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

Meghyn Bienvenu
CNRS & University of Montpellier, France
Vladimir V. Podolskii
Steklov Mathematical Institute, Moscow, Russia
Stanislav Kikot
Birkbeck University of London, UK
Vladislav Ryzhikov
Free University of Bozen-Bolzano, Italy
Roman Kontchakov
Birkbeck University of London, UK
Michael Zakharyaschev
Birkbeck University of London, UK

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

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

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODS 2017 Raleigh, North Carolina, USA
© 2017 ACM. ISBN 123-4567-24-567/08/06…$15.00
doi: 10.475/123_4

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

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

\[
\mathcal{T}, \mathcal{M}(\mathcal{D}) \models q(a) \iff \mathcal{I}_{\mathcal{M}(\mathcal{D})} \models q'(a),
\]

where \( \mathcal{I}_{\mathcal{M}(\mathcal{D})} \) is the FO-structure comprised of the atoms in \( \mathcal{M}(\mathcal{D}) \). When the form of \( \mathcal{M} \) is appropriately restricted (e.g., \( \mathcal{M} \) is a GAV mapping), one can further unfold \( q'(x) \) 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 \( q \). The DL-Lite family of description logics [11] was specifically designed to ensure (1) for OMQs with conjunctive queries (CQs) \( q \). Other ontology languages with this property include linear and sticky tuple-generating dependencies (tgds) [8, 9], and the OWL 2 QL profile [41] of the W3C standardised Web Ontology Language OWL 2, the focus of this work. Like many other ontology languages,
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 $q'(x)$ have been developed and implemented for the preceding languages [46, 43, 37, 49, 13, 18, 48, 34, 25, 40, 36], and a few mature OBDA systems have emerged, including pioneering MASTRO [10], commercial Stardog [44] and Ultrawrap [50], and the Optique platform [21] with the query answering engine Ontop [47, 38].

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

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

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

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

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

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

Finally, we revisit the NP- and LOGCFL-hardness results for OMQs with tree-shaped CQs. The known NP and LOGCFL lower bounds have been established using sequences $(T_n, q_n)$ of OMQs, where the depth of $T_n$ grows with $n \geq 33$. One might thus hope to make answering OMQs with tree-shaped CQs easier by restricting the ontology signature, size, or even by fixing the whole ontology, which is very relevant for applications as a typical OBDA scenario has users posing different queries over the same ontology. Our third main result is that this is not the case: we present ontologies as a typical OBDA scenario has users posing different queries over the same ontology. Thus, we show that no algorithm can construct FO-rewritings of these OMQs do exist.

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

2. PRELIMINARIES

An OWL 2 QL ontology (TBox in description logic), $A$, is a finite set of sentences (axioms) of the forms

\[
\forall x (\tau(x) \rightarrow \tau'(x)), \quad \forall x (\tau(x) \land \tau'(x) \rightarrow \bot), \quad \forall x y (g(x, y) \rightarrow g'(x, y)), \quad \forall x (g(x, y) \land g'(x, y) \rightarrow \bot), \quad \forall x (g(x, x) \rightarrow \bot),
\]

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

\[
\tau(x) ::= \top \mid A(x) \mid \exists y g(x, y), \quad g(x, y) ::= \top \mid P(x, y) \mid P(y, x).
\]

When writing ontology axioms, we omit the universal quantifiers and denote by $R_T$ the set of binary predicates $P$ occurring in $T$ and their inverses $P^\ominus$, assuming that $P^\ominus = P$. For every $\varrho \in R_T$, we take a fresh unary predicate $A_\varrho$ and add $A_\varrho(x) \leftrightarrow \exists y g(x, y)$ to $T$.

The resulting ontology is said to be in normal form, and we assume, without loss of generality, that all our ontologies are in normal form.

A data instance, $A$, is a finite set of unary or binary ground atoms (called an $A$Box in description logic). We denote by $\text{ind}(A)$ the set of individual constants in $A$ and write $g(a, b) \in A$ if $P(a, b) \in A$ and $g = P$, or $P(b, a) \in A$ and $g = P^\ominus$. We say that $A$ is complete for an ontology $T$ if $T, A \models S(a)$ implies $S(a) \in A$, for any ground atom $S(a)$ with $a \in \text{ind}(A)$.

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

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

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

\[
- a \in \text{ind}(A) \text{ and } T, A \models \exists y g_1(a, y); \\
- T \models g_i(x, x), \text{ for } 1 \leq i \leq n; \\
- T \models \exists x g_i(x, y) \rightarrow \exists z g_{i+1}(y, z), \text{ but } T \models g_i(x, y) \rightarrow g_{i+1}(y, x), \text{ for } 1 \leq i < n.
\]

We denote by $W_\mathcal{T}$ the set of words $g_1 \ldots g_n \in R^*_\mathcal{T}$ satisfying the last two conditions. Every $a \in \text{ind}(A)$ is interpreted in $\mathcal{C}_{T,A}$ by itself, and unary and binary predicates are interpreted as follows:

\[
- \mathcal{C}_{T,A} \models A(u) \text{ iff either } u \in \text{ind}(A) \text{ and } T, A \models A(u), \\
- \text{ or } u = u\varrho \text{ with } T \models \exists y g(x, y) \rightarrow A(x);
\]

\footnote{If the meaning is clear from the context, we use set-theoretic notation for lists.}
- \( \mathcal{C}_{T,A} \models P(u,v) \) iff one of the 3 conditions holds:
  (i) \( u,v \in \text{ind}(A) \) and \( T,A \models P(u,v) \);
  (ii) \( u = v \) and \( T \models P(x,x) \);
  (iii) \( T \models g(x,y) \rightarrow P(x,y) \) and
  either \( v = ug \) or \( v = v^g \).

\( T \) is of depth \( \infty \) if \( W_T \) is infinite, and of depth \( d < \infty \)
if \( d \) is the maximum length of the words in \( W_T \). (Note that the depth of \( T \) or \( \text{ind}(A) \).

An FO-formula \( q(x) \), possibly with equality, is an
\( \text{FO-rewriting of an OMQ} \ Q(x) \) = \((T,q'(x))\) if, for any
\( \text{data instance} A \) and any tuple \( a \subseteq \text{ind}(A), \)
\( T,A \models q(a) \) iff \( \mathcal{I}_A \models q'(a), \)
where \( \mathcal{I}_A \) is the FO-structure over the domain \( \text{ind}(A) \)
such that \( \mathcal{I}_A \models S(a) \) iff \( S(a) \in A \), for any ground atom
\( S(a) \). If \( q'(x) \) is a positive existential formula, we call it a
\( \text{PE-rewriting of} \ Q(x) \). A PE-rewriting whose matrix is a \( \Pi_k \)-formula
(with respect to \( \land \) and \( \lor \) ) is called a \( \Pi_k \)-
rewriting. The size \( |q'| \) of \( q' \) is the number of symbols in
it.

We also consider rewritings in the form of nonrecursive
datalog queries. A \textit{datalog program}, \( \Pi \), is a finite set of
Horn clauses \( \forall x (\gamma_0 \land \cdots \land \gamma_m) \), where each \( \gamma_i \)
is an antecedent \( Q(y) \) with \( y \subseteq z \) or an equality
\( (z = z') \) with \( z, z' \subseteq z \). (As usual, we omit \( \forall z \) from clauses.)

The atom \( \gamma_0 \) is the head of the clause, and \( \gamma_1, \ldots, \gamma_m \) 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 \( \Pi \) are \( \text{IDB predicates} \), the rest
(including \( = \) ) \( \text{EDB predicates} \). A predicate \( Q \) depends
on \( P \) in \( \Pi \) if \( \Pi \) has a clause with \( Q \) in the head and \( P \)
in the body. \( \Pi \) is a \textit{nonrecursive} \textit{datalog} (\textit{NDL})
program if the (directed) \textit{dependence graph} of the dependence
relation is acyclic.

An \textit{NDL query} is a pair \( (\Pi,G(x)) \), where \( \Pi \) is an NDL
program and \( G(x) \) a predicate. A tuple \( a \subseteq \text{ind}(A) \)
is an \textit{answer to} \( (\Pi,G(x)) \) over \( \text{a data instance} A \) if \( G(a) \)
holds in the first-order structure with domain \( \text{ind}(A) \)
expanded by \( a \) under the clauses in \( \Pi \); in this case we write \( \Pi,A \models G(a) \). The problem of checking whether
\( a \) is an answer to \( (\Pi,G(x)) \) over \( A \) is called the
\textit{query evaluation problem}. The \textit{arity} of \( \Pi \) is the
maximal arity, \( r(\Pi) \), of predicates in \( \Pi \). The \textit{depth}
of \( (\Pi,G(x)) \) is the length, \( d(\Pi,G) \), of the longest directed
path in the dependence graph for \( \Pi \) starting from \( G \).

NDL queries are \textit{equivalent} if they have exactly the
same answers over any data instance.

An NDL query \( (\Pi,G(x)) \) is an \textit{NDL-rewriting} of an
OMQ \( Q(x) = (T,q(x)) \) over \text{complete} data instances.
In case \( T,A \models q(a) \) iff \( \Pi,A \models G(a) \), for any complete
\( A \) and any \( a \subseteq \text{ind}(A) \). Rewritings over \text{arbitrary}
data instances are defined by dropping the completeness \text{condition}.

Given an NDL-rewriting \( (\Pi,G(x)) \) of \( Q(x) \) over
\text{complete} data instances, we denote by \( \Pi^* \) the result of
replacing each predicate \( S \) in \( \Pi \) with a fresh IDB pred-
icate \( S^* \) of the same arity and adding the clauses
\begin{align*}
  A^*(x) & \leftarrow \tau(x), & \text{if } T \models \tau(x) \rightarrow A(x),
  \\
  P^*(x,y) & \leftarrow q(x,y), & \text{if } T \models q(x,y) \rightarrow P(x,y),
  \\
  P^*(x,x) & \leftarrow \top(x), & \text{if } T \models P(x,x),
\end{align*}
where \( \tau(x) \) is an EDB predicate for the active domain
\([29]\). Clearly, \( (\Pi^*,G(x)) \) is an NDL-rewriting of \( Q(x) \)
over \text{arbitrary} data instances and \( |\Pi^*| \leq |\Pi| + |\Pi|^2 \).

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

\section{Optimal NDL-Rewritings}

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

\subsection{NL and LOGCFL fragments of NDL}

To simplify the analysis of non-Boolean NDL queries,
we are convenient to regard certain variables as parameters
to be instantiated with constants from the candidate answer.
Formally, an NDL query \( (\Pi,G(x_1, \ldots, x_n)) \) is called
\textit{ordered} if each of its IDB predicates \( Q \) comes with
\text{fixed variables} \( x_1, \ldots, x_k \) \( (1 \leq i_1 < \cdots < i_k \leq n) \),
called the \textit{parameters} of \( Q \), such that (i) every occurrence
of \( Q \) in \( \Pi \) is of the form \( Q(y_1, \ldots, y_m, x_{i_1}, \ldots, x_{i_k}) \),
(ii) the parameters of \( G \) are \( x_1, \ldots, x_n \), and (iii) \text{parameters}
of the head of every clause include all the \text{parameters}
of the predicates in the body. Observe that

\begin{example}

1. The NDL query \( (\Pi,G(x)) \), where
\begin{align*}
  \Pi = \{ G(x) \leftarrow R(x,y) \land Q(x), \ Q(x) \leftarrow R(y,x) \},
\end{align*}
\end{example}

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

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

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

\subsection{Linear NDL in NL}

An NDL program is \textit{linear} \([1]\) if the body of its every
clause contains at most one IDB predicate.

\begin{theorem}

For any \( w > 0 \), evaluation of \textit{linear} NDL
queries of width \( \leq w \) is \textit{NL-complete} for combined
complexity.

\end{theorem}
Proof. Let \((\Pi, G(x))\) be a linear NDL query. Deciding whether \(\Pi, A \models G(a)\) is reducible to finding a path to \(G(a)\) from a certain set \(X\) in the grounding graph \(G\) constructed as follows. The vertices of \(G\) are the IDB atoms of \(\Pi_A\), and \(G\) has an edge from \(Q(c)\) to \(Q'(c')\) iff \(\Pi_A\) contains \(Q(c') \leftarrow Q(c) \land S_1(e_1) \land \cdots \land S_k(e_k)\) with \(S_i(e_i) \in A\), for \(1 \leq i \leq k\) (we assume \(A\) contains all \(c = c_i\), for \(c \in \text{ind}(A)\)). The set \(X\) consists of all vertices \(Q(c)\) with IDB predicates \(Q\) being of in-degree 0 in the dependency graph of \(A\) for which there is a clause \(Q(c) \leftarrow S_1(e_1) \land \cdots \land S_k(e_k)\) in \(\Pi_A\) with \(S_i(e_i) \in A\) (\(1 \leq i \leq k\)). Bounding the width of \((\Pi, G)\) ensures that \(G\) 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. \qed

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 \(w > 0\). There is an \(L_{\text{NL}}\)-transducer that, for any linear NDL-rewriting \((\Pi, G(x))\) of an OMQ \(Q(x)\) over complete data instances with \(w(\Pi, G) \leq w\), computes a linear NDL-rewriting \((\Pi', G'(x))\) of \(Q(x)\) over arbitrary data instances such that \(w(\Pi', G') \leq w+1\).

3.1.2 Skinny NDL in LOGCFL

The complexity class LOGCFL can be defined in terms of nondeterministic auxiliary pushdown automata (NAuxPDAs) [14], which are nondeterministic Turing machines with an additional work tape constrained to operate as a pushdown store. Sudborough [53] proved that LOGCFL coincides with the class of problems that are solved by NAuxPDAs in logarithmic space and polynomial time (the space on the pushdown tape is not subject to the logarithmic bound). It is known that LOGCFL can also be defined in terms of logspace-uniform families of semi-unbounded fan-in circuits (where OR-gates have arbitrarily many inputs, and AND-gates two inputs) of polynomial size and logarithmic depth. Moreover, there is an algorithm that, given such a circuit \(C\), computes the output using an NAuxPDA in logarithmic space in the size of \(C\) and exponential time in the depth of \(C\) [55, pp. 392–397].

We call an NDL query \((\Pi, G)\) skinny if the body of any clause in \(\Pi\) has at most two\(^2\) atoms.

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

Proof. Define a monotone Boolean circuit \(C\) as follows: its output is \(G(a)\); for every atom \(\gamma\) in the head of a clause in \(\Pi_A\), we take an OR-gate whose output is \(\gamma\) and inputs are the bodies of the clauses with head \(\gamma\); for every such body, we take an AND-gate whose inputs are the atoms in the body. We set input \(\gamma\) to 1 iff \(\gamma \in A\). Clearly, \(C\) is a semi-unbounded fan-in circuit of depth \(O(d(\Pi, G))\) with \(O(|\Pi| \cdot |A|^{w(\Pi, G)})\) gates. Observing that our \(C\) 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. \qed

We use weight functions as a means of generalising skinny programs. A function \(\nu\) from the predicate names in \(\Pi\) to \(\mathbb{N}\) is a weight function for an NDL query \((\Pi, G(x))\) if \(\nu(Q) > 0\) and \(\nu(Q) \geq \nu(P_1) + \cdots + \nu(P_k)\), for any clause \(Q(z) \leftarrow P_1(z_1) \land \cdots \land P_k(z_k)\) in \(\Pi\). First, using the Huffman code, we show that any NDL query can be transformed into an equivalent skinny NDL query whose depth increases by a logarithm of the weight function.

Lemma 5. Any \((\Pi, G(x))\) with a weight function \(\nu\) is equivalent to a skinny \((\Pi', G(x))\) with \(\nu(\Pi') = O(\nu(\Pi)^2)\), \(w(\Pi', G) \leq w(\Pi, G)\) and \(d(\Pi', G) \leq d(\Pi, G) + \log \nu(\Pi)\).

Proof. The proof is by induction on \(d(\Pi, G)\). We take \(\Pi' = \Pi\) if \(d(\Pi, G) = 0\). Otherwise, let \(\Pi\) contain a clause \(Q(z) \leftarrow P_1(z_1) \land \cdots \land P_k(z_k)\). Suppose that, for each \(i\) (\(1 \leq i \leq k\)), we have an NDL query \((\Pi_i', P_i)\) equivalent to \((\Pi, P_i)\) and such that \(d(\Pi_i', P_i) \leq d(\Pi, P_i) + \log \nu(P_i)\)\(^-\):

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

In general, the Huffman tree is a binary tree with \(k\) leaves \(1, \ldots, k\), a root \(g\) and \(k-2\) internal nodes and such that the length of the path from \(g\) to any leaf \(i\) is \(\leq \lfloor \log(\nu(Q)/\nu(P_i)) \rfloor\). For each internal node \(v\) of the tree, we take a predicate \(P_v(z_v)\), where \(z_v\) is the union of \(z_u\) for all descendants \(u\) of \(v\); for the root \(g\), we take \(P_g(z_g) = G(z)\). Let \(\Pi'_v\) be the extension of the union of the \(\Pi'_i\) (\(1 \leq i \leq k\)) with clauses \(P_v(z_v) \leftarrow P_{u_1}(z_{u_1}) \land P_{u_2}(z_{u_2})\), 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{align*}
d(\Pi'_v, G) & \leq \max\{\lfloor \log(\nu(Q)/\nu(P_i)) \rfloor + d(\Pi'_i, P_i)\} \\
& \leq \max\{\log(\nu(Q)/\nu(P_i)) + d(\Pi, G) + \log \nu(P_i)\} \\
& = d(\Pi, G) + \log \nu(Q)\).
\end{align*}
\]

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

We now use Lemmas 4 and 5 to obtain the following:
Theorem 6. For any $c \geq 1$ and $w \geq 1$, evaluation of NDL queries $(I, G(x))$ having a weight function $\nu$ such that $d(I, G) + \log \nu(G) \leq c \log |I|$ and $w(I, G) \leq w$ is in LOGCFL for combined complexity.

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

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

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

3.2 LOGCFL rewritings for OMQ($d$, $t$, $\infty$)

Recall (see, e.g., [20]) that a tree decomposition of an undirected graph $G = (V,E)$ is a pair $(T, \lambda)$, where $T$ is an (undirected) tree and $\lambda$ 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 $(T, \lambda)$ is $\max_{t \in T} |\lambda(t)| - 1$. The treewidth of a graph $G$ is the minimum width over all tree decompositions of $G$. The treewidth of a CQ is the treewidth of its Gaiman graph.

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

\[
\begin{array}{cccccccc}
  & x_0 & r & x_1 & r & x_2 & r & x_3 & r & x_4 & r & x_5 & r & x_6 & r & x_7 \\
  \end{array}
\]

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

We define recursively a set $D$ of subtrees of $T$, a binary ‘predecessor’ relation $\prec$ on $D$, and a function $\sigma$ on $D$ indicating the splitting node. We begin by adding $T$ to $D$. Take any $D \subseteq D$ 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 $D_1, \ldots, D_k$. We set $\sigma(D) = t$ and, for $1 \leq i \leq k$, add $D_i$ to $D$ and set $D_i \prec D$; then, we apply the procedure recursively to each of $D_1, \ldots, D_k$. In Example 8 with $t$ splitting $T$, we have $\sigma(T) = t$, $D_1 \prec T$ and $D_2 \prec T$.

For each $D \subseteq D$, we recursively define a set of atoms

\[ q_D = \{ S(z) \in q \mid z \subseteq \lambda(\sigma(D)) \} \cup \bigcup_{D' \prec D} q_{D'} \]

By the definition of tree decomposition, $q_T = q$. Denote by $x_D$ the subset of $x$ that occurs in $q_D$. In Example 8, $x_D = \{x_0, x_7\}$, $x_{D_1} = \{x_0\}$ and $x_{D_2} = \{x_7\}$. Let $\partial D$ be the union of all $\lambda(t) \cap \lambda(t')$ for boundary nodes of $D$. For each $D \subseteq D$, we use $\partial D = \emptyset$, $\partial D_1 = \{x_3\}$ and $\partial D_2 = \{x_4\}$.

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

- if $z \in x$, then $w(z) = \varepsilon$;
- if $A(z) \in q$, then either $w(z) = \varepsilon$ or $w(z) = wq$ with $T \models \exists y q(y, x) \rightarrow A(x)$;
- if $P(y, z) \in q$, then one of the three conditions holds: (i) $w(y) = w(z) = \varepsilon$; (ii) $w(y) = w(z)$ and $T \models P(x, z)$; (iii) $T \models q(x, y) \rightarrow P(x, y)$ and either $w(z) = w(y)q$ or $w(y) = w(z)q$.
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 $x_D$ for a tuple of variables in the set $x_D$ (ordered in some way).

Also, given a tuple $a \in \text{ind}(A)^{|x_D|}$ and $x \in x_D$, we write $a(x)$ to refer to the component of $a$ that corresponds to $x$ (that is, the component with the same index).

We now define an NDL-rewriting of $Q(x) = (T, q(x))$. For any $D \in \mathcal{D}$ and type $w$ with $\text{dom}(w) = \partial D$, let $G^w_T(\partial D, x_D)$ be a fresh IBD predicate with parameters $x_D$ (note that $\partial D$ and $x_D$ may be disjoint).

For each type $s$ with $\text{dom}(s) = \lambda(\sigma(D))$ such that $s$ is compatible with $\sigma(D)$ and agrees with $w$ on their common domain, the NDL program $\Pi^\text{LOG}_Q$ contains

$$G^w_T(\partial D, x_D) \leftarrow A^{s^*} \land \bigwedge_{D' < D} G^w_T(s_t\cup w) \cap \partial D'((\partial D', x_D'),$$

where $(s \cup w) \cap \partial D'$ is the restriction of the union $s \cup w$ to $\partial D'$ (since $\text{dom}(s \cup w)$ covers $\partial D'$, the domain of the restriction is $\partial D'$), and $A^{s^*}$ is the conjunction of

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

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

(c) $A^*_q$, for $z$ with $s(z) = gw$, for some $w$.

The conjuncts in (a) ensure that atoms all of whose variables are assigned $\varepsilon$ hold in the data instance. The conjuncts in (b) ensure that if one variable in a binary atom is not mapped to $\varepsilon$, 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 $agw$, then $agw$ is indeed in the domain of $C_T, A$.

**Example 11.** With the query in Example 8, consider now the following ontology $T$:

$$P(x, y, ) \rightarrow S(x, y), \quad A^p(x) \leftarrow \exists y P(x, y),$$

$$P(x, y, ) \rightarrow R(y, x), \quad A^p_p(x) \leftarrow \exists y P(y, x)$$

(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: $s_1 = \{x_3 \rightarrow \varepsilon, x_4 \rightarrow \varepsilon\}$ and $s_2 = \{x_3 \rightarrow \varepsilon, x_4 \rightarrow P\}$. So we have $A^{s_1} = R(x_3, x_4)$ and $A^{s_2} = A^p(x_4) \land (x_3 = x_4)$. Thus, the predicate $G_T^s$ is defined by two clauses with the head $G_T^\varepsilon(x_0, x_7)$ and the following bodies:

$$G_T^{\varepsilon}(x_3, x_0) \land R(x_3, x_4) \land G_T^{s_2^*}(x_4, x_7),$$

$$G_T^{\varepsilon}(x_3, x_0) \land A^{p}(x_4) \land (x_3 = x_4) \land G_T^{s_2^*}\cap P(x_4, x_7),$$

for $s_1$ and $s_2$, respectively. Although $\{x_3 \rightarrow P, x_4 \rightarrow \varepsilon\}$ is also compatible with $t$, its predicate $G_T^{s_2}$ will have no definition in the rewriting, and hence can be omitted. The same is true of the other compatible types $\{x_3 \rightarrow \varepsilon, x_4 \rightarrow R\}$ and $\{x_3 \rightarrow R^-, x_4 \rightarrow \varepsilon\}$.

By induction on $\prec$, one can now show that $(\Pi^\text{LOG}_Q, G_T^s)$ is a rewriting of $Q(x)$; see Appendix A.3 for details.

Fix now $d$ and $t$, and consider $Q(x) = (T, q(x))$ from $\text{OMQ}(d, t, \infty)$. Let $T$ be a tree decomposition of $\aleph_0$ of treewidth $\leq t$ (by $s$), we can assume w.l.o.g. that $T$ has at most $|q|$ nodes. We take the following weight function: $\nu(G^s_T) = |D|$. Clearly, $\nu(G^s_T) \leq |Q|$. By Lemma 10, $w(\Pi^\text{LOG}_Q, G_T^s) \leq |\partial D| \leq 2(t + 1)$ and $d(\Pi^\text{LOG}_Q, G_T^s) \leq 2 |\log |T| = 2 \log \nu(G^s_T) \leq 2 \log |Q|$. Since $|\partial D| \leq |T|^2$ and there are at most $|T|^{2d(t+1)}$ options for $w$, there are polynomially many predicates $G^s_T$ and so $\Pi^\text{LOG}_Q$ is of polynomial size. Thus, by Corollary 7, the constructed NDL-rewriting over arbitrary data instances can be evaluated in $\text{LOGCFL}$. Finally, we note that a tree decomposition of treewidth $\leq t$ can be computed using an $\text{L}^\text{LOGCFL}$-transducer [24], and so the NDL-rewriting can also be constructed by an $\text{L}^\text{LOGCFL}$-transducer.

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

$$\text{poly}(|T|^d, |q|, |A|^t).$$

(4)

### 3.3 NL rewritings for OMQ($d, 1, t$)

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

Let $T$ be an ontology of finite depth $d$, and let $q(x)$ be a tree-shaped CQ with at most $t$ leaves. Fix one of the variables $q$ as root, and let $M$ be the maximal distance to a leaf from the root. For $0 \leq n \leq M$, let $z^n$ denote the set of all variables of $q$ at distance $n$ from the root; clearly, $|z^n| \leq t$. We call the $z^n$ slices of $q$ and observe that they satisfy the following: for every $P(z, z') \in q$ with $z \neq z'$, there exists $n < M$ such that either $z \in z^n$ and $z' \notin z^n+1$ or $z' \in z^n$ and $z \notin z^n+1$.

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

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

- if $z \in x$, then $w(z) = \varepsilon$;
- if $A(z) \in q$, then either $w(z) = \varepsilon$ or $w(z) = w_q$ with $T \models \exists y g(y, x) \rightarrow A(x)$;
- if $P(z, z') \in q$, then either $w(z) = \varepsilon$ or $T \models P(x, y)$.

If $w, s$ are types for $z^n$ and $z^n+1$, respectively, then we say $(w, s)$ is compatible with $(z^n, z^n+1)$ if $w$ is locally compatible with $z^n$, $s$ is locally compatible with $z^n$, and for every $P(z, z') \in q$ with $z \in z^n$ and $z' \in z^n+1$, one of the three conditions holds: $w(z) = s(z') = \varepsilon$, or $w(z) = s(z') \models T \models P(x, y)$, or $T \models g(y, x) \rightarrow P(x, y)$ with either $s(z') = w(z)q$ or $w(z) = s(z')q^\varepsilon$.
Consider the NDL program \( III^{L_{IN}} \) defined as follows. For every \( 0 \leq n < M \) and every pair of types \((w, s)\) that is compatible with \((z^n, z^{n+1})\), we include the clause
\[
G_n^w(z^n, x^n) \leftarrow A_{w/s}^n(z^n, z^{n+1}) \land G_{n+1}^w(z^{n+1}, x^{n+1}),
\]
where \( x^n \) are the parameters of \( G_n^w \) and \( A_{w/s}^n(z^n, z^{n+1}) \) is the conjunction of atoms \((a)-(c)\) as defined in Section 3.2, for the union \( w \cup s \). For every type \( w \) locally compatible with \( z^M \), we include the clause
\[
G_M^w(z_M, x_M) \leftarrow A_{w}^w(z^M).
\]
(Recall that \( z^M \) is a disjoint union of \( z_2^M \) and \( x^M \).) We use \( G \) with parameters \( x \) as the goal predicate and include \( G(x) \leftarrow G_0^w(z_0^0, x) \) for every predicate \( G_0^w \) occurring in the head of one of the preceding clauses.

By induction on \( n \), we show in Appendix A.4 that \( III^{L_{IN}}(G(x)) \) is a rewriting of \((\langle T, q(x) \rangle)\) over complete data instances. It should be clear that \( III^{L_{IN}} \) can be computed by an \( L_{NL} \)-transducer. We apply Lemma 3 to obtain an NDL-rewriting for arbitrary data instances, and then use Theorem 2 to conclude that the resulting program can be evaluated in \( NL \).

The obtained NDL-rewriting implies that answering \((\langle T, q(x) \rangle)\) with \( \ell \) of finite depth \( d \) and tree-shaped \( q \) with \( \ell \) leaves over any data \( A \) can be done in time
\[
\text{poly}(|T|^d, |q|, |A|^{\ell}).
\]

3.4 LOGCFL rewritings for OMQ(\(\infty, 1, \ell\))

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

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

For OMQs with bounded-leaf CQs and ontologies of unbounded depth, our rewriting uses the notion of tree witness [34]. Consider an OMQ \( Q(x) = (\langle T, q(x) \rangle) \). Let \( t = (t_r, t_l) \) be a pair of disjoint sets of variables in \( q \) such that \( t_l \neq \emptyset \) but \( t_r \cap x = \emptyset \). Set
\[
q_t = \{ S(z) \in q \mid z \subseteq t_r \cup t_l \text{ and } z \nsubseteq t_r \}.
\]
If \( q_t \) is a minimal subset of \( q \) containing every atom of \( q \) with a variable from \( t_r \), and such that there is a homomorphism \( h : q_t \to G_T(A_{q_t}(x)) \) with \( h^{-1}(a) = t_r \), we call \( t \) a tree witness for \( Q(x) \) generated by \( q \). Intuitively, \( t \) identifies a minimal subset of \( q \) that can be mapped to the tree-shaped part of the canonical model consisting of labelled nulls: the variables in \( t_r \) are mapped to an individual constant, say, \( a \), at the root of a tree and the \( t_l \) are mapped to the labelled nulls of the form \( aw \), for some \( w \in W_T \) that begins with \( a \). Note that the same tree witness can be generated by different \( q \).

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

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

Let \( Q(x_0) = (\langle T, q_0(x_0) \rangle) \) be an OMQ with a tree-shaped CQ. We will repeatedly apply Lemma 14 to decompose the CQ into smaller and smaller subqueries. Formally, for a tree-shaped CQ \( q \), we denote by \( z_q \) a vertex in the Gaifman graph \( G \) of \( q \) that satisfies the condition of Lemma 14; if \(|\text{var}(q)| = 2 \) and \( q \) has at least one existentially quantified variable, then we assume that \( z_q \) is such. Let \( \Omega \) be the smallest set that contains \( q_0(x_0) \) and the following CQs, for every \( q(x) \in \Omega \) with existentially quantified variables:

- for each \( z_i \) adjacent to \( z_q \) in \( G \), the CQ \( q_i(x_i) \) comprising all binary atoms with both \( z_i \) and \( z_q \), and all atoms whose variables cannot reach \( z_q \) in \( G \) without passing by \( z_i \), where \( x_i \) is the set of variables in \( x \cup \{z_i \} \) that occur in \( q_i \);

- for each tree witness \( t \) for \((\langle T, q(x) \rangle)\) with \( t_r \neq \emptyset \) and \( z_q \in t_r \), the CQs \( q^1_t(x^1_t) \cdots q^k_t(x^k_t) \) that correspond to the connected components of the set of atoms of \( q \) that are not in \( q^1_t \), where each \( x^i_t \) is the set of variables in \( x \cup t_l \) that occur in \( q^i_t \).

The two cases are depicted below:

![Diagram](image)

Note that \( t_r \neq \emptyset \) ensures that part of the query without \( q_t \) is mapped onto individual constants.

The NDL program \( III^{T \downarrow} \) uses IDB predicates \( G_q(x) \), for \( q(x) \in \Omega \), whose parameters are the variables in \( x_0 \) that occur in \( q(x) \). For each \( q(x) \in \Omega \), if it has no existentially quantified variables, then we include the clause \( G_q(x) \leftarrow q(x) \). Otherwise, we include the clause
\[
G_q(x) \leftarrow \bigwedge S(z) \land \bigwedge_{1 \leq i \leq n} G_{q_i}(x_i),
\]
where \( q_1(x_1), \ldots, q_n(x_n) \) are the subqueries induced by the neighbours of \( z_q \) in \( G \), and, for each tree witness \( t \) for \((\langle T, q(x) \rangle)\) with \( t_r \neq \emptyset \) and \( z_q \in t_r \) and for every \( q \) generating \( t \), the following clause
\[
G_q(x) \leftarrow A_q(z_0) \land \bigwedge_{z \in t_r \setminus \{z_0\}} (z = z_0) \land \bigwedge_{1 \leq i \leq k} G_{q_i}(x^i_t),
\]
where \( z_0 \) is any variable in \( t_r \) and \( q^1_t, \ldots, q^k_t \) are the connected components of \( q \) without \( q_t \). Finally, if \( q_0 \) is Boolean, then we include clauses \( G_{q_0} \leftarrow A(x) \) for all unary predicates \( A \) such that \( T, \{A(a)\} \models q_0 \).

The program \( III^{T \downarrow} \) is inspired by a similar construction from [31]. By adapting the proof, we can show that \( III^{T \downarrow}(G_q(x_0)) \) is indeed a rewriting; see Appendix A.5.

Now fix \( \ell > 1 \) and consider \( Q(x) = (\langle T, q(x) \rangle) \) from the class OMQ(\(\infty, 1, \ell\)). The size of \( III^{T \downarrow} \) is polynomially bounded in \(|Q| \) since \( q \) has \( O(|q|^\ell) \) tree witnesses.
and tree-shaped subqueries. It is readily seen that the function \( n \) defined by setting \( n(G_q) = |q| \) is a weight function for \( (\Pi^T_{Q_0}, G_{q_0}) \) with \( n(G_{q_0}) \leq |Q| \). Moreover, by Lemma 14, \( d(\Pi, G_q) \leq \log n(G_q) + 1 \); and clearly, \( w(\Pi^T_{Q_0}, G_{q_0}) \leq \ell + 1 \). By Corollary 7, the obtained NDL-rewritings can be evaluated in LOGCFL. Finally, we note that since the number of leaves is bounded, it is in \( NL \) to decide whether a vertex satisfies the conditions of Lemma 14, and in LOGCFL to decide whether \( \mathcal{T}, \{A(a)\} \models q_0 \) [5] or whether a (logspace) representation of a possible tree witness is indeed a tree witness. This allows us to show that \( (\Pi^T_{Q_0}, G_{q_0}) \) can be generated by an \( LOGCFL \)-transducer. It also follows that answering OMQs \( (\mathcal{T}, q(x)) \) with a tree-shaped CQ with \( \ell \) leaves over any data instance \( A \) can be done in time
\[
poly(|\mathcal{T}|, |q|, |A|). \tag{6}
\]

4. PARAMETERISED COMPLEXITY

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

4.1 Ontology Depth

Consider the following problem \( p\text{Depth}-\text{TreeOMQ} \):

**Instance:** an OMQ \( Q = (\mathcal{T}, q) \) with \( \mathcal{T} \) of finite depth and tree-shaped Boolean CQ \( q \).

**Parameter:** the depth of \( \mathcal{T} \).

**Problem:** decide whether \( \mathcal{T}, \{A(a)\} \models q \).

**Theorem 15.** \( p\text{Depth}-\text{TreeOMQ} \) is \( W[2]-\text{hard} \).

**Proof.** The proof is by reduction of the problem \( p\text{-HittingSet} \), which is known to be \( W[2]-\text{complete} \) [20]:

**Instance:** a hypergraph \( H = (V, E) \) and \( k \in \mathbb{N} \).

**Parameter:** \( k \).

**Problem:** decide whether there is \( A \subseteq V \) such that \(|A| = k \) and \( e \cap A \neq \emptyset \), for every \( e \in E \).

(Such a set \( A \) of vertices is called a hitting set of size \( k \)).

Suppose that \( H = (V, E) \) is a hypergraph with vertices \( V = \{v_1, \ldots, v_n\} \) and hyperedges \( E = \{e_1, \ldots, e_m\} \). Let \( \mathcal{T}^k \) be the (normal form of an) ontology with the following axioms, for \( 1 \leq l \leq k \):

\[
V^1_l(x) \rightarrow \exists z \left( P(z, x) \land V^i_l(z) \right), \quad \text{for } 0 \leq i < i' \leq n,
\]

\[
V^i_l(x) \rightarrow E^i_j(x), \quad \text{for } v_i \in e_j, \; e_j \in E,
\]

\[
E^i_j(x) \rightarrow \exists z \left( P(x, z) \land E^{i-1}_j(z) \right), \quad \text{for } 1 \leq j \leq m.
\]

Let \( q^k \) be a tree-shaped Boolean CQ with the following atoms, for \( 1 \leq l \leq m \):

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

The first axiom of \( \mathcal{T}^k \) 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 \( q^k \) 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 \( (\mathcal{T}^k, \{V^0_l(a)\}) \) and the CQ \( q^k \), for \( H = (V, \{e_1, e, e_3\}) \) with \( V = \{1, 2, 3\}, e_1 = \{1, 3\}, e_2 = \{2, 3\} \) and \( e_3 = \{1, 2\} \), is shown below:

Points \( \Box \) at level \( l \) belong to \( V^l_1 \). By Appendix B.1 we prove that \( \mathcal{T}^k \), \( \{V^0_l(a)\} \models q^k \) 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 \( q^k \) into the part of \( \mathcal{T}^k \) shown in black.

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

4.2 Number of Leaves

Next we consider the problem \( p\text{Leaves}-\text{TreeOMQ} \):

**Instance:** an OMQ \( Q = (\mathcal{T}, q) \) with \( \mathcal{T} \) of finite depth and tree-shaped Boolean CQ \( q \).

**Parameter:** the number of leaves in \( q \).

**Problem:** decide whether \( \mathcal{T}, \{A(a)\} \models q \).

**Theorem 16.** \( p\text{Leaves}-\text{TreeOMQ} \) is \( W[1]-\text{hard} \).

**Proof.** The proof is by reduction of the following \( W[1]-\text{complete} \) \( \text{PartitionedClique} \) problem [19]:

**Instance:** a graph \( G = (V, E) \) whose vertices are partitioned into \( p \) sets \( V_1, \ldots, V_p \).

**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 \( V = \{v_1, \ldots, v_M\} \) partitioned into \( V_1, \ldots, V_p \). 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 \( i \)th block of a branch, we will select a vertex \( v_{ij} \) from \( V_i \) by marking the positions \( 2j \) and \( 2j + 1 \) with the binary predicate \( S \); we also mark the positions of...
the neighbours of \( v_i \) in \( G \) with the predicate \( Y \). We use the unary predicate \( B \) to mark the end of the \( p \)th 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 \( v_j \), for \( V_i \) is a neighbour of the vertices selected from \( V_{i+1}, \ldots, V_p \), it suffices to check that positions \( 2j_1 \) and \( 2j_1 + 1 \) in blocks \( i + 1, \ldots, p \) 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 \( q_G \) (right side of the picture) which, starting from a variable labelled \( B \) (mapped to the end of a \( p \)th block), splits into \( p - 1 \) branches, with the \( i \)th branch checking for a sequence of \( i \) evenly-spaced \( YY \) markers leading to an \( SS \) marker. The distance from the end of the \( p \)th block (marked \( B \)) to the positions \( 2j_1 \) and \( 2j_1 + 1 \) in the \( p \)th block (where the first \( YY \) should occur) depends on the choice of \( v_j \). We thus add an outgoing edge at the end of the \( p \)th 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 \( q_G \) looks as follows (for readability, we use atoms with star-free regular expressions):

\[
B(y) \land \bigwedge_{1 \leq i < p} (U^{2M-2} \cdot (YY \cdot U^{2M-2}) \cdot SS)(y, z_i),
\]

and the ontology \( T_G \) contains the following axioms:

\[
A(x) \rightarrow \exists y L^1_k(x, y), \quad \text{for } v_j \in V_1, \quad \exists z L^k_j(z, x) \rightarrow \exists y L^{k+1}_j(x, y), \quad \text{for } 1 \leq k < 2M, \quad v_j \in V, \\
\exists z L^k_j(z, x) \rightarrow \exists y L^k_j(x, y), \quad \text{for } 1 \leq k < 2M, \quad v_j \in V, \quad v_j' \in V_{i+1}, \\
L^k_j(x, y) \rightarrow S(y, x), \quad \text{for } k \in \{2j, 2j + 1\}, \\
L^k_j(x, y) \rightarrow Y(y, x), \quad \text{for } \{v_j, v_j'\} \in E \quad \text{and } k \in \{2j', 2j' + 1\}, \\
L^k_j(x, y) \rightarrow U(y, x), \quad \text{for } 1 \leq k \leq 2M, \quad v_j \in V, \\
\exists z L^{2M}_j(z, x) \rightarrow B(x), \quad \text{for } v_j \in V_p, \\
B(x) \rightarrow \exists y (U(x, y) \land U(y, x)).
\]

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

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

5. OMQs with a Fixed Ontology

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

**Theorem 17.** There is an ontology \( T_1 \) such that answering OMQs of the form \( (T_1, q) \) with Boolean tree-shaped CQs \( q \) is NP-hard for query complexity.

**Proof.** The proof is by reduction of SAT. Given a CNF \( \varphi \) with variables \( p_1, \ldots, p_k \) and clauses \( \chi_1, \ldots, \chi_m \), take a Boolean CQ \( q_\varphi \) with \( A(y) \) and, for \( 1 \leq j \leq m \), the following atoms with \( z_j = y: \)

\[
P_+(z_j, z_j'-1), \quad \text{if } p_i \text{ occurs in } \chi_j \text{ positively,} \\
P_-(z_j, z_j'-1), \quad \text{if } p_i \text{ occurs in } \chi_j \text{ negatively,} \\
P_0(z_j, z_j'-1), \quad \text{if } p_i \text{ does not occur in } \chi_j, \\
B_0(z_j'^j).\]

Thus, \( q_\varphi \) is a star with centre \( A(y) \) and \( m \) rays encoding the \( \chi_j \) by the binary predicates \( P_+, P_- \) and \( P_0 \). Let \( T_1\) be an ontology with the axioms

\[
A(x) \rightarrow \exists y (P_+(y, x) \land P_0(y, x) \land B_-(y) \land A(y)), \\
B_- \rightarrow \exists y' (P_-(y', x') \land B_0(x')) , \\
A(x) \rightarrow \exists y (P_-(y, x) \land P_0(y, x) \land B_+(y) \land A(y)), \\
B_+ \rightarrow \exists y' (P_+(y, x') \land B_0(x')) , \\
B_0 \rightarrow \exists y (P_+(y, x) \land P_-(x, y) \land P_0(x, y) \land B_0(y)).
\]

Intuitively, \( (T_1, \{A(a)\}) \) generates an infinite binary tree whose nodes of depth \( n \) represent all \( 2^n \) truth assignments to \( n \) propositional variables. The CQ \( q_\varphi \) 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 \( \varphi \). Each non-root node of the tree also starts an infinite ‘sink’ branch of \( B_0 \)-nodes, where the remainder of the ray for \( \chi \) can be mapped as soon as one of its literals is satisfied. We show in Appendix C.1 that \( T_1, \{ A(a) \} \models q_\varphi \) iff \( \varphi \) is satisfiable. To illustrate, the CQ \( q_\varphi \) for \( \varphi = (p_1 \vee p_2) \land \neg p_1 \) and a fragment of the canonical model \( C_{T_1, \{ A(a) \}} \) are shown below:

Here, \( \bullet \) 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 \( C_{T_1, \{ A(a) \}} \).

The proof above uses OMQs \( Q_\varphi = (T_1, q_\varphi) \) over a data instance with a single individual constant. Thus:

**Corollary 18.** No polynomial-time algorithm can construct FO- or NDL-rewritings for the OMQs \( Q_\varphi \) unless \( P = NP \).

**Proof.** Indeed, if a polynomial-time algorithm could find a rewriting \( q_\varphi' \) of \( Q_\varphi \), then we would be able to check whether \( \varphi \) is satisfiable in polynomial time by evaluating \( q_\varphi' \) over the data instance \( \{ A(a) \} \).

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

**Theorem 19.** The \( Q_\varphi \) have polynomial FO-rewritings.

**Proof.** Define \( q_\varphi' \) as the FO-sentence

\[
\forall xy ((x = y) \land A(x) \land \varphi^*) \lor \exists xy ((x \neq y) \land q_\varphi^*(x, y)),
\]

where \( \varphi^* \) is \( T \) if \( \varphi \) is satisfiable and \( \bot \) otherwise, and \( q_\varphi^*(x, y) \) is the polynomial-size FO-rewriting of \( q_\varphi \) over data with at least 2 constants [23, Corollary 14]. Recall that the proof of Theorem 17 shows that, if \( A \) has a single constant, \( a \), and there is a homomorphism from \( q_\varphi \) to \( C_{T_1, A} \), then \( A(a) \in A \) and \( \varphi \) is satisfiable. Thus, the first disjunct of \( q_\varphi' \) is an FO-rewriting of \( Q_\varphi \) over data instances with a single constant; the case of at least 2 constants follows from [23, Corollary 14].

Whether the OMQs \( Q_\varphi \) have a polynomial-size PE- or NDL-rewritings remains open. We have only managed to construct a modification \( q_\varphi^*(x) \) of \( q_\varphi \) with the following interesting properties (details are given in Appendix C.2). Let \( \mathfrak{S} \) 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 \( QL \) be any query language such that, for every \( QL \)-query \( \Phi(x) \) and every \( A \in \mathfrak{S} \), the answer to \( \Phi(a) \) over \( A \) can be computed in time polynomial in \( |\Phi| \) and \( |A| \). Typical examples of \( QL \) are modal-like languages such as certain fragments of XPath [35] or description logic instance queries [4].

**Theorem 20.** The OMQs \( (T_1, q_\varphi(x)) \) do not have polynomial-size rewritings in \( QL \) unless \( NP \subseteq P/\text{poly} \).

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

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

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

**Theorem 22.** There is an ontology \( T_1 \) such that answering OMQs of the form \( (T_1, q) \) with Boolean linear CQs \( q \) is LOGCFL-hard for query complexity.

The proof is by reduction of the recognition problem for the hardest LOGCFL language \( L \) [26, 52]. We construct an ontology \( T_1 \) and a logspace transducer that converts the words \( w \) in the alphabet of \( L \) to linear CQs \( q_w \) such that \( w \in \mathcal{L} \) iff \( T_1, \{ A(a) \} \models q_w \).

6. EXPERIMENTS & CONCLUSIONS

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

The starting point of our research was the observation that standard query rewriting systems tend to produce suboptimal rewritings of the OMQs in these three classes. This is obviously so for UCQ-rewriters [46, 43, 13, 25, 40, 36]. However, this is also true of more elaborate PE-rewriters (which use disjunctions inside conjunctions) [47, 54] whose rewritings in theory can be...
of superpolynomial size; see Fig. 1(b). Surprisingly, even NDL-rewriters such as Clipper [18], Presto [49] and Rapid [13] do not fare much better in practice.

To illustrate, we generated three sequences of OMQs in the class $OMQ(1,1,2)$ (lying in the intersection of $OMQ(d,t,\infty)$, $OMQ(d,1,\ell)$ and $OMQ(\infty,1,\ell)$) with the ontology from Example 11 and linear CQs of up to 15 atoms as in Example 8 (which are associated with words from $\{R,S\}^\ast$). By Fig. 1(a), answering these OMQs can be done in NL. The barcharts in Fig. 2 show the number of clauses in their NDL-rewritings produced by Clipper, Presto and Rapid, as well as by our algorithms Lin, Log and Tw from Sections 3.2–3.4, respectively. The first three NDL-rewritings display a clear exponential growth, with Clipper and Rapid failing to produce rewritings for longer CQs. In contrast, our rewritings grow linearly in accord with theory.

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

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

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

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

A more general avenue for future research is to extend the study of succinctness and optimality of rewritings to suitable ontology languages with predicates of higherarity, such as linear and sticky tgdss.
7. REFERENCES


S. Kikot, R. Kontchakov, and M. Zakharyaschev. Sound, complete and minimal
S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Kaminski, Y. Nenov, and B. Cuenca Grau.
E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, E. Kharlamov, D. Hovland, E. Jiménez-Ruiz,
M. Zakharyaschev. Exponential lower bounds and separation for query rewriting.


APPENDIX

A. PROOFS FOR SECTION 3

A.1 Lemma 3

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

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

\[
Q(z) \leftarrow I \land EQ \land E_1 \land \ldots \land E_n,
\]

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

\[
v(E_i) = \begin{cases} 
B(z) | T \models B(x) \rightarrow A(x) \} \cup \\
\{ \phi(y_i, z) | T \models \exists y \phi(y, x) \rightarrow A(x) \}, & \text{if } E_i = A(z), \\
\{ \phi(z, z') | T \models \phi(x, y) \rightarrow P(x, y) \}, & \text{if } E_i = P(z, z'), 
\end{cases}
\]

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

\[
Q_0(z_0) \leftarrow I,
\]

\[
Q_{i+1}(z_i) \leftarrow H_i(z_i) \land E'_{i}, \quad \text{for } 1 \leq i \leq n \text{ and } E'_{i} \in v(E_i),
\]

\[
Q(z) \leftarrow H_{n+1}(z_n) \land EQ,
\]

where \( z_i \) is the restriction of \( z \) to variables occurring in \( I \) if \( i = 0 \) and in \( Q_i(z_i) \) and \( E'_{i} \) except for \( y_i \) if \( i > 0 \) (note that \( z_n = z \)). Let \( \Pi' \) be the program obtained from \( \Pi \) by replacing each clause \( \lambda \) by the set of clauses \( \lambda^* \). By construction, \( \Pi' \) is a linear NDL program and its width cannot exceed \( w(\Pi, G) + 1 \) (the possible increase of 1 is due to the replacement of unary atoms \( A(z) \) by binary atoms \( \phi(y_i, z) \)).

We now argue that \((\Pi', G(x))\) is a rewriting of \( Q(x) \) over arbitrary data instances. It can be easily verified that \((\Pi', G(x))\) is equivalent to \((\Pi', G(x))\), where NDL program \( \Pi'' \) is obtained from \( \Pi \) by replacing each clause \( Q(z) \leftarrow I \land EQ \land E_1 \land \ldots \land E_n \) by the (possibly exponentially larger) set of clauses of the form

\[
Q(z) \leftarrow I \land EQ \land E'_1 \land \ldots \land E'_n,
\]

for all \( E'_{i} \in v(E_i) \) and \( 1 \leq i \leq n \). It thus suffices to show that \((\Pi', G(x))\) is a rewriting of \( Q(x) \) over arbitrary data instances.

First suppose that \( T, A \models q(a) \), where \( A \) is an arbitrary data instance. Let \( A' \) be the complete data instance obtained from \( A \) by adding the ground atoms:

\[
P(a, b) \quad \text{if } \quad g(a, b) \in A \text{ and } T \models g(x, y) \rightarrow P(x, y);
\]

\[
A(a) \quad \text{if } \quad B(a) \in A \text{ and } T \models B(x) \rightarrow A(x);
\]

\[
A(a) \quad \text{if } \quad g(a, b) \in A \text{ and } T \models \exists y \phi(y, x) \rightarrow A(x).
\]

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

- if \( E_i = A(z) \), then \( A(c(z)) \in A' \), so there must exist either a unary ground atom \( B(c(z)) \in A \) such that \( T \models B(x) \rightarrow A(x) \) or a binary ground atom \( g(a, c(z)) \in A \), for some \( a \in \text{ind}(A) \), such that \( T \models \exists y \phi(y, x) \rightarrow A(x) \); in the latter case, we set \( c'(y) = a \);

- similarly, if \( E_i = P(z, z') \), then there must exist a binary ground atom \( g(c(z), c(z')) \in A \) such that \( T \models \phi(x, y) \rightarrow P(x, y) \).

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

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

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

\[ \square \]

A.2 Theorem 6

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

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

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

\[ \square \]

A.3 Log-re rewritings

**Lemma 23.** For any complete data instance \( A \), any \( D \in \Delta \), any type \( w \) with \( \text{dom}(w) = \partial D \) and any tuples \( b \in \text{ind}(A)^{|\partial D|} \) and \( a \in \text{ind}(A)^{|\partial a|} \), we have
\[ \Pi_{Q}^{\Log}, A \models G_{w}^{w}(b, a) \iff \text{there is a homomorphism } h : q_{D} \to C_{T, A} \text{ such that } \\
\quad h(x) = a(x), \quad \text{for } x \in x_{D}, \\
\quad \text{and } h(z) = b(z)w(z), \quad \text{for } z \in \partial D. \quad (7) \]

**Proof.** (\(\Rightarrow\)) The proof is by induction on \(\prec\). For the basis of induction, let \(D\) be of size 1. By the definition of \(\Pi_{Q}^{\Log}\), there exists a type \(s\) such that \(\text{dom}(s) = \lambda(\sigma(D))\) and \(w\) agrees with \(s\) on \(\partial D\) and a respective tuple \(c \in \text{ind}(A)^{\lambda(\sigma(D))}\) such that \(c(z) = b(z)\), for all \(z \in \partial D\), and \(c(x) = a(x)\), for all \(x \in x_{D}\), and \(\Pi_{Q}^{\Log}, A \models \text{At}^{s}(c)\). Then, for any atom \(S(z) \in q_{D}\), we have \(z \subseteq \lambda(\sigma(D))\), whence \(C_{T, A} \models S(h(z))\) as \(w\) agrees with \(s\) on \(\partial D\).

For the inductive step, suppose that we have \(\Pi_{Q}^{\Log}, A \models G_{w}^{w}(b, a)\). By the definition of \(\Pi_{Q}^{\Log}\), there exists a type \(s\) such that \(\text{dom}(s) = \lambda(\sigma(D))\) and \(w\) agrees with \(s\) on their common domain and a respective tuple \(c \in \text{ind}(A)^{\lambda(\sigma(D))}\) such that \(c(z) = b(z)\), for all \(z \in \partial D\), and \(c(x) = a(x)\), for all \(x \in x_{D}\), and

\[ \Pi_{Q}^{\Log}, A \models \text{At}^{s}(c) \land \bigwedge_{D' \prec D} G_{D'}^{D''}(b_{D'}, a_{D''}), \]

where \(b_{D'}\) and \(a_{D''}\) are the restrictions of \(b \cup \partial c\) to \(\partial D'\) and of \(a\) to \(x_{D'}\), respectively. By the inductive hypothesis, for any \(D' < D\), there is a homomorphism \(h_{D'} : q_{D'} \to C_{T, A}\) such that (7) is satisfied.

Let us show that the \(h_{D}\) agrees on common variables. Suppose that \(z\) is shared by \(q_{D'}\) and \(q_{D''}\), for \(D' < D\) and \(D'' < D\). By the definition of tree decomposition, for every \(z \in V\), the nodes \(\{t \mid z \in \lambda(t)\}\) induce a connected subtree of \(T\), and so \(z \in \lambda(\sigma(D)) \land \lambda(t') \land \lambda(t'')\), where \(t'\) and \(t''\) are the unique neighbours of \(\sigma(D)\) lying in \(D'\) and \(D''\), respectively. Since \(w' = (w \cup s) \restriction \partial D'\) and \(w'' = (w \cup s) \restriction \partial D''\) are the restrictions of \(w \cup s\), we have \(w'(z) = w''(z)\). This implies that

\[ h_{D'}(z) = c(z)w'(z) = c(z)w''(z) = h_{D''}(z). \]

Now we define \(h\) on every \(z\) in \(q_{D}\) by taking

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

If follows that \(h\) is well defined, \(h\) satisfies (7) and that \(h\) is a homomorphism from \(q_{D}\) to \(C_{T, A}\). Indeed, take an atom \(S(z) \in q_{D}\). Then either \(z \in \lambda(\sigma(D))\), in which case \(C_{T, A} \models S(h(z))\) since \(w\) is compatible with \(\sigma(D)\) and \(\Pi_{Q}^{\Log}, A \models \text{At}^{s}(c)\), or \(S(z) \in q_{D'}\) for some \(D' < D\), in which case we use the fact that \(h\) extends a homomorphism \(h_{D'}\).

(\(\Leftarrow\)) The proof is by induction on \(\prec\). Fix \(D\) and \(w\) such that \(|w| = |\partial D|\). Take tuples \(b \in \text{ind}(A)^{|\partial D|}\) and \(a \in \text{ind}(A)^{|\partial D|}\), and a homomorphism \(h : q_{D} \to C_{T, A}\) satisfying (7). Define a type \(s\) and a tuple \(c \in \text{ind}(A)^{\lambda(\sigma(D))}\) by taking, for all \(z \in \lambda(\sigma(D))\),

\[ s(z) = w \quad \text{and} \quad c(z) = a, \quad \text{if } h(z) = aw, \quad \text{for } a \in \text{ind}(A). \]

By definition, \(\text{dom}(s) = \lambda(\sigma(D))\) and, by (7), \(s\) and \(w\) agree on the common domain. For the inductive step, for each \(D' \prec D\), let \(h_{D'}\) be the restriction of \(h\) to \(q_{D'}\) and let \(b_{D'}\) and \(a_{D'}\) be the restrictions of \(b \cup c\) to \(\partial D'\) and of \(a\) to \(x_{D'}\), respectively. By the inductive hypothesis, \(\Pi_{Q}^{\Log}, A \models G_{w}^{w}(b_{D'}, a_{D'})\). (This argument is not needed for the basis of induction.) Since \(h\) is a homomorphism, we have \(\Pi_{Q}^{\Log}, A \models \text{At}^{s}(c)\), whence \(\Pi_{Q}^{\Log}, A \models G_{w}^{w}(b, a)\).

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

\[ \text{poly}(|T|^{dt}, |q|, |A|^{t}). \quad (4) \]

Indeed, we can evaluate \(\Pi_{Q}^{\Log}(G_{T}(x))\) in time polynomial in \(|\Pi_{Q}^{\Log}|\) and \(|A|^{|\Pi_{Q}^{\Log}|}\), which are bounded by a polynomial in \(|T|^{2d(t+1)}, |q|\) and \(|A|^{2t(t+1)}\).

**A.4 Lin-rewritings**

**Lemma 24.** For any complete data instance \(A\), any predicate \(G_{w}^{w}\), any \(a \in \text{ind}(A)^{|\partial D|}\) and \(b \in \text{ind}(A)^{|\partial D|}\), we have \(\Pi_{Q}^{\Lin}, A \models G_{w}^{w}(b, a)\) if there is a homomorphism \(h : q_{n} \to C_{T, A}\) such that

\[ h(x) = a(x), \quad \text{for } x \in x^{n}, \\
\quad \text{and} \quad h(z) = b(z)w(z), \quad \text{for } z \in z^{n}. \quad (8) \]

**Proof.** The proof is by induction on \(n\).

For the base case \((n = M)\), first suppose that we have \(\Pi_{Q}^{\Lin}, A \models G_{w}^{w}(b, a)\). The only rule in \(\Pi_{Q}^{\Lin}\) with head predicate \(G_{w}^{w}\) is \(G_{w}^{w}(z_{M}, x^{M}) \leftarrow \text{At}^{w}(z_{M})\) with \(z_{M} = z_{M}^{M} \cup x^{M}\), which is equivalent to

\[ G_{w}^{w}(z_{M}, x^{M}) \leftarrow \bigwedge_{z \in z_{M}} \left( \bigwedge_{A(z) \in q} A(z) \land \bigwedge_{P(z, z) \in q} P(z, z) \land \right) \bigwedge_{w(z) = g_{w}} A_{g}(z). \quad (9) \]

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

- \(A(a(x)) \in A\) for every \(A(x) \in q\) such that \(x \in x^{M}\);
- \(A(b(z)) \in A\) for every \(A(z) \in q\) such that \(z \in z_{M}\) and \(w(z) = \varepsilon\);
- \(P(a(x), a(x)) \in A\) for every \(P(x, x) \in q\) such that \(x \in x^{M}\);
- \(P(b(z), b(z)) \in A\) for every \(P(z, z) \in q\) such that \(z \in z_{M}\) and \(w(z) = \varepsilon\);
- \(A_{g}(z) \in A\) for every \(z \in z_{M}\) with \(w(z) = g_{w}\).

Now let \(\tilde{h}^{M}\) be the unique mapping from \(z_{M}\) to \(\Delta^{C_{T, A}}\) satisfying (8). First note that \(\tilde{h}^{M}\) is well-defined, since by the last item, if \(w(z) = g_{w}\), then we have \(A_{g}(z) \in A\).
and $gw \in W_\tau$, so $b(z)gw$ belongs to $\Delta^c_{\tau,A}$. To show that $h_M$ is a homomorphism of $q_M$ into $C_{\tau,A}$, first recall that the atoms of $q_M$ are of two types: $A(z)$ or $P(z, z)$, with $z \in z_M$. Take some $A(z) \in q_M$. If $w(z) = \varepsilon$, then we immediately obtain either $A(h_M(z)) = A(a(z)) \in A$ or $A(h_M(z)) = A(b(z)) \in A$, depending on whether $z \in z_M^+$ or in $x^M$. Otherwise, if $w(z) \neq \varepsilon$, then the local compatibility of $w$ with $z_M$ means that the final letter $q$ in $w(z)$ is such that $T \models \exists y \theta(y, x) \to A(x)$, hence $h_M(z) = b(z)w(z) \in A^c_{\tau,A}$. Finally, suppose that $P(z, z) \in q$. The local compatibility of $w$ with $z_M$ ensures that either $w(z) = \varepsilon$ or $T \models P(x, x)$. In the former case, we have either $P(a(z), a(z)) \in A$ or $P(b(z), b(z)) \in A$, depending again on whether $z \in z_M^+$ or $z \in x^M$. In the latter case, $(h_M(z), h_M(z)) \in P^c_{\tau,A}$.

For the other direction, $(\Rightarrow)$, of the base case, suppose that the mapping $h_M$ given by $(8)$ defines a homomorphism from $q_M$ into $C_{\tau,A}$. We therefore have:

- $a(z) \in A^c_{\tau,A}$ for every $A(x) \in q$ with $x \in x^M$;
- $b(z)w(z) \in A^c_{\tau,A}$ for every $A(z) \in q$ with $z \in z_M^+$;
- $(a(x), a(x)) \in P^c_{\tau,A}$ for every $P(x, x) \in q$ such that $x \in x^M$;
- $(b(z), b(z)) \in P^c_{\tau,A}$ for every $P(z, z) \in q$ such that $z \in z_M^+$;
- $T, A \models \exists y \theta(b(z), y)$ for every $z \in z_M^+$ with $w(z) = gw$ (for otherwise $b(z)w(z)$ would not belong to the domain of $C_{\tau,A}$).

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

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

are present in $A$ when $b$ and $a$ substituted for $z_M^+$ and $x^M$, respectively. The third and fourth items, again together with completeness of $A$, ensure the presence of the atoms in

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

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

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

It follows that the body of the unique rule for $G^w_M$ is satisfied when $b$ and $a$ are substituted for $z_M^+$ and $x^M$, respectively, and thus $\Pi^\varepsilon_{Q^1}, A \models G^w_M(b, a)$. 

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

$$
G^w_k(z^k, z^k+1) \leftarrow \Pi^\varepsilon_{Q^1}(z^k, z^k+1) \cap G^s_{k+1}(z_M^+, z^k+1)
$$
is entailed from $\Pi^L_{Q^1}, A$ when the individuals in $c$ are substituted for $z^k \cup z^k+1$. Recall that $\Pi^L_{Q^1}(z^k, z^k+1)$ is the conjunction of the following atoms, for $z, z' \in z^k \cup z^k+1$:

- $A(z)$, if $A(z) \in q$ and $(w \cup s)(z) = \varepsilon$;
- $P(z, z')$, if $P(z, z') \in q$ and $(w \cup s)(z) = (w \cup s)(z') = \varepsilon$;
- $z = z'$, if $P(z, z') \in q$ and either $(w \cup s)(z) \neq \varepsilon$ or $(w \cup s)(z') \neq \varepsilon$;
- $A_\varepsilon(z)$, if $(w \cup s)(z) = gw$.

In particular, we have $\Pi^\varepsilon_{Q^1}, A \models G^w_k(z^k, z^k+1)$. By the induction hypothesis, there exists a homomorphism $h^{k+1}_M : q_{k+1} \to C_{\tau,A}$ such that $h^{k+1}_M(z) = c(z)w(z)$ for every $z \in z^k \cup z^k+1$. Define a mapping $h^k$ from $\var(q_k)$ to $\Delta^c_{\tau,A}$ by setting $h^k(z) = h^{k+1}_M(z)$ for every variable $z \in \var(q_k)$. Setting $h^k(x) = a(x)$ for every $x \in z^k \cap x$, and setting $h^k(z) = b(z)w(z)$ for every $z \in z^k$. Using the same argument as was used in the base case, we can show that $h^k$ is well-defined. For atoms from $q_k$ involving only variables from $q_{k+1}$, we can use the induction hypothesis to conclude that they are satisfied under $h^k$, and for atoms only involving variables from $z^k$, we can argue as in the base case. It thus remains to handle role atoms that contain one variable from $z^k$ and one variable from $z^k+1$. Consider such an atom $P(z, z') \in q_k$, for $z \in z^k$ and $z' \in z^k+1$. If $(w(z) = s(z') = \varepsilon$, then the atom $P(z, z')$ appears in the body of the clause we are considering. It follows that $\Pi^L_{Q^1}, A \models c(z) = c(z')$, hence $(c(z), c(z')) \in P^c_{\tau,A}$. It then suffices to note that $c$ agrees with $a$ and $b$ on the variables in $z^k$. Next suppose that either $w(z) \neq \varepsilon$ or $s(z') \neq \varepsilon$. It follows that the clause body contains $z = z'$, hence $c(z) = c(z')$. As $(w, s)$ is compatible with $(z^k, z^k+1)$, one of the following must hold: either

- $(a) s(z) = w(z)$ and $T \models P(x, x)$
- $(b) \varnothing \models \theta(x, y) \to P(x, y)$ and either $s(z') = w(z)\varnothing$ or $w(z) = s(z')\varnothing$.

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

$$
(h^k(z), h^k(z')) = (b(z)w(z), c(z')s(z')) = (b(z)w(z), c(z')w(z)) \in P^c_{\tau,A}
$$

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

$$
(h^k(z), h^k(z')) = (b(z)w(z), c(z')s(z')) = (b(z)w(z), c(z')w(z)) \notin P^c_{\tau,A}
$$

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

$$
(h^k(z), h^k(z')) = (b(z)w(z), c(z')s(z')) = (b(z)s(z')\varnothing, c(z')s(z')) \in P^c_{\tau,A}
$$
For the converse direction, $(\iff)$, of the induction step, let $w$ be a type that is locally compatible with $z^k$, let $a \in \text{ind}(A)[z^k]$, $b \in \text{ind}(A)[z^k]$, and let $h^k: q_k \to C_{T,A}$ be a homomorphism satisfying
\[
h^k(x) = a(x), \quad \text{for } x \in x^k,\]
and $h^k(z) = b(z)w(z)$, for $z \in z^k_{k+1}$. (10)

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

\[
G_k^w(z^k, x^k) \leftarrow A_{\mathsf{w/s}}(z^k, z^k_{k+1}) \land G_{k+1}^w(x^k_{k+1}, x^k_{k+1})
\]

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

- $A(z)$, if $A(z) \in q$ and $(w \cup s)(z) = c$,
- $P(z,z')$, if $P(z,z') \in q$ and $(w \cup s)(z) = (w \cup s)(z') = c$,
- $z = z'$, if $P(z,z') \in q$ and either $(w \cup s)(z) \neq c$ or $(w \cup s)(z') \neq c$,
- $A_q(z)$, if $(w \cup s)(z)$ is of the form $gw$.

It follows from Equation (10) and the fact that $h^k$ is a homomorphism that each of the ground atoms obtained by taking an atom from $A_{\mathsf{w/s}}(z^k, z^k_{k+1})$ and substituting $a, b, c$ for $z^k, z^k_{k+1}$ and $z^k$, respectively, is present in $A$. By applying the induction hypothesis to the predicate $G_{k+1}$ and the homomorphism $h^{k+1}: q_{k+1} \rightarrow C_{T,A}$ obtained by restricting $h^k$ to $\text{var}(q_{k+1})$, we obtain that $\Pi_{Q}^\mathsf{INV}, A \models G^w_k(b, a)$. Since for the considered substitution, all body atoms are entailed, we can conclude that $\Pi_{Q}^\mathsf{INV}, A \models G^w_k(b, a)$. \(\blacksquare\)

It follows that answering OMQs $Q(x) = (T, q(x))$ with $T$ of finite depth $d$ and tree-shaped $q$ with $\ell$ leaves over any data instance $A$ can be done in time
\[
\text{poly}(|T|^{2d}, |q|, |A|^\ell).
\]
Indeed, $(\Pi_{Q}^\mathsf{INS}, G(x))$ can be evaluated in time polynomial in $\Pi_{Q}^\mathsf{INV}$ and $|A|^{|G(x)|}$, which are bounded by a polynomial in $|T|^{2d}$, $|q|$ and $|A|^{2\ell}$.

### A.5 Tw-rewritings

**Lemma 25.** For any OMQ $Q(x_0) = (T, q_0(x_0))$ with a tree-shaped $CQ$, any complete data instance $A$, any $q(x) \in \mathcal{Q}$ and $a \in \text{ind}(A)[z^k]$, we have $\Pi_{Q}^\mathsf{TW}, A \models G_q(a)$ if there exists a homomorphism $h: q \rightarrow C_{T,A}$ such that $h(x) = a$.

**Proof.** An inspection of the definition of the set $\mathcal{Q}$ shows that every $q(x) \in \mathcal{Q}$ is a tree-shaped query having at least one answer variable, with the possible exception of the original query $q_0(x_0)$, which may be Boolean.

Just as we did for subtrees in Section 3.2, we associate a binary relation on the queries in $\mathcal{Q}$ by setting $q'(x') \prec q(x)$ whenever $q'(x')$ was introduced when applying one of the two decomposition conditions on $p$. The proof is by induction on the subqueries in $\mathcal{Q}$, according to $\prec$. We will start by establishing the statement for all queries in $\mathcal{Q}$ other than $q_0(x_0)$, and afterwards, we will complete the proof by giving an argument for $q_0(x_0)$.

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

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

- **Case 1:** $G_q(a)$ was derived by an application of the following clause:

\[
G_q(z) \leftarrow \bigwedge_{A(z_q) \in q} A(z_q) \land \bigwedge_{P(z_q, z_q) \in q} P(z_q, z_q) \land \bigwedge_{i \leq n} G_q(x_i),
\]

where $q_1(x_1), \ldots, q_n(x_n)$ are the subqueries induced by the neighbours of $z_q$ in the Gaifman graph $G$ of $q$. Then there exists a substitution $c$ for the variables in the body of this rule that coincides with $a$ on $z$ and is such that the ground atoms obtained by applying $c$ to the variables in the body are all entailed from $\Pi_{Q}^\mathsf{TW}, A$. In particular, $\Pi_{Q}^\mathsf{TW}, A \models \text{var}(q) \models \{x\}$...
$G_q(c(x_i))$ for every $1 \leq i \leq n$. We can apply the induction hypothesis to the $q_i(x_i)$ to obtain homomorphisms $h_i: q_i \rightarrow C_{T,A}$ such that $h_i(z_i) = c(x_i)$. Let $h$ be the mapping from $\text{var}(q)$ to $\Delta^{C_{T,A}}$ defined by taking $h(z) = h_i(z)$, for $z \in \text{var}(q_i)$. Note that $h$ is well-defined since $\text{var}(q) = \bigcup_{i=1}^{n} \text{var}(q_i)$, and the $q_i$ have no variable in common other than $z_q$, which is sent to $c(z_q)$ by every $h_i$. To see why $h$ is a homomorphism from $q$ to $C_{T,A}$, observe that

$$q = \bigcup_{i=1}^{n} q_i \cup \{ A(z_q) \in q \} \cup \{ P(z_q, z_q) \in q \}.$$  

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

- **Case 2:** $G_q(a)$ was derived by an application of the following clause, for a tree witness $t$ for $(T, q(x))$ generated by $\theta$ with $t \neq \emptyset$ and $z_0 \in t$:

$$G_q(x) \leftarrow A_0(z_0) \land \bigwedge_{z \in t \setminus \{z_0\}} (z = z_0) \land \bigwedge_{1 \leq i \leq k} G_q^i(x^i),$$

where $q_1^i, \ldots, q_k^i$ are the connected components of $q$ without $q_i$ and $z_0$ is some variable in $t$. There must exist a substitution $c$ for the variables in the body of this rule that coincides with $a$ on $x$ and is such that the ground atoms obtained by applying $c$ to the variables in the body are all entailed from $\Pi^T_{\mathcal{A}} \cdot \mathcal{A}$. In particular, for every $1 \leq i \leq k$, we have $\Pi^T_{\mathcal{A}} \cdot \mathcal{A} \models G_q^i(c(x^i))$. We can apply the induction hypothesis to the $q_i^i(x^i)$ to find homomorphisms $h_1, \ldots, h_k$ of $q_1^i, \ldots, q_k^i$ into $C_{T,A}$ such that $h_i(x^i) = c(x^i)$. Since $t$ is a tree witness for $(T, q(x))$ generated by $\theta$, there exists a homomorphism $h_0$ of $q_i$ into $C_{T,A}$ with $t_0 = h^{-1}(a)$ and such that $h_0(z)$ begins by $a$ for every $z \in t_0$. Now take $z_0 \in t_0$ such that $A_0(z_0)$ is the atom in the clause body (recall that $t_0 \neq \emptyset$), and so $\Pi^T_{\mathcal{A}} \cdot \mathcal{A} \models A_0(c(z_0))$, which means that $A_0(c(z_0))$ must appear in $A$. It follows that for every element in $C_{T, \{A_0(a)\}}$ of the form $a gw$, there exists a corresponding element $c(z_0) gw$ in $\Delta^{C_{T,A}}$. We now define a mapping $h$ from $\text{var}(q)$ to $\Delta^{C_{T,A}}$ as follows:

$$h(z) = \begin{cases} h_i(z), & \text{for every } z \in \text{var}(q_i), \\ c(z_0) gw, & \text{if } z \in t_0 \text{ and } h_0(z) = a gw, \\ c(z_0) & \text{if } z \in t_0. \end{cases}$$

Every variable in $\text{var}(q)$ occurs in $t_0 \cup t_i$ or in exactly one of the $q_i^i$, and so is assigned a unique value by $h$. Note that although $t_0 \cap \text{var}(q_i^i)$ is not necessarily empty, due to the equality atoms, we have $h(z) = h(z')$ for all $z, z' \in t$, and so the function is well-defined. We claim that $h$ is a homomorphism from $q$ into $C_{T,A}$. Clearly, the atoms occurring in some $q_i^i$ are preserved under $h$. Now consider some unary atom $A(z)$ with $z \in t$. Then $h(z) = c(z_0) gw$, where $h_0(z) = a gw$. Since $h_0$ is a homomorphism, we know that $w$ ends with a role $\sigma$ such that $T \models \exists y \sigma(y, x) \rightarrow A(x)$. It follows that $h(z)$ also ends with $\sigma$, and thus $h(z) \in A^{C_{T,A}}$. Next, consider a binary atom $P(z, z')$, where at least one of $z$ and $z'$ belongs to $t$. As $h_i$ is a homomorphism, either

$$- T \models \sigma(x, y) \rightarrow P(x, y), \text{ for some } \sigma,$$

or $T \models P(x, y)$ and $h_i(z') = h_i(z)$. We also know that $c(z) = c(z_0)$ for all $z \in t$, hence $h(z) = h_0(z)$ for all $z \in t$. It follows that in the former case we have $h_i(z') = h_i(z) \sigma$ or $h_i(z) = h_i(z') \sigma$ with $T \models \sigma(x, y) \rightarrow P(x, y)$. In the latter case, we have $h(z') = h(z)$ with $T \models P(x, x)$. Thus, $P(z, z')$ is preserved under $h$. Finally, since $c$ coincides with $a$ on $x$, we have $h(x) = a$.

For the converse direction, $(\subseteq)$, of the induction step, suppose that $h$ is a homomorphism of $q$ into $C_{T,A}$ such that $h(x) = a$. There are two cases to consider, depending on where $h$ maps the ‘splitting’ variable $z_q$.

- **Case 1:** $h(z_q) \in \text{ind}(\mathcal{A})$. Let $q_1(x_1), \ldots, q_n(x_n)$ be the subqueries of $q(x)$ induced by the neighbours of $z_q \in \mathcal{G}$. Recall that $t_i$ consists of $z_q$ and the variables in $\text{var}(q_i) \cap x$. By restricting $h$ to $\text{var}(q_i)$, we obtain, for each $1 \leq i \leq n$, a homomorphism of $q_i(x_i)$ into $C_{T,A}$ that maps $z_q$ to $h(z_q)$ and $\text{var}(q_i) \cap x$ to $a(\text{var}(q_i) \cap x)$. Consider $\alpha^*(x) = a(x)$ for every $x \in \text{var}(q_i) \cap x$ and $\alpha^*(z_q) = h(z_q)$. By the induction hypothesis, for every $1 \leq i \leq n$, we have $\Pi^T_{\mathcal{A}} \cdot \mathcal{A} \models G_q^i(\alpha^*(x))$. Next, since $h$ is a homomorphism, we must have $h(z_q) \in A^{C_{T,A}}$ whenever $A(z_q) \in q$ and $(h(z_q), h(z_q)) \in P^{C_{T,A}}$ whenever $P(z_q, z_q) \in q$. Since $\mathcal{A}$ is a complete data instance, $A(h(z_q)) \in \mathcal{A}$ for every $A(z_q) \in q$ and $P(h(z_q), h(z_q)) \in \mathcal{A}$ for every $P(z_q, z_q) \in q$. We have thus shown that, under the substitution $\alpha^*$, every atom in the body of the clause

$$G_q(z) \leftarrow \bigwedge_{A(z_q) \in q} A(z_q) \land \bigwedge_{1 \leq i \leq n} P(z_q, z_q) \land \bigwedge_{A(z_q) \in q} \bigwedge_{1 \leq i \leq n} G_q^i(x_i),$$

is entailed from $\Pi^T_{\mathcal{A}} \cdot \mathcal{A}$. It follows that we must also have $\Pi^T_{\mathcal{A}} \cdot \mathcal{A} \models G_q^i(\alpha^*(x_i))$.

- **Case 2:** $h(z_q) \notin \text{ind}(\mathcal{A})$. Then $h(z_q)$ is of the form $b gw$, for some $\theta$. Let $v$ be the smallest subset of $\text{var}(q)$ that contains $z_q$ and satisfies the following closure property:

- if $z \in V$, $h(z) \notin \text{ind}(\mathcal{A})$ and $q$ contains an atom with $z$ and $z'$, then $z' \in V$. 


Let \( V' \) consist of all variables \( z \) in \( V \) such that \( h(z) \notin \text{ind}(A) \). We observe that \( h(z) \) begins by \( bq \) for every \( z \in V' \) and \( h(z) = b \) for every \( z \in V \setminus V' \). Define \( q_V \) as the CQ comprising all atoms in \( q \) whose variables are in \( V \) and which contain at least one variable from \( V' \); the answer variables of \( q_V \) are \( V \setminus V' \). By replacing the initial \( b \) by \( a \) in the mapping \( h \), we obtain a homomorphism \( h_V \) of \( q_V \) into \( C_{\mathcal{T}, \{A_p(a)\}} \) with \( V \setminus V' = h_V^{-1}(a) \). It follows that \( t = (t_x, t_y) \) with \( t_x = V \setminus V' \) and \( t_y = V' \) is a tree witness for \((\mathcal{T}, q(x))\) generated by \( q \) (and \( q \neq q_V \)). Moreover, \( t_y \neq \emptyset \) because \( q \) has at least one answer variable. This means that the program \( \Pi_{q}^{\mathcal{T}} \) contains the following clause

\[
q(x) \leftarrow A_q(z_0) \land \bigwedge_{i \in t_x \{z_0\}} (z = z_0) \land \bigwedge_{1 \leq i \leq k} G_q^i(x_i^i),
\]

where \( q_1^i, \ldots, q_k^i \) are the connected components of \( q \) without \( q \) and \( z_0 \). Recall that the query \( x^i = \var(q_i^j) \cap (x \cup t_x) \). Let \( a^* \) be the substitution for \( x \cup t_x \) such that \( a^*(x) = a(x) \) for \( x \in x \) and \( a^*(z) = h(z) \) for \( z \in t_x \). Then, for every \( 1 \leq i \leq k \), there exists a homomorphism \( h_i \) from \( q_i^i \) to \( C_{\mathcal{T}, A} \) such that \( h_i(z) = a^*(x) \) for every \( x \in x_i^i \). By the induction hypothesis, we obtain \( \Pi_{q_i}^{\mathcal{T}, A} = G_{q_i^j}(a^*(x_i^j)) \). Next, since \( h(z) = b \) for every \( z \in t_y \), we have \( a^*(z) = a^*(z') \) for every \( z, z' \in t_y \). Moreover, \( A \) contains all variables in the body of the clause under consideration are entailed by \( \Pi_{\mathcal{T}, A}^{\mathcal{T}, A} \). Therefore, we must also have \( \Pi_{q_i}^{\mathcal{T}, A} = G_{q_i^j}(a) \).

We have thus shown the lemma for all queries \( Q \) other than \( q_0(x_0) \) and \( q_0(x_0) \).

For the first direction, \( (\Rightarrow) \), suppose \( \Pi_{q_0}^{\mathcal{T}, A} = G_{q_0}(a) \). There are four cases, depending on which type of clause was used to derive \( G_{q_0}(a) \). We skip the first three cases, which are identical to those considered in the base case and induction step, and focus instead on the case in which \( G_{q_0}(a) \) was derived using a clause of the form \( G_{q_0} \leftarrow A(x) \) with \( A \) a unary predicate such that \( \mathcal{T}, \{A(\alpha)\} \models q_0(a) \). In this case, we must exist some \( b \in \text{ind}(A) \) such that \( \mathcal{T}, A \models A(b) \). By completeness of \( A \), we obtain \( A(b) \models A(b) \). Since \( \mathcal{T}, \{A(\alpha)\} \models q_0, \) we get \( \mathcal{T}, A \models q_0 \), which implies the existence of a homomorphism from \( q_0 \) into \( C_{\mathcal{T}, A} \).

For the converse direction, \( (\Leftarrow) \), suppose that there is a homomorphism \( h : q_0 \rightarrow C_{\mathcal{T}, A} \) such that \( h(x_0) = a \). We focus on the case in which \( q_0 \) is Boolean \( (x_0 = 0) \) and none of the variables in \( q_0 \) is mapped to an individual constant (the other cases can be handled exactly as in the induction basis and induction step). In this case, there must exist an individual constant \( b \) and some \( g \) such that \( h(z) \) begins by \( bq \) for every \( z \in \text{var}(q_0) \). It follows that \( \mathcal{T}, \{A_p(a)\} \models q_0, \) since the mapping \( h' \) defined by setting \( h'(z) = aqw \) whenever \( h(z) = bq \) is a homomorphism from \( q_0 \) to \( C_{\mathcal{T}, \{A_p(a)\}} \). It follows that \( \Pi_{q}^{\mathcal{T}} \) contains the clause \( G_{q_0} \leftarrow A_q(x) \). Since \( bq \) occurs in \( \Delta_{\mathcal{T}, A} \), we have \( \mathcal{T}, A \models A_{q}(b) \). By completeness of \( A \), \( A_{q}(b) \models A \), and so by applying the clause \( G_{q_0} \leftarrow A_q(x) \), we obtain \( \Pi_{q}^{\mathcal{T}}, A \models G_{q_0} \).

A.6 Rewritings Zoo

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

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

```
    x_0 R    S    x_2 R    x_3 R    S    x_5 R    x_6 R    x_7
```

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

\[
P(x, y) \rightarrow S(x, y), \quad P(x, y) \rightarrow R(y, x),
\]

\[
A_p(x) \leftrightarrow \exists y P(x, y), \quad A_{p-}(x) \leftrightarrow \exists y P(y, x),
\]

\[
A_R(x) \leftrightarrow \exists y R(x, y), \quad A_{R-}(x) \leftrightarrow \exists y R(y, x),
\]

\[
A_S(x) \leftrightarrow \exists y S(x, y), \quad A_{S-}(x) \leftrightarrow \exists y S(y, x).
\]

A.6.1 UCQ rewriting

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

\[
G(x_0, x_7) \leftarrow [R(x_0, x_1) \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 [A_{p-}(x_0) \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 A_{p-}(x_1) \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 A_p(x_3)] \land \[R(x_3, x_6) \land A_p(x_6)] \land R(x_6, x_7).
\]

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

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

\[
\begin{align*}
&\left([ (R(x_0, x_1) \land S(x_1, x_2) \land R(x_2, x_3)) \right. \\
&\left. \lor (A_p-(x_0) \land R(x_0, x_3)) \lor (R(x_0, x_3) \land A_p(x_3)) \right]
\end{align*}
\]

\[
\begin{align*}
&\left([ (R(x_3, x_4) \land S(x_4, x_5) \land R(x_5, x_6)) \right. \\
&\left. \lor (A_p-(x_3) \land R(x_5, x_6)) \lor (R(x_3, x_6) \land A_p(x_6)) \right]
\end{align*}
\]

\[
R(x_6, x_7).
\]

(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 \(RAp\).

A.6.2 Log-rewriting

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

\[
\begin{align*}
G^7_7(x_0, x_7) &\leftarrow G^{x_0\rightarrow x}(x_3, x_0) \land R(x_3, x_4) \land G^{x_4\rightarrow x}(x_4, x_7), \\
G^7_7(x_0, x_7) &\leftarrow G^{x_3\rightarrow x}(x_3, x_0) \land A_p-(x_4) \land (x_3 = x_4) \land G^{x_4\rightarrow x}(x_4, x_7).
\end{align*}
\]

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

\[
\begin{align*}
G^{x_0\rightarrow x}(x_3, x_0) &\leftarrow (x_0 = x_1) \land A_p-(x_1) \land (x_1 = x_2) \land R(x_2, x_3), \\
G^{x_1\rightarrow x}(x_3, x_0) &\leftarrow R(x_0, x_1) \land (x_1 = x_2) \land A_p(x_2) \land (x_2 = x_3), \\
G^{x_3\rightarrow x}(x_3, x_0) &\leftarrow R(x_0, x_1) \land S(x_1, x_2) \land R(x_2, x_3), \\
G^{x_4\rightarrow x}(x_4, x_7) &\leftarrow (x_4 = x_5) \land A_p(x_5) \land (x_5 = x_6) \land R(x_6, x_7), \\
G^{x_5\rightarrow x}(x_4, x_7) &\leftarrow S(x_4, x_5) \land R(x_5, x_6) \land R(x_6, x_7), \\
G^{x_4\rightarrow x}(x, x_7) &\leftarrow A_p-(x_4) \land (x_4 = x_5) \land R(x_5, x_6) \land R(x_6, x_7).
\end{align*}
\]

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 \(G^T_7\) consists of 8 rules. Note, however, that the rewriting illustrated above is a slight simplification of the definition given in Section 3.2: here, for the leaves of the tree decomposition, we directly use the atoms \(At^e\) instead of including a rule \(G^D_p(\delta D, x_p) \leftarrow At^e\) in the rewriting. This simplification clearly does not affect the width of the NDL query or the choice of weight function.

A.6.3 Lin-rewriting

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

\[
G(x_0, x_7) \leftarrow G_0^{x_0\rightarrow x}(x_0, x_7).
\]

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

\[
\begin{align*}
&G_0^{x_0\rightarrow x}(x_0, x_7) \leftarrow R(x_0, x_1) \land P_1^{x_1\rightarrow x}(x_1, x_7), \\
&G_0^{x_0\rightarrow x}(x_0, x_7) \leftarrow (x_0 = x_1) \land A_p-(x_1) \land G_1^{x_1\rightarrow x}(x_1, x_7), \\
&G_1^{x_1\rightarrow x}(x_1, x_7) \leftarrow S(x_1, x_2) \land G_2^{x_2\rightarrow x}(x_2, x_7), \\
&G_1^{x_1\rightarrow x}(x_1, x_7) \leftarrow (x_1 = x_2) \land A_p(x_2) \land G_2^{x_2\rightarrow x}(x_2, x_7), \\
&G_1^{x_1\rightarrow x}(x_1, x_7) \leftarrow A_r-(x_1) \land (x_1 = x_2) \land G_2^{x_2\rightarrow x}(x_2, x_7), \\
&G_2^{x_2\rightarrow x}(x_2, x_7) \leftarrow R(x_2, x_3) \land G_3^{x_3\rightarrow x}(x_3, x_7), \\
&G_2^{x_2\rightarrow x}(x_2, x_7) \leftarrow A_p(x_2) \land (x_2 = x_3) \land G_3^{x_3\rightarrow x}(x_3, x_7).
\end{align*}
\]

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

\[
\begin{align*}
&G_3^{x_3\rightarrow x}(x_3, x_7) \leftarrow R(x_3, x_4) \land P_4^{x_4\rightarrow x}(x_4, x_7), \\
&G_3^{x_3\rightarrow x}(x_3, x_7) \leftarrow (x_3 = x_4) \land A_p-(x_4) \land G_4^{x_4\rightarrow x}(x_4, x_7), \\
&G_4^{x_4\rightarrow x}(x_4, x_7) \leftarrow S(x_4, x_5) \land G_5^{x_5\rightarrow x}(x_5, x_7), \\
&G_4^{x_4\rightarrow x}(x_4, x_7) \leftarrow (x_4 = x_5) \land A_p(x_5) \land G_5^{x_5\rightarrow x}(x_5, x_7), \\
&G_4^{x_4\rightarrow x}(x, x_7) \leftarrow A_p-(x_4) \land (x_4 = x_5) \land G_5^{x_5\rightarrow x}(x_5, x_7), \\
&G_5^{x_5\rightarrow x}(x_5, x_7) \leftarrow R(x_5, x_6) \land G_6^{x_6\rightarrow x}(x_6, x_7), \\
&G_5^{x_5\rightarrow x}(x_5, x_7) \leftarrow A_p(x_2) \land (x_5 = x_6) \land G_6^{x_6\rightarrow x}(x_6, x_7).
\end{align*}
\]

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

\[
G_6^{x_6\rightarrow x}(x_6, x_7) \leftarrow R(x_6, x_7).
\]

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

A.6.4 Tw-rewriting

We begin by splitting the query roughly in the middle, that is, we choose \(x_2\) and consider two subqueries:

\[
\begin{align*}
q_03(x_0, x_3) &\equiv \exists x_1 x_2 \left( R(x_0, x_1) \land S(x_1, x_2) \land R(x_2, x_3) \right) \\
&\text{and} \\
q_37(x_3, x_7) &\equiv \exists x_4 x_5 x_6 \left( R(x_3, x_4) \land S(x_4, x_5) \land R(x_5, x_6) \land R(x_6, x_7) \right).
\end{align*}
\]

Since there is no tree witness \(t\) for \((T, q(x_0, x_7))\) that contains \(x_3\) in \(t\), we have only one top-level rule:

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

Next, we focus on \(q_{03}\) and choose \(x_1\) as the splitting variable. In this case, there is a tree witness \(t^1\) with \(t^1 = \{x_1\}\) and \(t^1 = \{x_0, x_2\}\), and we obtain two rules for \(G_{03}\):

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

\[
G_{03}(x_0, x_3) \leftarrow R(x_0, x_1) \land G_{13}(x_1, x_3),
\]
The subquery \( q_{13}(x_1, x_3) \) contains two atoms and is split at \( x_3 \). Since there is a tree witness \( t^2 \) for \( (T, q_{13}(x_1, x_3)) \) with \( t^2 = \{ x_2 \} \) and \( t^3 = \{ x_1, x_3 \} \), we obtain two rules:

\[
G_{13}(x_1, x_3) \leftarrow S(x_1, x_2) \lor R(x_2, x_3),
\]

\[
G_{13}(x_1, x_3) \leftarrow A_P(x_1) \lor (x_1 = x_3).
\]

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

\[
G_{37}(x_3, x_7) \leftarrow G_{35}(x_3, x_5) \lor G_{57}(x_5, x_7),
\]

\[
G_{37}(x_5, x_7) \leftarrow R(x_3, x_4) \land A_P(x_4) \land (x_4 = x_6) \lor R(x_6, x_7),
\]

\[
G_{35}(x_3, x_5) \leftarrow R(x_3, x_5) \lor S(x_3, x_7),
\]

\[
G_{57}(x_3, x_5) \leftarrow A_P(x_3) \lor (x_3 = x_5),
\]

\[
G_{57}(x_3, x_5) \leftarrow R(x_3, x_4) \lor R(x_4, x_7).
\]

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

## B. PROOFS FOR SECTION 4

### B.1 Theorem 15

**Theorem 15.** \( p\text{Depth}-\text{TreeOMQ} \) is \( W[2] \)-hard.

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

\[
V^l_i - 1(x) \rightarrow \exists z V^l_i(x, z) \quad \text{and} \quad V^l_i(x, z) \rightarrow P(z, x) \land V^l_i(z), \quad \text{for } 0 \leq i < i' \leq n,
\]

\[
V^l_i(x) \rightarrow E^l_i(z), \quad \text{for } v_i \in e_j, \quad e_j \in E,
\]

\[
E^l_i(x) \rightarrow \exists z \eta^l_i(x, z) \quad \text{and} \quad \eta^l_i(x, z) \rightarrow P(x, z) \land E^l_i - 1(z), \quad \text{for } 1 \leq j \leq m.
\]

We show that \( \mathcal{C} \models q^k_H \) iff \( H \) has a hitting set of size \( k \).

\((\Rightarrow)\) Suppose \( h : q^k_H \rightarrow \mathcal{C} \) is a homomorphism. Note that \( \mathcal{C} \) satisfies the following properties: (i) \( w \in E^l_j \) iff \( w = av_1^e_i 1_{i_2} \ldots v_2^e_i 1_{i_3} \ldots \eta_1^e_i \ldots \eta_2^e_i \) such that \( v_1 \in e_j \) and (ii) all points in \( \Delta^C \) have at most one \( P \)-predecessor. By starting with some \( E^l_j \) atom and applying first \( i \) and then iterating \( ii \), we conclude that \( h(y) = av_1^e_i \ldots v_k^e_i \) for some \( 1 \leq i_1 < i_2 < \ldots < i_k \leq n \). We claim that \( \{ v_{i_1}, v_{i_2}, \ldots, v_{i_k} \} \) is a hitting set in \( H \). Indeed, for every branch \( j \) of \( q^k_H \), there is \( 1 \leq s \leq k \) such that this branch is mapped on \( e_s \) in the following way:

\[
h(z_j^s) = av_1^e_i v_2^e_i \ldots v_t^e_i, \quad s \leq l \leq k - 1,
\]

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

\((\Leftarrow)\) Suppose \( \{ v_{i_1}, v_{i_2}, \ldots, v_{i_k} \} \) is a hitting set in \( H \). We construct a homomorphism \( h \) from \( q^k_H \) to \( C \). First, we set \( h(y) = av_1^e_i \ldots v_k^e_i \). Then, for each \( 1 \leq j \leq m \), we find such \( v_{i_j} \in e_j \) and define \( h \) as follows:

\[
h(z_j^s) = av_1^e_i v_2^e_i \ldots v_{i_j}^e_i, \quad s \leq l \leq k - 1,
\]

\[
h(z_j^s) = av_1^e_i v_2^e_i \ldots v_{i_j}^e_i \eta_1^e_i \eta_2^e_i \ldots \eta_j^e_i, \quad 0 \leq l < s.
\]

It should be clear that \( h \) is indeed a homomorphism. \( \square \)

### B.2 Theorem 16

**Theorem 16.** \( p\text{Leaves}-\text{TreeOMQ} \) is \( W[1] \)-hard.

**Proof.** We prove that \( T_G, \{ A(a) \} \models q_G \) iff \( G \) has a clique containing one vertex from each set \( V_i \).

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

\[
B(x) \rightarrow \exists y P(x, y),
\]

\[
P(x, y) \rightarrow U(x, y),
\]

\[
P(x, y) \rightarrow U(y, x).
\]

To simplify notation, we will abbreviate \( C_{T_G, \{ A(a) \}} \) by \( \mathcal{C} \) and for every \( 1 \leq j \leq M \), we let \( w(v_j) = L_{j} L_{j+1} \ldots L_{j+2M} \).

Observe that for every \( v_{j_1} \in V_1, v_{j_2} \in V_2, \ldots, v_{j_p} \in V_p \), the element \( aw(v_{j_1})w(v_{j_2}) \ldots w(v_{j_p}) \) belongs to \( \Delta^C \). Further, observe that if \( aw \in \Delta^C \) with \( |w| = 2M \cdot p \), then there exist \( v_{j_1} \in V_1, v_{j_2} \in V_2, \ldots, v_{j_p} \in V_p \) such that \( w = w(v_{j_1})w(v_{j_2}) \ldots w(v_{j_p}) \).

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

\[
aw(v_{j_1})w(v_{j_2}) \ldots w(v_{j_p})
\]

where \( v_{j_1} \in V_1, v_{j_2} \in V_2, \ldots, v_{j_p} \in V_p \). We claim that \( \{ v_{j_1}, \ldots, v_{j_p} \} \) is a clique in \( G \). To see why, consider the \( j \)-th branch of \( q_G \); compactly represented as follows:

\[
(U^{2M-2} \cdot (YY \cdot U^{2M-2})^j \cdot SS)(y, z_i)
\]

By examining the axioms, we see that starting from the first occurrence of \( YY \), every \( U \) and \( Y \) atom takes us one step closer to a (prior to the first \( YY \), we may go back and forth on the extra \( P \)-edge leaving from \( h(y) \)). It follows that \( SS \) must be mapped within the \( p \)-th block of the selected branch, and since \( S \) is present only at positions \( 2j_{p-1} \) and \( 2j_{p-1} + 1 \) of the block, we must have

\[
h(z_i) = aw(v_{j_1}) \ldots w(v_{j_{p-1}})L_{j_{p-1}} \ldots L_{2j_{p-1}}
\]

As the
distance between consecutive occurrences of YY (and between the final YY and the SS) is \(2M - 2\), it follows that all YY blocks occur at positions \(2j_{p-i}\) and \(2j_{p-i} + 1\) of blocks \(p - i + 1, \ldots, p\), which implies that \(v_{j_{p-i}+1}, \ldots, v_{j_p}\) are neighbours of \(v_j\), in \(G\). Since \(q_G\) contains branches for every \(1 \leq i < p\), the selected vertices \(v_1, \ldots, v_j\) are all neighbours in \(G\), and \(G\) contains a clique with the required properties.

(\(\Leftarrow\)) Suppose that \(v_j \in V_1, \ldots, v_p \in V_p\) form a clique. We construct a homomorphism \(h\) of \(q_G\) into \(C\).

First, set \(h(y) = aw\) where \(w = w(v_{j_1})w(v_{j_2})\ldots w(v_{j_p})\) and observe that the atom \(B(y)\) is satisfied by this assignment. We will use \(w[\ell, \ell']\) to denote the subword of \(w\) beginning with the \(\ell\)th symbol of \(w\) and ending with the \(\ell'\)th symbol (note that \(w = [2M \cdot p, \text{so } w = w[1, 2M \cdot p])\)). Next, consider the \(i\)th branch of the query, which connects \(y\) to \(z_i\), and let \(y_0, y_1, \ldots, y_{2M(i+1)}\) be the variables lying between \(y\) and \(z_i\) with \(y_0 = y\) and \(z_i = y_{2M(i+1)}\). For \(0 \leq k \leq 2j_{p-i} - 1\), we set \(h(y_k) = h(y)\) if \(k\) is even, and \(h(y_k) = h(y)P\) otherwise. Observe that because \(P\) is included in both \(U\) and \(U'\), we satisfy all binary atoms between variables \(y_0, \ldots, y_{2j_{p-i}-1}\). For \(2j_{p-i} < k \leq 2M(i+1)\), we set

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

Note that, in particular, this yields

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

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

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

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

\(\square\)

### C. PROOFS FOR SECTION 5

#### C.1 Theorem 17

**Theorem 17.** There is an ontology \(T_f\) such that answering OMQs of the form \((T_f, q)\) with Boolean tree-shaped CQs \(q\) is NP-hard for query complexity.

**Proof.** We assume that \(T_f\) consists of the following axioms:

\[
A(x) \rightarrow \exists y v_+(x, y)
\]

\[
v_+(x, y) \rightarrow P_+(y, x) \land P_0(y, x) \land B_-(y) \land A(y),
\]

\[
B_-(x) \rightarrow \exists y \eta_-(x, y)
\]

\[
\eta_-(x, y) \rightarrow P_-(x, y) \land B_0(y),
\]

\[
A(x) \rightarrow \exists y v_-(x, y)
\]

\[
v_-(x, y) \rightarrow P_-(y, x) \land P_0(y, x) \land B_+(y) \land A(y),
\]

\[
B_+(x) \rightarrow \exists y \eta_+(x, y)
\]

\[
\eta_+(x, y) \rightarrow P_+(x, y) \land B_0(y),
\]

\[
B_0(x) \rightarrow \exists y \eta_0(x, y)
\]

\[
\eta_0(x, y) \rightarrow P_+(x, y) \land P_-(x, y) \land P_0(x, y) \land B_0(y).
\]

Let \(C\) be the canonical model of \((T_f, \{A(a)\})\). We prove that \(C \models q_c\) if \(\varphi\) is satisfiable.

(\(\Rightarrow\)) Suppose \(h\) is a homomorphism from \(q_c\) to \(C\) and \(h(z_j^1) = h(y) = aq_1 \ldots aq_{s-l}\), for some roles \(q_i\). Since \(A(y) \in q_c\), it follows that \(q_1 \in \{v_+, v_\}\). Moreover, because of the structure of \(C\), without any loss of generality we may assume that \(n = k\). Define a valuation \(\nu: \{p_1, \ldots, p_k\} \rightarrow \{t, f\}\) by taking \(\nu(p_1) = t\) if \(q_1 = v_+\), \(\nu(p_1) = f\) if \(q_1 = v_\). We claim that \(\nu\) makes \(\varphi\) true. To verify that the clause \(\chi_j\) is satisfied, consider a number \(1 \leq s \leq k\), that such the \(j\)th branch of the query is mapped on \(C\) in the following way:

\[
h(z_j^1) = aq_1 \ldots aq_l,
\]

\[
h(z_j^1) = aq_1 \ldots aq_l \nu_1 \ldots \nu_{s-l},
\]

\(s \leq l \leq k, 0 \leq l < s, \ldots, p_s\) for some roles \(\gamma_1 \ldots \gamma_{s-l}\).\(\gamma_1 \in \{\nu_-, \nu_+\}\) and \(\gamma_i = \nu_\) for \(2 \leq i \leq s - l\). Such \(s\) and the roles \(\gamma_i\) exist, because the \(P\)-atoms in \(C\) are directed towards the root if they cover \(\nu\)-atoms, and away from the root if they cover \(\eta\)-atoms \((s \geq 1\) since \(B_0(z_j^1) \in q_c\)). Clearly, \(T_f \models \gamma_1(x, y) \rightarrow P_+(x, y)\) iff \(\rho_\nu = v_+\) if \(\nu(p_1) = t\) and \(\rho_\nu = v_-\) if \(\nu(p_1) = f\). It follows that either \(P'(z_j^1, z_j^{s-l-1}) \in q_c\) and \(\nu(p_1) = t\), or \(P'(z_j^1, z_j^{s-l-1}) \in q_c\) and \(\nu(p_1) = f\). In either case, \(\chi_j\) contains a literal with \(p_s\) satisfied by \(\nu\).

(\(\Leftarrow\)) Suppose a valuation \(\nu: \{p_1, \ldots, p_k\} \rightarrow \{t, f\}\) satisfies \(\varphi\). Consider the sequence of roles \(q_1 \ldots q_k\), such that for \(1 \leq l \leq k\) we have \(q_l = v_\), if \(\nu(p_1) = f\), and \(q_l = v_+\), if \(\nu(p_1) = t\). We claim that there exists a homomorphism \(h\) from \(q_c\) to \(C\). First, let \(h(y) = aq_1 \ldots aq_k\).

To map the \(j\)th branch of the query, consider the maximal \(1 \leq s \leq k\), such that a \(p_s\)-literal (positive or negative) makes \(\chi_j\) true. Set

\[
h(z_j^1) = aq_1 \ldots aq_l,
\]

\[
h(z_j^1) = aq_1 \ldots aq_s \gamma_1 \ldots \gamma_{s-l},
\]

\(s \leq l \leq k - 1, 0 \leq l < s, \ldots, p_s\) if \(p_s\) occurs positively, \(\gamma_1 = \eta_+\) if \(p_s\) occurs negatively and \(\gamma_i = \eta_0\) for \(i \geq 2\) that \(z_j^1\), for \(s \leq l \leq k - 1\), are mapped correctly follows from the maximality of \(s\). That \(z_j^1\), for \(s \leq l \leq k - 1\), are mapped correctly follows from the fact that \(p_s\) occurs in \(\chi_j\) positively if \(P_+(z_j^1, z_j^{s-l}) \in q_c\) if \(\nu(p_s) = t\) if \(\nu(p_s) = v_-\) if \(\nu(p_s) = v_+\) (similarly for negative \(p_s\)). Finally, \(z_j^1\) is mapped correctly for \(0 \leq l < s - 1\) since the sequence of roles \(\gamma_2 \ldots \gamma_{s-l}\) can embed any \(P_+, P_-,\) or \(P_0\) roles, and \(B_0\) concept. Thus, \(h\) is a homomorphism from \(q_c\) to \(C\).

\(\square\)

#### C.2 Theorem 20

We need several intermediate results and definitions before we present the proof in the end of the section. Suppose \(\varphi\) is a propositional formula in CNF having
Figure 3: Example of $\bar{q}_\varphi(x)$ and $\mathcal{C}_{T_1,A_m^\alpha}$ for $\varphi = \chi_1 \land \ldots \land \chi_4$ with $\chi_1 = (p_1 \lor \neg p_3 \lor \neg p_4)$, $\chi_2 = (\neg p_3 \land p_4)$, $\chi_3 = p_1$, $\chi_4 = (\neg p_3 \lor \neg p_4)$ and $\alpha = (0,1,1,0)$

$k$ variables $p_1,\ldots,p_k$ and $m$ clauses $\chi_1,\ldots,\chi_m$. We assume that $m = 2^l$. We associate with each query $\Phi(a)$, where $1 \leq j \leq m$, $1 \leq l \leq k$, and $z_j^l = y^k$: $\bar{q}_\varphi(x) = P_0(y^1, x), \ldots, P_0(y^k, y^{k-1})$, $P_+ (z_j^l, z_{j+1}^l)$ if $\chi_j$ contains $p_l$, $P_-(z_j^l, z_{j+1}^l)$ if $\chi_j$ contains $\neg p_l$, $P_0(z_j^l, z_{j+1}^l)$ if $\chi_j$ contains no occurrence of $p_l$.

Then, for $0 \leq l \leq \ell - 1$, $P_-(z_j^l, z_{j+1}^l)$, if the $l$th bit of $(j-1)_2$ is 0, $P_+(z_j^l, z_{j+1}^l)$, if the $l$th bit of $(j-1)_2$ is 1, $B_0(z_j^\ell)$.

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

Denote by $f_\varphi: \{0,1\}^m \rightarrow \{0,1\}$ the monotone function such that $f_\varphi(\alpha) = 1$ if the CNF $\varphi^\alpha$, which is obtained from $\varphi$ by removing all conjuncts $\chi_i$ with $\alpha_i = 1$, is satisfiable. It is readily checked that we have

**Lemma 26.** For any $\alpha \in \{0,1\}^m$, $T_1,A_m^\alpha \models \bar{q}_\varphi(a)$ iff $f_\varphi(\alpha) = 1$.

Let $QL$ be any query language such that, for any $QL$-query $\Phi(x)$ and any $A_m^\alpha$, the answer to $\Phi(a)$ over $A_m^\alpha$ can be computed in time $poly(|\Phi(a)|)$.

**Theorem 27.** The OMQ $(T_1,\bar{q}_\varphi(x))$ does not have a polynomial-size rewriting in $QL$ unless $NP \subseteq P/poly$.

**Proof.** Take any sequence of CNFs $\varphi_n$ of polynomial size in $n$ such that $f_{\varphi_n}$ is $NP$-hard [23, Sec. 3]. Suppose there is a $QL$-rewriting $\Phi_n$ of $(T_1,\bar{q}_\varphi(x))$ of polynomial size. By adapting the proof of $P \subseteq P/poly$ [3, Theorem 6.6] to the algorithm that checks $A_m^\alpha \models \Phi_n(a)$, we obtain a sequence of polynomial-size circuits computing $f_{\varphi_n}$, from which $NP \subseteq P/poly$. □

**C.3 Theorem 21**

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

More precisely, we are going to prove:

**Theorem 28.** The evaluation problem for PE-queries over data instances of the form $A_m^\alpha$ is NP-hard.

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

The query $q_m(x)$ takes the form $q_m(x) = \exists z \left((r(x, z) \land s(x, z) \land t(x, z))\right)$, where the subqueries (without quantified variables) $r$, $s$ and $t$ and the variables $z$ are defined as follows. Among the variables $z$, there are variables $z_1,\ldots,z_m$ corresponding to the leaves of $A_m^\alpha$, variables $x_1,\ldots,x_k$ corresponding to the propositional variables of $\varphi_k$, and variables $x'_1,\ldots,x'_k$ corresponding to their negations (there are other auxiliary variables which will be introduced later on).

Now we will describe the subqueries $r$, $s$, $t$ of $q_m$. The subquery $r$ expresses that the variables $z_1,\ldots,z_m$ in-
deed correspond to the clauses of $\varphi_k$; it takes the form $r = \bigwedge_{i=1}^m r_i$. Each $r_i$ corresponds to a leaf of $A^\alpha_m$. Consider a path from the root $a$ to this $i$th leaf. Let $P_1, \ldots, P_\ell$ be the sequence of labels on the edges of this path, that is, each $P_i$ is either $P_-$ or $P_+$. Then

$$r_i = P_i(x, y_1^i) \land P_2(y_1^i, y_2^i) \land \cdots \land P_\ell(y_\ell^i, z_i),$$

where $y_1^i, \ldots, y_\ell^i$ are variables among $z$.

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

$$P_\pm(x, u_1^i) \land P_\pm(u_1^i, u_2^i) \land \cdots \land P_\pm(u_{\ell-2}^i, u_{\ell-1}^i) \land \left[ (P_\pm(u_{\ell-1}^i, x_i) \land P_\pm(x_i', u_{\ell-1}^i) \land B_0(x_i)) \lor (P_\pm(u_{\ell-1}^i, x_i') \land P_\pm(x_i, u_{\ell-1}^i) \land B_0(x_i')) \right],$$

where $u_1^i, \ldots, u_{\ell-1}^i$ are variables among $z$ and $P_\pm(x, y) = P_-(x, y) \lor P_+(x, y)$.

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

$$t_i = B_0(z_i) \lor B_0(l_{i,1}) \lor B_0(l_{i,2}) \lor B_0(l_{i,3}).$$

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

\[ \square \]

### C.4 Theorem 22

**Theorem 22.** There is an ontology $T_1$ such that answering OMOs of the form $(T, q)$ with Boolean linear CQs $q$ is LOGCFL-hard for query complexity.

**Proof.** Our proof encodes the hardest LOGCFL language $L$ [26] as formulated in [52]. The language $L$ enjoys the following property: for every language $L'$ over the alphabet $\Sigma'$ in LOGCFL, there exists a logspace transducer $\tau$ converting words over $\Sigma'$ to the words over the alphabet $\Sigma$ of $L$ in the sense that $w \in L'$ iff $\tau(w) \in L$. We construct an ontology $T_1$ and a logspace transducer that converts the words $w \in \Sigma^*$ to linear Boolean CQs $q_w$ such that

$$w \in L \iff T_1, \{A(a)\} \models q_w.$$  

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

\begin{align*}
S & \rightarrow SS, & S & \rightarrow \epsilon, \\
S & \rightarrow a_1 S b_1, & S & \rightarrow a_2 S b_2.
\end{align*}

With each word $w = c_0 \ldots c_n$ over $\Sigma_0$ we associate conjunction $\gamma_w(u_0, v_0, \ldots, u_n, v_n, u_{n+1})$ of the following atoms:

$$R_{c_0}(u_0, v_0), S_{c_0}(v_0, u_1), R_{c_1}(u_1, v_1), S_{c_1}(v_1, u_2), \ldots,$$

$$R_{c_n}(u_n, v_n), S_{c_n}(v_n, u_{n+1}),$$

where $R_c$ and $S_c$ are binary predicates, for $c \in \Sigma_0$. Let $T_0$ contain the following axioms, for $i = 1, 2$:

$$D(x) \rightarrow \exists y \left( R_a(x, y) \land S_b(y, x) \land (11) \right)$$

$$\exists z \left( S_a(y, z) \land R_b(z, y) \land D(z) \right).$$

An initial part of the canonical model of $(T_0, \{A(a), D(a)\})$ encoded by these axioms is shown below:

\[ (each large gray node belongs to $D$, each solid arrow with label $c$ belongs to $R_c$ and each dashed arrow with label $c$ to $S_c$, for $c \in \Sigma_0). \]

Let $q_w^A$ be the following linear Boolean CQ:

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

The following claim can readily be verified:

**Proposition 29.** For every $w \in \Sigma_0^*$, we have $w \in B_0$ iff $T_0, \{A(a), D(a)\} \models q_w^A$.

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

$$L = \{ [x_1 y_1 z_1] [x_2 y_2 z_2] \ldots [x_k y_k z_k] \mid k \geq 1,$$

$$x_i \in (\Sigma_0 \cup \{\#\})^* \cup \{\epsilon\} \text{ and }$$

$$z_i \in \{\epsilon\} \cup (\{\#\} \Sigma_0 \cup \{\#\})^* \text{ for all } i \leq k,$$

$$\text{ and } y_1 y_2 \ldots y_k \in B_0 \}. \]$$

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

$$[a_1 a_2 \# b_2 b_1] \notin L.$$  

(12)
\[ [a_1 a_2 \# b_2 b_1] [b_2 b_1] \in \mathcal{L}, \quad (13) \]
\[ [a_1 a_2 \# b_2 b_1] [a_1 b_1] \notin \mathcal{L}, \quad (14) \]
\[ \# a_1 a_2 \# b_2 b_1] [a_1 b_1] \in \mathcal{L}. \quad (15) \]

We say that a word \( w \) over \( \Sigma \) is block-formed if the following conditions are satisfied:
- the word begins with \([ \) and ends with \( ]\),
- after each \([ \) there is no \([ \) before \(] \);
- each non-final \([ \) is followed immediately by \( ]\);
- between each pair of matching \([ \) and \( ]\) there is at least one symbol.

With these definitions at hand, we first describe a log-space transducer that, given a word \( w \) over \( \Sigma \), returns a linear Boolean CQ \( q_w \) with binary predicates \( R_c \) and \( S_c \), for \( c \in \Sigma \), and unary predicates \( A \) and \( E \). If the word \( w = c_0 \ldots c_m \) is block-formed, then \( q_w \) consists of the following axioms:

\[ A(u_0) \land \gamma_w(u_0, v_0, \ldots, u_n, v_n, u_{n+1}) \land A(u_{n+1}). \]

Otherwise, the transducer returns a query that consists of a prefix of \( A(u_0) \land \gamma_w(u_0, v_0, \ldots, u_n, v_n, u_{n+1}) \) and ends in \( E(u_i) \), for some \( i \), which will indicate an error (as all queries containing \( E \) will be false in \( \mathcal{T}_1, \{A(a)\} \)). It is straightforward to verify that the required transducer can be implemented in \( L \).

Let \( \mathcal{T}_1 \) contain the two axioms (11) and the following axioms:

\[ A(x) \rightarrow D(x), \quad (16) \]
\[ D(x) \rightarrow \exists y (R_1(x, y) \land S_1(y, x)), \quad (17) \]
\[ D(x) \rightarrow \exists y (R_1(x, y) \land S_0(y, x) \land \exists z (S_0(y, z) \land R_0(z, y) \land F(z))), \quad (18) \]
\[ D(x) \rightarrow \exists y (R_1(x, y) \land S_1(y, x)), \quad (19) \]
\[ D(x) \rightarrow \exists y (R_0(x, y) \land S_1(y, x) \land \exists z (S_0(y, z) \land R_1(z, y) \land F(z))), \quad (20) \]
\[ F(x) \rightarrow \exists y (R_c(x, y) \land S_c(y, x)), \quad (21) \]

for \( c \in \Sigma_0 \cup \{\#\} \).

The four additional branches of the canonical model of \( \mathcal{T}_1, \{A(a)\} \) at each point in \( D \) are shown below:

![Diagram showing the four additional branches of the canonical model]({attachment:image.png})

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

Let \( q_w^D \) be defined identically to \( q_w^A \) except that the two occurrences of \( A \) are replaced by \( D \). The following property is established similarly to Proposition 29:

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

\[ w = [x, \text{ for } x \in (\Sigma_0 \cup \{\#\})^* \{\#\} \cup \{\}, \quad \text{iff} \quad \{17, 18, 21\}, \{D(d)\} \models q_w^D. \]

For any block-formed word \( w \in \Sigma^* \),

\[ w = [z, \text{ for } z \in \{\} \cup \{\} \cup \{\}^* \{\}, \quad \text{iff} \quad \{19, 20, 21\}, \{D(d)\} \models q_w^D. \]

With these properties established, it can readily be verified that \( \mathcal{T}_1, \{A(a)\} \not\models q_w \) iff \( w \in \mathcal{L} \). Consider a block-formed word \( w \in \Sigma^* \). Let \([w_1 \# w_2 \# \ldots \# w_n]\) be its \( m \)-th block and \( w_j = w_{m} \) (that is, \( w_j \) is the segment of the \( B_0 \)-word in this block). By Proposition 30, the subtree generated by (18) matches the (translation of) \([w_1 \# \ldots \# w_{j-1} \#] \), whereas the subtree generated by (20) matches \#\(w_{j+1} \# \ldots \# w_n\). By Proposition 29, the \( w_j \) itself is mapped into the main tree generated by (11). Note that (17) and (19) are needed for the case when \( j = 1 \) and \( j = n \), respectively. Finally, observe that (the translation of) \( w \) has to be mapped starting from \( a \) (the root of the tree) and ending at \( a \), and that the tree of the canonical model does not contain concept \( E \), so only a block-formed \( w \) can be mapped to the canonical model. In particular, \( \mathcal{T}_1, \{A(a)\} \not\models q_w \) for \( w \) of (12) and (14), and \( \mathcal{T}_1, \{A(a)\} \models q_w \) for \( w \) of (13) and (15).

**D. EXPERIMENTS**

**D.1 Computing rewritings**

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

We considered the following three sequences:

\[ RRSRSRSRRSRRSSR, \quad (\text{Sequence 1}) \]
\[ SRRRRRSRRRRRRRRRRRRRRRRRR, \quad (\text{Sequence 2}) \]
\[ SRRRSSRSRRRRRRRRRRRRRRRRRRRRR, \quad (\text{Sequence 3}) \]

For each of the three sequences, we consider the line-shaped queries with 1–15 atoms formed by their prefixes. Table 1 presents the sizes of the different types of rewritings.
We see in Table 3 that for most queries in Sequence 1 rewriting shows the best performance, while for Sequences 2 and 3 algorithms Log and Tw* are the winners (Tables 4 and 5). Note also that even within a single sequence the results may vary with the number of atoms.

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

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

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

\[ G(x, y) \leftarrow S(x, z) \land P_{13}(z, y), \]

\[ P_{13}(x, y) \leftarrow R(x, z) \land R(z, y), \]
<table>
<thead>
<tr>
<th>data set</th>
<th>query size</th>
<th>evaluation time (sec)</th>
<th>no. of generated tuples</th>
<th>no. of answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.ttl</td>
<td>1.002</td>
<td>0.022</td>
<td>0.039</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>2.003</td>
<td>0.035</td>
<td>0.031</td>
<td>0.006</td>
</tr>
<tr>
<td></td>
<td>3.005</td>
<td>0.043</td>
<td>0.036</td>
<td>0.006</td>
</tr>
<tr>
<td></td>
<td>4.007</td>
<td>0.054</td>
<td>0.031</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>5.008</td>
<td>0.067</td>
<td>0.028</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>6.009</td>
<td>0.078</td>
<td>0.026</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>7.010</td>
<td>0.081</td>
<td>0.032</td>
<td>0.007</td>
</tr>
<tr>
<td></td>
<td>8.012</td>
<td>0.085</td>
<td>0.033</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>9.013</td>
<td>0.086</td>
<td>0.034</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>10.014</td>
<td>0.089</td>
<td>0.035</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>11.015</td>
<td>0.091</td>
<td>0.036</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>12.016</td>
<td>0.093</td>
<td>0.037</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>13.017</td>
<td>0.095</td>
<td>0.038</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>14.018</td>
<td>0.096</td>
<td>0.039</td>
<td>0.011</td>
</tr>
<tr>
<td></td>
<td>15.019</td>
<td>0.098</td>
<td>0.040</td>
<td>0.011</td>
</tr>
</tbody>
</table>

Table 3: Evaluating rewritings on RDFox - 1
<table>
<thead>
<tr>
<th>data- set</th>
<th>query size</th>
<th>evaluation time (sec)</th>
<th>no. of answers</th>
<th>no. of generated tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hapto</td>
<td>Upper</td>
<td>Presto</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hapto</td>
<td>Upper</td>
<td>Presto</td>
</tr>
<tr>
<td>1.ttl</td>
<td>0.009</td>
<td>0.005</td>
<td>0.005</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>2.ttl</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>3.ttl</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>4.ttl</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td>5.ttl</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
<td>0.009</td>
</tr>
</tbody>
</table>

**Table 4: Evaluating rewritings on RDFox - 2**
<table>
<thead>
<tr>
<th>dataset</th>
<th>query</th>
<th>evaluation time (sec)</th>
<th>no of answers</th>
<th>no of generated tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepToC</td>
<td>Cipper</td>
<td>Presto</td>
<td>LIN</td>
<td>Loc</td>
</tr>
<tr>
<td>1</td>
<td>0.004</td>
<td>0.003</td>
<td>0.004</td>
<td>0.021</td>
</tr>
<tr>
<td>2</td>
<td>0.006</td>
<td>0.006</td>
<td>0.017</td>
<td>0.022</td>
</tr>
<tr>
<td>3</td>
<td>0.053</td>
<td>0.065</td>
<td>0.849</td>
<td>0.897</td>
</tr>
<tr>
<td>4</td>
<td>0.012</td>
<td>0.01</td>
<td>0.074</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>0.011</td>
<td>0.009</td>
<td>0.07</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>0.018</td>
<td>0.015</td>
<td>0.139</td>
<td>0.023</td>
</tr>
<tr>
<td>7</td>
<td>0.017</td>
<td>0.015</td>
<td>0.145</td>
<td>0.01</td>
</tr>
<tr>
<td>8</td>
<td>0.025</td>
<td>0.034</td>
<td>0.329</td>
<td>0.026</td>
</tr>
<tr>
<td>9</td>
<td>0.025</td>
<td>0.043</td>
<td>0.433</td>
<td>0.01</td>
</tr>
<tr>
<td>10</td>
<td>0.035</td>
<td>0.086</td>
<td>0.549</td>
<td>0.029</td>
</tr>
<tr>
<td>11</td>
<td>0.034</td>
<td>0.086</td>
<td>1.445</td>
<td>1.164</td>
</tr>
<tr>
<td>12</td>
<td>0.048</td>
<td>0.223</td>
<td>4.877</td>
<td>0.013</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>-</td>
<td>13.007</td>
<td>0.141</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>382.922</td>
<td>3.738</td>
</tr>
<tr>
<td>15</td>
<td>0.02</td>
<td>0.037</td>
<td>387.714</td>
<td>0.017</td>
</tr>
</tbody>
</table>

Table 5: Evaluating rewritings on RDFox - 3

<table>
<thead>
<tr>
<th>dataset</th>
<th>query</th>
<th>evaluation time (sec)</th>
<th>no of answers</th>
<th>no of generated tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepToC</td>
<td>Cipper</td>
<td>Presto</td>
<td>LIN</td>
<td>Loc</td>
</tr>
<tr>
<td>1</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
<td>0.004</td>
</tr>
<tr>
<td>2</td>
<td>0.006</td>
<td>0.006</td>
<td>0.02</td>
<td>0.023</td>
</tr>
<tr>
<td>3</td>
<td>0.022</td>
<td>0.019</td>
<td>0.04</td>
<td>0.039</td>
</tr>
<tr>
<td>4</td>
<td>0.013</td>
<td>0.011</td>
<td>0.047</td>
<td>0.01</td>
</tr>
<tr>
<td>5</td>
<td>0.012</td>
<td>0.011</td>
<td>0.044</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>0.028</td>
<td>0.016</td>
<td>0.081</td>
<td>0.027</td>
</tr>
<tr>
<td>7</td>
<td>0.018</td>
<td>0.015</td>
<td>0.094</td>
<td>0.011</td>
</tr>
<tr>
<td>8</td>
<td>0.025</td>
<td>0.036</td>
<td>0.182</td>
<td>0.027</td>
</tr>
<tr>
<td>9</td>
<td>0.026</td>
<td>0.037</td>
<td>0.215</td>
<td>0.012</td>
</tr>
<tr>
<td>10</td>
<td>0.038</td>
<td>0.091</td>
<td>0.327</td>
<td>0.028</td>
</tr>
<tr>
<td>11</td>
<td>0.036</td>
<td>0.09</td>
<td>1.467</td>
<td>0.345</td>
</tr>
<tr>
<td>12</td>
<td>0.052</td>
<td>0.268</td>
<td>1.888</td>
<td>0.014</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>-</td>
<td>4.579</td>
<td>0.032</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
<td>26.213</td>
<td>0.38</td>
</tr>
<tr>
<td>15</td>
<td>-</td>
<td>-</td>
<td>26.689</td>
<td>0.017</td>
</tr>
</tbody>
</table>

Table 5: Evaluating rewritings on RDFox - 3
\[ G(x, y) \leftarrow A_P(x) \land R(x, y) \]

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

\[
\begin{align*}
G(x, y) &\leftarrow S(x, z) \land R(x, v) \land R(v, y), \\
G(x, y) &\leftarrow A_P(x) \land R(x, y),
\end{align*}
\]

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