8] 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.

[9] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *Journal of Artificial Intelligence Research (JAIR)*, 48:115–174, 2013.

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

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

[19] F. Di Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *Proc. of the 16th Int. Conf. on Extending Database Technology, EDBT 2013*, pages 561–572. ACM, 2013.

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

[21] 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.

[22] 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.

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

[24] 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.

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

[26] 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.

[27] 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.

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

[29] G. Gottlob, G. Orsi, and A. Pieris. Query rewriting and optimization for ontological databases. *ACM Transactions on Database Systems (TODS)*, 39(3):25, 2014.

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

[31] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.

[32] 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.*, volume 1486 of *CEUR Workshop Proceedings*. CEUR-WS, 2015.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 558–573. Springer, 2013.

[52] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Query rewriting and optimisation with database dependencies in Ontop. In *Informal Proc. of the 26th Int. Workshop on Description Logics, DL 2013*, volume 1014 of *CEUR Workshop Proceedings*, pages 917–929. CEUR-WS, 2013.

[53] R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proc. of the 9th Extended Semantic Web Conf., ESWC 2012*, volume 7295 of

Lecture Notes in Computer Science, pages 360–374. Springer, 2012.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

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

[59] M. Thomazo. Compact rewritings for existential rules. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence, IJCAI 2013*, pages 1125–1131. IJCAI/AAAI, 2013.

[60] T. Venetis, G. Stoilos, and V. Vassalos. Rewriting minimisations for efficient ontology-based query answering. In *Proc. of the 28th Int. Conf. on Tools with Artificial Intelligence, ICTAI 2016*, pages 1095–1102. IEEE, 2016.

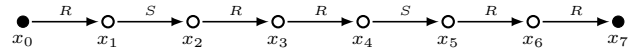
[61] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991.

[62] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of the 7th Int. Conf. on Very Large Data Bases, VLDB'81*, pages 82–94. IEEE Computer Society, 1981.

APPENDIX

We illustrate both standard UCQ-rewritings and the NDL-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)



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

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

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

UCQ rewriting

The nine 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, which we give as an NDL program with goal predicate G:

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

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).

Log-*rewriting*

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

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

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

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

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

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

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 clauses. 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 clause $G_D^{\boldsymbol{w}}(\partial D, \boldsymbol{x}_D) \leftarrow \mathsf{At}^{\boldsymbol{s}}$ in the rewriting. This simplification clearly does not affect the width of the NDL query or the choice of weight function.

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 clause 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 clauses:

$$G_0^{x_0\mapsto\varepsilon}(x_0,x_7) \leftarrow R(x_0,x_1) \wedge P_1^{x_1\mapsto\varepsilon}(x_1,x_7),$$
$$G_0^{x_0\mapsto\varepsilon}(x_0,x_7) \leftarrow (x_0=x_1) \wedge A_{P^-}(x_1) \wedge G_1^{x_1\mapsto P^-}(x_1,x_7),$$

$$G_1^{x_1\mapsto\varepsilon}(x_1,x_7) \leftarrow S(x_1,x_2) \wedge G_2^{x_2\mapsto\varepsilon}(x_2,x_7),$$
$$G_1^{x_1\mapsto\varepsilon}(x_1,x_7) \leftarrow (x_1=x_2) \wedge A_P(x_2) \wedge G_2^{x_2\mapsto P}(x_2,x_7),$$
$$G_1^{x_1\mapsto P^-}(x_1,x_7) \leftarrow A_{P^-}(x_1) \wedge (x_1=x_2) \wedge G_2^{x_2\mapsto\varepsilon}(x_2,x_7),$$

$$G_2^{x_2\mapsto\varepsilon}(x_2,x_7) \leftarrow R(x_2,x_3) \wedge G_3^{x_3\mapsto\varepsilon}(x_3,x_7),$$
$$G_2^{x_2\mapsto P}(x_2,x_7) \leftarrow A_P(x_2) \wedge (x_2=x_3) \wedge G_3^{x_3\mapsto\varepsilon}(x_3,x_7).$$

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

$$G_3^{x_3\mapsto\varepsilon}(x_3,x_7) \leftarrow R(x_3,x_4) \wedge P_4^{x_4\mapsto\varepsilon}(x_4,x_7),$$
$$G_3^{x_3\mapsto\varepsilon}(x_3,x_7) \leftarrow (x_3=x_4) \wedge A_{P^-}(x_4) \wedge G_4^{x_4\mapsto P^-}(x_4,x_7),$$

$$G_4^{x_4\mapsto\varepsilon}(x_4,x_7) \leftarrow S(x_4,x_5) \wedge G_5^{x_5\mapsto\varepsilon}(x_5,x_7),$$
$$G_4^{x_4\mapsto\varepsilon}(x_4,x_7) \leftarrow (x_4=x_5) \wedge A_P(x_5) \wedge G_5^{x_5\mapsto P}(x_5,x_7),$$
$$G_4^{x_4\mapsto P^-}(x_4,x_7) \leftarrow A_{P^-}(x_4) \wedge (x_4=x_5) \wedge G_5^{x_5\mapsto\varepsilon}(x_5,x_7),$$

$$G_5^{x_5\mapsto\varepsilon}(x_5,x_7) \leftarrow R(x_5,x_6) \wedge G_6^{x_6\mapsto\varepsilon}(x_6,x_7),$$
$$G_5^{x_5\mapsto P}(x_5,x_7) \leftarrow A_P(x_5) \wedge (x_5=x_6) \wedge G_6^{x_6\mapsto\varepsilon}(x_6,x_7).$$

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

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

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).

Tw-*rewriting*

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

$$\boldsymbol{q}_{03}(x_0,x_3) = \exists x_1 x_2 \big(R(x_0,x_1) \wedge S(x_1,x_2) \wedge R(x_2,x_3)\big),$$
$$\boldsymbol{q}_{37}(x_3,x_7) = \exists x_4 x_5 x_6 \big(R(x_3,x_4) \wedge S(x_4,x_5) \wedge R(x_5,x_6) \wedge R(x_6,x_7)\big).$$

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 clause:

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

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 clauses for G_{03}:

$$G_{03}(x_0,x_3) \leftarrow R(x_0,x_1) \wedge G_{13}(x_1,x_3),$$
$$G_{03}(x_0,x_3) \leftarrow A_{P^-}(x_0) \wedge (x_0=x_2) \wedge R(x_2,x_3)$$

(although we should write $G_{03}(x_3,x_0)$, placing parameter x_0 last, we keep the natural ordering to improve readability).

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 clauses:

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

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

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

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 clause $G_{\boldsymbol{q}}(\boldsymbol{x}) \leftarrow \boldsymbol{q}(\boldsymbol{x})$, for each $\boldsymbol{q}(\boldsymbol{x})$ without existentially quantified variables. This simplification does not affect the width of the NDL query and the choice of weight function.