Tree-like Queries in OWL 2 QL: Succinctness and Complexity Results

Meghyn Bienvenu*, Stanislav Kikot† and Vladimir Podolskii‡
*Laboratoire de Recherche en Informatique, CNRS & Université Paris-Sud, Orsay, France
†Institute for Information Transmission Problems & MIPT, Moscow, Russia
‡Steklov Mathematical Institute, Moscow, Russia

***Abstract*—This paper investigates the impact of query topology on the difficulty of answering conjunctive queries in the presence of OWL 2 QL ontologies. Our first contribution is to clarify the worst-case size of positive existential (PE), non-recursive Datalog (NDL), and first-order (FO) rewritings for various classes of tree-like conjunctive queries, ranging from linear queries to bounded treewidth queries. Perhaps our most surprising result is a superpolynomial lower bound on the size of PE-rewritings that holds already for linear queries and ontologies of depth 2. More positively, we show that polynomial-size NDL-rewritings always exist for tree-shaped queries with a bounded number of leaves (and arbitrary ontologies), and for bounded treewidth queries paired with bounded depth ontologies. For FO-rewritings, we equate the existence of polysize rewritings with well-known problems in Boolean circuit complexity. As our second contribution, we analyze the computational complexity of query answering and establish tractability results (either NL- or LOGCFL-completeness) for a range of query-ontology pairs. Combining our new results with those from the literature yields a complete picture of the succinctness and complexity landscapes for the considered classes of queries and ontologies.**

I. Introduction

Recent years have witnessed a growing interest from both the knowledge representation and database communities in *ontology-based data access* (OBDA), in which the conceptual knowledge provided by an ontology is exploited when querying data. Formally, given an ontology $\mathcal{T}$ (logical theory), a data instance $\mathcal{A}$ (set of ground facts), and a conjunctive query (CQ) $\mathbf{q}(\mathbf{x})$, the problem is to compute the *certain answers* to $\mathbf{q}$, that is, the tuples of constants $\mathbf{a}$ that satisfy $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{a})$.

As scalability is crucial in data-intensive applications, much of the work on OBDA focuses on so-called 'lightweight' ontology languages, which provide useful modelling features while retaining good computational properties. The DL-Lite family [1] of lightweight description logics has played a particularly prominent role, as witnessed by the recent introduction of the OWL 2 QL profile [2] (based upon DL-Lite) into the W3C-endorsed ontology language OWL 2. The popularity of these languages is due to fact that they enjoy *first-order (FO) rewritability*, which means that for every CQ $\mathbf{q}(\mathbf{x})$ and ontology $\mathcal{T}$, there exists a computable FO-query $\mathbf{q}'(\mathbf{x})$ (called a *rewriting*) such that the certain answers to $\mathbf{q}(\mathbf{x})$ over $(\mathcal{T}, \mathcal{A})$ coincide with the answers of $\mathbf{q}'(\mathbf{x})$ over the data instance $\mathcal{A}$ (viewed as an FO interpretation). First-order rewritability provides a means of reducing the *entailment* problem of identifying certain answers to the simpler problem of FO *model checking*; the latter can be rephrased as SQL query evaluation and delegated to highly-optimized relational database management systems (RDBMSs). This appealing theoretical result spurred the development of numerous query rewriting algorithms for OWL 2 QL and its extensions, cf. [1], [3]–[11]. Most produce rewritings expressed as unions of conjunctive queries (UCQs), and experimental evaluation has shown that such rewritings may be huge, making them difficult, or even impossible, to evaluate using RDBMSs.

The aim of this paper is to gain a better understanding of the difficulty of query rewriting and query answering in OWL 2 QL and how it varies depending on the topology of the query.

Succinctness of Query Rewriting It is not difficult to see that exponential-size rewritings are unavoidable if rewritings are given as UCQs (consider the query $B_1(x) \wedge \ldots \wedge B_n(x)$ and ontology $\{A_i(x) \rightarrow B_i(x) \mid 1 \leq i \leq n\}$). A natural question is whether an exponential blowup can be avoided by moving to other standard query languages, like positive existential (PE) queries, non-recursive datalog (NDL) queries, or first-order (FO-) queries. More generally, under what conditions can we ensure polynomial-size rewritings? A first (negative) answer was given in [12], which proved exponential lower bounds for the worst-case size of PE- and NDL-rewritings, as well as a superpolynomial lower bound for FO-rewritings (under the widely-held assumption that NP $\not\subseteq$ P/poly). Interestingly, all three results hold already for *tree-shaped* CQs, which are a well-studied and practically relevant class of CQs that often enjoy better computational properties, cf. [15], [16]. While the queries used in the proofs had a simple structure, the ontologies induced full binary trees of depth n. This raised the question of whether better results could be obtained by considering restricted classes of ontologies. A recent study [13] explored this question for ontologies of *depth* 1 and 2, that is, ontologies for which the trees of labelled nulls appearing in the canonical model (aka chase) are guaranteed to be of depth at most 1 or 2 (see Section II for a formal definition). It was shown that for depth 1 ontologies, polysize PE-rewritings do not exist, polysize NDL-rewritings do exist, and polysize FO-rewritings exist iff NL/poly $\subseteq$ NC^1. For depth 2 ontologies, neither polysize PE- nor NDL-rewritings exist, and polysize FO-rewritings do not exist unless NP $\not\subseteq$ P/poly. These results used simpler ontologies, but the considered CQs were no longer tree-shaped. For depth 1 ontologies, this distinction

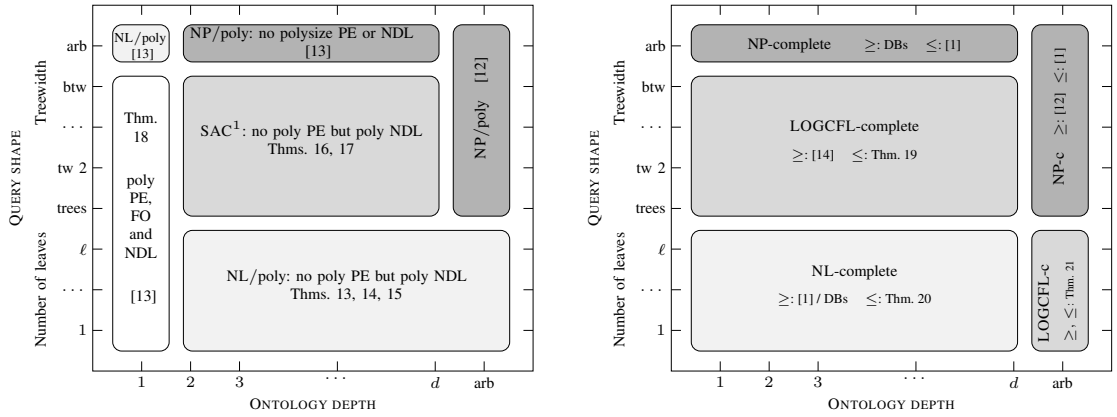


Fig. 1. Succinctness landscape for query rewriting [left] and complexity landscape for query answering [right]. We use the following abbreviations: 'arb' for 'arbitrary', '(b)tw' for '(bounded) treewidth', 'poly' for 'polynomial-size', 'DBs' for 'inherited from databases', and 'c' for 'complete'. On the left, 'NP/poly' indicates that 'polysize FO-rewritings only if NP/poly $\subseteq$ NC^1' and $C \in \{\text{NL/poly}, \text{SAC}^1\}$ means 'polysize FO-rewritings iff $C \subseteq \text{NC}^1$'.

is crucial, as it was further shown in [13] that polysize PE-rewritings *do* exist for tree-shaped CQs.

While existing results go a fair ways towards understanding the succinctness landscape of query rewriting in OWL 2 QL, a number of questions remain open:

- What happens if we consider tree-shaped queries and bounded depth ontologies?
- What happens if we consider generalizations or restrictions of tree-shaped CQs?

Complexity of Query Answering Succinctness results help us understand when polysize rewritings are possible, but they say little about the complexity of query answering itself. On the one hand, the existence of polysize rewritings is not sufficient to guarantee efficient query answering, since small rewritings may nonetheless be difficult to produce and/or evaluate. On the other hand, negative results show that query rewriting may not always be practicable, but they leave open whether another approach to query answering might yield better results. It is therefore important to investigate the complexity landscape of query answering, independently of any algorithmic approach.

We briefly review the relevant literature. In relational databases, it is well-known that CQ answering is NP-complete in the general case. A seminal result by Yannakakis established the tractability of answering tree-shaped CQs [15], and this result was later extended to wider classes of queries, most notably to bounded treewidth CQs [17]. Gottlob et al. [18] pinpointed the precise complexity of answering tree-shaped and bounded treewidth CQs, showing both problems to be complete for the class LOGCFL of all languages logspace-reducible to context-free languages [19]. In the presence of arbitrary OWL 2 QL ontologies, the NP upper bound for arbitrary CQs continues to hold [1], but answering tree-shaped queries becomes NP-hard [12]. Interestingly, the latter problem was recently proven tractable in [16] for DL-Lite$_{\mathsf{core}}$ (a slightly less expressive logic than OWL 2 QL), raising the hope that other restrictions might also yield tractability. We therefore have the following additional question:

- How do the aforementioned restrictions on queries and ontologies impact the complexity of query answering?

Contributions In this paper, we address the preceding questions by providing a complete picture of both the worst-case size of query rewritings and the complexity of query answering for tree-shaped queries, their restriction to *linear and bounded leaf queries* (i.e. tree-shaped CQs with a bounded number of leaves), and their generalization to *bounded treewidth queries*. Figure 1 gives an overview of new and existing results.

Regarding succinctness, we establish a superpolynomial lower bound on the size of PE-rewritings that holds already for linear queries and depth 2 ontologies, significantly strengthening earlier negative results. For NDL-rewritings, the situation is brighter: we show that polysize rewritings always exist for bounded branching queries (and arbitrary OWL 2 QL ontologies), and for bounded treewidth queries and bounded depth ontologies. We also prove that the succinctness problems concerning FO-rewritings are equivalent to well-known problems in circuit complexity: NL/poly $\subseteq$ NC^1 in the case of linear and bounded leaf queries, and SAC^1 $\subseteq$ NC^1 in the case of tree-shaped and bounded treewidth queries and bounded depth ontologies. Finally, to complete the succinctness landscape, we show that the result from [13] that all tree-shaped queries and depth 1 ontologies have polysize PE-rewritings generalizes to the wider class of bounded treewidth queries. To prove our results, we establish tight connections between Boolean functions induced by queries and ontologies and the non-uniform complexity classes NL/poly and SAC^1, reusing and further extending the machinery developed in [12], [13].

Our complexity analysis reveals that all query-ontology combinations that have not already been shown NP-hard are in fact tractable. Specifically, in the case of bounded depth ontologies, we prove membership in LOGCFL for bounded treewidth queries (generalizing the result in [18]) and membership in NL for bounded leaf queries. We also show LOGCFL-completeness for linear and bounded leaf queries in the presence of arbitrary OWL 2 QL ontologies. This last result is the most interesting technically, as upper and lower bounds rely on two different characterizations of the class LOGCFL.

For lack of space, some proofs are deferred to the appendix.

II. Preliminaries

A. Querying OWL 2 QL Knowledge Bases

We will work with the fragment of *OWL 2 QL* profile [2] that corresponds to the description logic *DL-Lite*$_R$ [1], as it covers the most important features of the language and simplifies the technical treatment. Moreover, to make the paper accessible to a wider audience, we eschew the more common OWL and description logic notations in favour of traditional first-order logic (FO) syntax.

1) Knowledge bases: We assume countably infinite, mutually disjoint sets N_1 and N_2 of *unary* and *binary predicate* names. We will typically use the characters A, B for unary predicates and P, R for binary predicates. For a binary predicate P, we will use P^- to denote the *inverse* of P and will *treat an atom* $P^-(t,t')$ *as shorthand for* $P(t',t)$ (by convention, $P^{--}=P$). The set of binary predicates and their inverses is denoted $\mathsf{N}_2^{\pm}$, and we use ϱ to refer to its elements.

An *OWL 2 QL knowledge base* (KB) can be seen as a *pair of FO theories* $(\mathcal{T},\mathcal{A})$, constructed using predicates from N_1 and N_2. The first theory $\mathcal{T}$, called the *ontology* (or TBox), consists of finitely many sentences (or *axioms*) of the forms

$$\forall x\,\big(\tau(x)\to\tau'(x)\big), \qquad \forall x,y\,\big(\varrho(x,y)\to\varrho'(x,y)\big),$$
$$\forall x\,\big(\tau(x)\wedge\tau'(x)\to\bot\big), \quad \forall x,y\,\big(\varrho(x,y)\wedge\varrho'(x,y)\to\bot\big),$$

where $\varrho\in\mathsf{N}_2^{\pm}$ (see earlier) and $\tau(x)$ is defined as follows:

$$\tau(x) \;::=\; A(x) \quad (A\in\mathsf{N}_1) \;\mid\; \exists y\,\varrho(x,y) \quad (\varrho\in\mathsf{N}_2^{\pm})$$

Note that to simplify notation, we will omit the universal quantifiers when writing ontology axioms. The *signature* of $\mathcal{T}$, denoted $\mathsf{sig}(\mathcal{T})$, is the set of predicate names in $\mathcal{T}$, and the *size* of $\mathcal{T}$, written $|\mathcal{T}|$, is the number of symbols in $\mathcal{T}$.

The second theory $\mathcal{A}$, called the *data instance* (or ABox), is a finite set of *ground facts*. We use $\mathsf{inds}(\mathcal{A})$ to denote the set of individual constants appearing in $\mathcal{A}$.

The *semantics* of KB $(\mathcal{T},\mathcal{A})$ is the standard FO semantics of $\mathcal{T}\cup\mathcal{A}$. Interpretations will be given as pairs $\mathcal{I}=(\Delta^{\mathcal{I}},\cdot^{\mathcal{I}})$, with $\Delta^{\mathcal{I}}$ the domain and $\cdot^{\mathcal{I}}$ the interpretation function; models, satisfaction, consistency, and entailment are defined as usual.

2) Query answering: A *conjunctive query* (CQ) $\mathbf{q}(\mathbf{x})$ is an FO formula $\exists\mathbf{y}\,\varphi(\mathbf{x},\mathbf{y})$, where φ is a conjunction of atoms of the forms $A(z_1)$ or $R(z_1,z_2)$ with $z_i\in\mathbf{x}\cup\mathbf{y}$. The free variables $\mathbf{x}$ are called *answer variables*. Note that we assume w.l.o.g. that CQs do not contain constants, and where convenient, we regard a CQ as the set of its atoms. We use $\mathsf{vars}(\mathbf{q})$ (resp. $\mathsf{avars}(\mathbf{q})$) to denote the set of variables (resp. answer variables) of $\mathbf{q}$. The *signature* and *size* of $\mathbf{q}$, defined similarly to above, are denoted $\mathsf{sig}(\mathbf{q})$ and $|\mathbf{q}|$ respectively.

A tuple $\mathbf{a}\subseteq\mathsf{inds}(\mathcal{A})$ is a *certain answer* to $\mathbf{q}(\mathbf{x})$ over $\mathcal{K}=(\mathcal{T},\mathcal{A})$ if $\mathcal{I}\models\mathbf{q}(\mathbf{a})$ for all $\mathcal{I}\models\mathcal{K}$; in this case we write $\mathcal{K}\models\mathbf{q}(\mathbf{a})$. By first-order semantics, $\mathcal{I}\models\mathbf{q}(\mathbf{a})$ iff there is a mapping $h:\mathsf{vars}(\mathbf{q})\to\Delta^{\mathcal{I}}$ such that (i) $h(z)\in A^{\mathcal{I}}$ whenever $A(z)\in\mathbf{q}$, (ii) $(h(z),h(z'))\in r^{\mathcal{I}}$ whenever $r(z,z')\in\mathbf{q}$, and (iii) h maps $\mathsf{avars}(\mathbf{q})$ to $\mathbf{a}^{\mathcal{I}}$. If the first two conditions are satisfied, then h is a *homomorphism* from $\mathbf{q}$ to $\mathcal{I}$, and we write $h\colon\mathbf{q}\to\mathcal{I}$. If (iii) also holds, then we write $h\colon\mathbf{q}(\mathbf{a})\to\mathcal{I}$.

3) Canonical model: We recall that every consistent *OWL 2 QL* KB $(\mathcal{T},\mathcal{A})$ possesses a *canonical model* (or chase) $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ with the property that

$$\mathcal{T},\mathcal{A}\models\mathbf{q}(\mathbf{a}) \quad \text{iff} \quad \mathcal{C}_{\mathcal{T},\mathcal{A}}\models\mathbf{q}(\mathbf{a}) \tag{1}$$

for every CQ $\mathbf{q}$ and tuple $\mathbf{a}\subseteq\mathsf{inds}(\mathcal{A})$. Thus, query answering in *OWL 2 QL* corresponds to *deciding existence of a homomorphism of the query into the canonical model*.

Informally, $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is obtained from $\mathcal{A}$ by repeatedly applying the axioms in $\mathcal{T}$, introducing fresh elements (labelled nulls) as needed to serve as witnesses for the existential quantifiers. Formally, the domain $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ consists of $\mathsf{inds}(\mathcal{A})$ and all words $a\varrho_1\varrho_2\dots\varrho_n$ ($n\geq 1$) with $a\in\mathsf{inds}(\mathcal{A})$ and $\varrho_i\in\mathsf{N}_2^{\pm}$ ($1\leq i\leq n$) such that

- $\mathcal{T},\mathcal{A}\models\exists y\varrho_1(a,y)$ and $\mathcal{T},\mathcal{A}\not\models\varrho_1(a,b)$ for any $b\in\mathsf{inds}(\mathcal{A})$;
- for every $1\leq i<n$: $\mathcal{T}\models\exists y\,\varrho_i(y,x)\to\exists y\,\varrho_{i+1}(x,y)$ and $\mathcal{T}\not\models\varrho_i(x,y)\to\varrho_{i+1}(y,x)$.

Predicate names are interpreted as follows:

$$\begin{aligned}
A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} &= \{a\in\mathsf{inds}(\mathcal{A})\mid\mathcal{T},\mathcal{A}\models A(a)\}\cup\\
&\qquad\{w\varrho\in\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}\mid\mathcal{T}\models\exists y\,\varrho(y,x)\to A(x)\}\\
P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} &= \{(a,b)\mid\mathcal{T},\mathcal{A}\models P(a,b)\}\cup\\
&\qquad\{(w,w\varrho)\mid\mathcal{T}\models\varrho(x,y)\to P(x,y)\}\cup\\
&\qquad\{(w\varrho,w)\mid\ \mathcal{T}\models\varrho(x,y)\to P(y,x)\}
\end{aligned}$$

Every constant $a\in\mathsf{inds}(\mathcal{A})$ is interpreted as itself: $a^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}=a$.

Many of our constructions will exploit the fact that the canonical model has a forest structure: there is a *core* involving the individual constants from the dataset and an *anonymous part* consisting of trees of labelled nulls rooted at the constants.

Example 1. *Figure 2 presents a KB* $(\mathcal{T}_0,\mathcal{A}_0)$*, its canonical model* $\mathcal{C}_{\mathcal{T}_0,\mathcal{A}_0}$*, a CQ* $\mathbf{q}_0$*, and a homomorphism* $\mathbf{q}_0(c,a)\to\mathcal{C}_{\mathcal{T}_0,\mathcal{A}_0}$ *witnessing that* (c,a) *is a certain answer to* $\mathbf{q}_0$*.*

4) Considered classes of queries and ontologies: For ontologies, the parameter of interest is the depth of an ontology. An ontology $\mathcal{T}$ is *of depth* ω if there is a data instance $\mathcal{A}$ such that the domain of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is infinite; $\mathcal{T}$ is *of depth* d, $0\leq d<\omega$, if d is the greatest number such that some $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ contains an element of the form $a\varrho_1\dots\varrho_d$. The depth of $\mathcal{T}$ can be computed in polynomial time, and if $\mathcal{T}$ is of finite depth, then its depth cannot exceed $2|\mathcal{T}|$.

The different classes of tree-like queries considered in this paper are defined by associating with every CQ $\mathbf{q}$ the undirected graph $G_{\mathbf{q}}$ whose vertices are the variables of $\mathbf{q}$, and which contains an edge $\{u,v\}$ whenever $\mathbf{q}$ contains some atom $R(u,v)$ or $R(v,u)$. We call a CQ $\mathbf{q}$ *tree-shaped* if the graph $G_{\mathbf{q}}$ is acyclic, and we say that $\mathbf{q}$ has k *leaves* if the graph $G_{\mathbf{q}}$ contains exactly k vertices of degree 1. A *linear* CQ is a tree-shaped CQ with 2 leaves.

The most general class of queries we consider are *bounded treewidth queries*. We recall that a *tree decomposition* of an undirected graph $G=(V,E)$ is a pair (T,λ) such that T is an (undirected) tree and λ assigns a label $\lambda(N)\subseteq V$ to every node N of T such that the following conditions are satisfied:

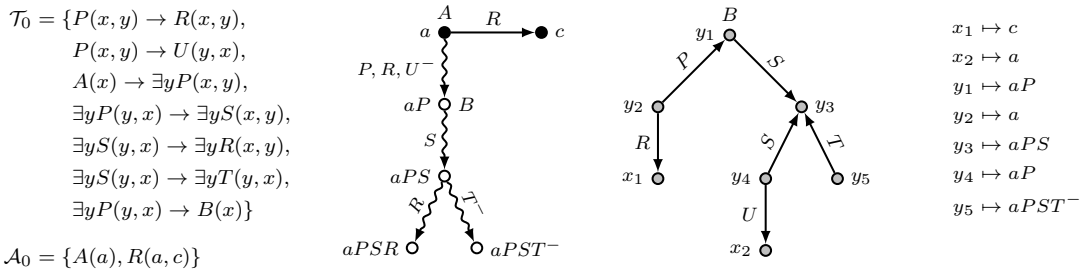


Fig. 2. From left to right: example KB $(\mathcal{T}_0, \mathcal{A}_0)$, canonical model $\mathcal{C}_{\mathcal{T}_0,\mathcal{A}_0}$, tree-shaped query $\mathbf{q}_0(x_1, x_2)$, and homomorphism $h_0 : \mathbf{q}_0(c, a) \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$.

1) For every $v \in V$, there exists a node N with $v \in \lambda(N)$.
2) For every $e \in E$, there exists a node N with $e \subseteq \lambda(N)$.
3) For every $v \in V$, the nodes $\{N \mid v \in \lambda(N)\}$ induce a connected subtree of T.

The *width of a tree decomposition* (T, λ) is equal to $\mathsf{max}_N|\lambda(N)| - 1$, and the *treewidth of a graph* G is the minimum width over all tree decompositions of G. The *treewidth of a CQ* $\mathbf{q}$ is defined as the treewidth of the graph $G_{\mathbf{q}}$.

B. Query Rewriting and Boolean Functions

We next recall the definition of query rewriting and show how the (worst-case) size of rewritings can be related to representations of particular Boolean functions. We assume the reader is familiar with Boolean circuits [20], [21], built using AND, OR, NOT and input gates. The *size of a circuit* $\boldsymbol{C}$, denoted $|\boldsymbol{C}|$, is defined as its number of gates. We will be particularly interested in *monotone circuits* (that is, circuits with no NOT gates). (Monotone) *formulas* are (monotone) circuits whose underlying graph is a tree.

1) Query rewriting: With every data instance $\mathcal{A}$, we associate the interpretation $\mathcal{I}_{\mathcal{A}}$ whose domain is $\mathsf{inds}(\mathcal{A})$ and whose interpretation function makes true precisely the facts in $\mathcal{A}$. We say an FO formula $\mathbf{q}'(\mathbf{x})$ with free variables $\mathbf{x}$ and without constants is an *FO-rewriting of CQ* $\mathbf{q}(\mathbf{x})$ *and ontology* $\mathcal{T}$ if, for any data instance $\mathcal{A}$ and tuple $\mathbf{a} \subseteq \mathsf{inds}(\mathcal{A})$, we have $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{a})$ iff $\mathcal{I}_{\mathcal{A}} \models \mathbf{q}'(\mathbf{a})$. If $\mathbf{q}'$ is a positive existential formula (i.e. it only uses $\exists$, $\wedge$, $\vee$), then it is called a *PE-rewriting* of $\mathbf{q}$ and $\mathcal{T}$.

We also consider rewritings in the form of nonrecursive Datalog queries. We remind the reader that a *Datalog program* is a finite set of rules $\forall \mathbf{x}\, (\gamma_1 \wedge \dots \wedge \gamma_m \to \gamma_0)$, where each γ_i is an atom of the form $G(x_1, \dots, x_l)$ with $x_i \in \mathbf{x}$. The atom γ_0 is called the *head* of the rule, and $\gamma_1, \dots, \gamma_m$ its *body*. All variables in the head must also occur in the body. A predicate G *depends* on a predicate H in program Π if Π contains a rule whose head predicate is G and whose body contains H. The program Π is called *nonrecursive* if there are no cycles in the dependence relation for Π. For a nonrecursive Datalog program Π and a predicate goal, we say that (Π, goal) is an *NDL-rewriting of* $\mathbf{q}$ *and* $\mathcal{T}$ in case $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{a})$ iff $\Pi, \mathcal{A} \models \mathsf{goal}(\mathbf{a})$, for every data instance $\mathcal{A}$ and tuple $\mathbf{a} \subseteq \mathsf{inds}(\mathcal{A})$.

Remark 2. *In this paper, we focus on so-called* pure rewritings*, as considered in [12], [13] and used in existing query rewriting systems. Impure rewritings, which exploit existential quantification over fixed constants, behave differently regarding succinctness [22]. Please see [23] for detailed discussion.*

2) Upper bounds via tree witness functions: The upper bounds on rewriting size shown in [13] rely on associating a Boolean function $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ with every query $\mathbf{q}$ and ontology $\mathcal{T}$. The definition of the function $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ makes essential use of the notion of tree witness [24], which we recall next.

For every $\varrho \in \mathsf{N}_2^{\pm}$, we let $\mathcal{C}^{\varrho}_{\mathcal{T}}$ be the canonical model of the KB $(\mathcal{T} \cup \{A_\varrho(x) \to \exists y \varrho(x,y)\}, \{A_\varrho(a)\})$, where A_ϱ is a fresh unary predicate. Suppose that $\mathbf{q}' \subseteq \mathbf{q}$ (recall that we view queries as sets of atoms) and there is a homomorphism $h\colon \mathbf{q}' \to \mathcal{C}^{\varrho}_{\mathcal{T}}$ such that $h(x) = a$ for every $x \in \mathsf{avars}(\mathbf{q})$. Let $\mathfrak{t}_\mathsf{r} = \{x \in \mathsf{vars}(\mathbf{q}') \mid h(x) = a\}$ and $\mathfrak{t}_\mathsf{i} = \mathsf{vars}(\mathbf{q}') \setminus \mathfrak{t}_\mathsf{r}$. We call the pair $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ a *tree witness for* $\mathbf{q}$ *and* $\mathcal{T}$ *generated by* ϱ if $\mathfrak{t}_\mathsf{i} \neq \emptyset$ and $\mathbf{q}'$ is a *minimal* subset of $\mathbf{q}$ such that, for every $y \in \mathfrak{t}_\mathsf{i}$, every atom in $\mathbf{q}$ containing y belongs to $\mathbf{q}'$. In this case, we denote $\mathbf{q}'$ by $\mathbf{q}_\mathfrak{t}$. We denote by $\Theta^{\mathbf{q}}_{\mathcal{T}}$ (resp. $\Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]$) the set of tree witnesses for $\mathbf{q}$ and $\mathcal{T}$ (resp. generated by ϱ).

Example 3. *There are 3 tree witnesses for* $\mathbf{q}_0$ *and* $\mathcal{T}_0$*:* $\mathfrak{t}^1 = (\{x_2, y_2\}, \{y_1, y_3, y_4, y_5\})$*,* $\mathfrak{t}^2 = (\{y_1, y_4\}, \{y_3, y_5\})$*, and* $\mathfrak{t}^3 = (\{y_3\}, \{y_5\})$*, generated by* P*,* S*, and* T^- *respectively.*

Every homomorphism of $\mathbf{q}$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ induces a partition of $\mathbf{q}$ into subqueries $\mathbf{q}_{\mathfrak{t}_1}, \dots, \mathbf{q}_{\mathfrak{t}_n}$ $(\mathfrak{t}_i \in \Theta^{\mathbf{q}}_{\mathcal{T}})$ that are mapped into the anonymous part and the remaining atoms that are mapped into the core. The *tree witness function for* $\mathbf{q}$ *and* $\mathcal{T}$ captures the different ways of partitioning $\mathbf{q}$:

$$f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}} = \bigvee_{\substack{\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}} \\ \text{independent}}} \left(\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} p_\eta \wedge \bigwedge_{\mathfrak{t} \in \Theta} p_\mathfrak{t} \right)$$

Here p_η and $p_\mathfrak{t}$ are Boolean variables, $\mathbf{q}_\Theta$ stands for $\bigcup_{\mathfrak{t} \in \Theta} \mathbf{q}_\mathfrak{t}$, and '$\Theta$ independent' means $\mathbf{q}_\mathfrak{t} \cap \mathbf{q}_{\mathfrak{t}'} = \emptyset$ for all $\mathfrak{t} \neq \mathfrak{t}' \in \Theta$.

In [13], it is shown how a Boolean formula or circuit computing $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ can be transformed into a rewriting of $\mathbf{q}$ and $\mathcal{T}$. Thus, the circuit complexity of $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ provides an upper bound on the size of rewritings of $\mathbf{q}$ and $\mathcal{T}$.

Theorem 4 (from [13])**.** *If* $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ *is computed by a* (*monotone*) *Boolean formula* χ *then there is a* (*PE-*) *FO-rewriting of* $\mathbf{q}$ *and* $\mathcal{T}$ *of size* $O(|\chi| \cdot |\mathbf{q}| \cdot |\mathcal{T}|)$*.*

If $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ *is computed by a monotone Boolean circuit* $\boldsymbol{C}$ *then there is an NDL-rewriting of* $\mathbf{q}$ *and* $\mathcal{T}$ *of size* $O(|\boldsymbol{C}| \cdot |\mathbf{q}| \cdot |\mathcal{T}|)$*.*

Observe that $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ contains a variable $p_\mathfrak{t}$ for every tree witness $\mathfrak{t}$, and so it can only be used to show polynomial

upper bounds in cases where $|\Theta^{\mathbf{q}}_{\mathcal{T}}|$ is bounded polynomially in $|\mathbf{q}|$ and $|\mathcal{T}|$. We therefore introduce the following variant:

$$f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}} = \bigvee_{\substack{\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}} \\ \text{independent}}} \Big(\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} p_\eta \wedge \bigwedge_{\mathfrak{t} \in \Theta} \big(\bigwedge_{z,z' \in \mathfrak{t}} p_{z=z'} \wedge \bigvee_{\substack{\varrho \in \mathsf{N}^{\pm}_2, \, z \in \mathfrak{t} \\ \mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]}} \bigwedge p^{\varrho}_z \big) \Big)$$

Intuitively, we use $p_{z=z'}$ to enforce that variables z and z' are mapped to elements of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ that begin by the same individual constant and p^{ϱ}_z to ensure z is mapped to an element whose initial constant a satisfies $\mathcal{T}, \mathcal{A} \models \exists y \varrho(a, y)$.

We observe that the number of variables in $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ is polynomially bounded in $|\mathbf{q}|$ and $|\mathcal{T}|$. Moreover, we can prove that it has the same properties as $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ regarding upper bounds.

Theorem 5. *Thm. 4 remains true if $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ is replaced by $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$.*

3) Lower bounds via primitive evaluation functions: In order to obtain lower bounds on the size of rewritings, we associate with each pair $(\mathbf{q}, \mathcal{T})$ a third function $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ that describes the result of evaluating $\mathbf{q}$ over data instances containing a single individual constant. Given an assignment $\gamma : \mathsf{sig}(\mathcal{T}) \cup \mathsf{sig}(\mathbf{q})) \to \{0, 1\}$, we let

$$\mathcal{A}_\gamma = \{A(a) \mid \gamma(A) = 1\} \cup \{R(a, a) \mid \gamma(R) = 1\}$$

and set $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}(\gamma) = 1$ iff $\mathcal{T}, \mathcal{A}_\gamma \models \mathbf{q}(\mathbf{a})$, where $\mathbf{a}$ is a tuple of a's of the required length. We call $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ the *primitive evaluation function* for $\mathbf{q}$ and $\mathcal{T}$.

Theorem 6 (implicit in [13])**.** *If $\mathbf{q}'$ is a (PE-) FO-rewriting of $\mathbf{q}$ and $\mathcal{T}$, then there is a (monotone) Boolean formula χ of size $O(|\mathbf{q}'|)$ that computes $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$.*

If (Π, G) is an NDL-rewriting of $\mathbf{q}$ and $\mathcal{T}$, then $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ is computed by a monotone Boolean circuit $\boldsymbol{C}$ of size $O(|\Pi|)$.

III. Succinctness Results for Query Rewriting

In this section, we relate the upper and lower bound functions from Section II-B to non-uniform models of computation, which allows us to exploit results from circuit complexity to infer bounds on rewriting size. As in [13], we use hypergraph programs (defined next) as a useful intermediate formalism.

A. Tree Hypergraph Programs (THGPs)

A hypergraph takes the form $H = (V, E)$, where V is a set of *vertices* and $E \subseteq 2^V$ a set of *hyperedges*. A subset $E' \subseteq E$ is *independent* if $e \cap e' = \emptyset$, for any distinct $e, e' \in E'$.

A *hypergraph program (HGP)* P consists of a hypergraph $H_P = (V_P, E_P)$ and a function $\mathfrak{l}_P$ that labels every vertex with 0, 1, or a conjunction of literals built from a set L_P of propositional variables. An input for P is a valuation of L_P. The HGP P computes the Boolean function f_P defined as follows: $f_P(\alpha) = 1$ iff there is an independent subset of E that *covers all zeros*—that is, contains every vertex in V whose label evaluates to 0 under α. A HGP is *monotone* if there are no negated variables among its vertex labels. The *size* $|P|$ of HGP P is $|V_P| + |E_P| + |L_P|$.

In what follows, we will focus on a particular subclass of HGPs whose hyperedges correspond to subtrees of a tree. Formally, given a tree $T = (V_T, E_T)$ and $u, v \in V_T$, the *interval* $\langle u, v \rangle$ is the set of edges that appear on the unique simple path connecting u and v. If $v_1, \dots, v_k \in V_T$, then the *generalized interval* $\langle v_1, \dots, v_k \rangle$ is defined as the union of intervals $\langle v_i, v_j \rangle$ over all pairs (i, j). We call $v \in V_T$ a *boundary vertex* for generalized interval I (w.r.t. T) if there exist edges $\{v, u\} \in E_T \cap I$ and $\{v, u'\} \in E_T \setminus I$. A hypergraph $H = (V_H, E_H)$ is a *tree hypergraph*[1] if there is a tree $T = (V_T, E_T)$ such that $V_H = E_T$ and every hyperedge in E_H is a *generalized interval of T all of whose boundary vertices have degree 2 in T*. A *tree hypergraph program (THGP)* is a HGP based on a tree hypergraph. As a special case, we have *interval hypergraphs* [26], [27] and *interval HGPs*, whose underlying trees have exactly 2 leaves.

B. Primitive Evaluation Function and THGPs

Our first step will be to show how functions given by THGPs can be computed using primitive evaluation functions.

Consider a THGP $P = (H_P, \mathfrak{l}_P)$ whose underlying tree T has vertices $v_1, \dots, v_n$, and let $T^{\downarrow}$ be the directed tree obtained from T by fixing its leaf v_1 as the root and orienting edges away from v_1. We wish to construct a tree-shaped CQ $\mathbf{q}_P$ and ontology $\mathcal{T}_P$ of depth 2 whose primitive evaluation function $f^{\mathsf{prim}}_{\mathbf{q}_P,\mathcal{T}_P}$ can be used to compute f_P. The query $\mathbf{q}_P$ is obtained by simply 'doubling' the edges in $T^{\downarrow}$:

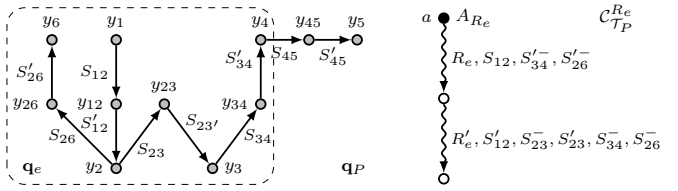
$$\mathbf{q}_P = \exists \mathbf{y} \bigwedge_{(v_i, v_j) \in T^{\downarrow}} (S_{ij}(y_i, y_{ij}) \wedge S'_{ij}(y_{ij}, y_j)).$$

The ontology $\mathcal{T}_P$ is defined as the union of $\mathcal{T}_e$ over all hyperedges $e \in E_P$. Let $e = \langle v_{i_1}, \dots, v_{i_m} \rangle \in E_P$ with v_{i_1} the vertex in e that is highest in $T^{\downarrow}$, and suppose w.l.o.g. that every v_{i_j} is either a boundary vertex of e or a leaf in T. Then $\mathcal{T}_e$ is defined as follows:

$$\begin{aligned}
&\{B_e(x) \to \exists y R_e(x, y), \exists y R_e(y, x) \to \exists y R'_e(x, y)\} \cup \\
&\{R_e(x, y) \to S_{i_1,k}(x, y) \mid \{v_{i_1}, v_k\} \in e\} \cup \\
&\{R_e(y, x) \to S'_{j_\ell, i_\ell}(x, y) \mid 1 < \ell \leq n, (v_{j_\ell}, v_{i_\ell}) \in T^{\downarrow}\} \cup \\
&\{R'_e(x, y) \to S'_{j,k}(x, y) \mid \{v_j, v_k\} \in e, (v_j, v_k) \in T^{\downarrow}, \\
&\qquad\qquad v_k \neq v_{i_\ell} \text{ for all } 1 < \ell \leq m\} \cup \\
&\{R'_e(x, y) \to S_{j,k}(y, x) \mid \{v_j, v_k\} \in e, (v_j, v_k) \in T^{\downarrow}, v_j \neq v_{i_1}\}.
\end{aligned}$$

Observe that both $\mathbf{q}_P$ and $\mathcal{T}_P$ are of polynomial size in $|P|$.

Example 7. *Consider a THGP P whose tree hypergraph has vertices $\{\{v_1, v_2\}, \{v_2, v_3\}, \{v_2, v_6\}, \{v_3, v_4\}, \{v_4, v_5\}\}$ and a single hyperedge $e = \langle v_1, v_4, v_6 \rangle$. Fixing v_1 as root and applying the above construction, we obtain the query $\mathbf{q}_P$ and the canonical model $\mathcal{C}^{R_e}_{\mathcal{T}_P}$ pictured below:*



[1]Our definition of tree hypergraph is a minor variant of the notion of (sub)tree hypergraph (aka hypertree) from graph theory, cf. [25]–[27].

Observe how the axioms in $\mathcal{T}_p$ ensure that the subquery $\mathbf{q}_e$ induced by hyperedge e maps into $\mathcal{C}_{\mathcal{T}_P}^{R_e}$.

The next theorem specifies how f_P can be computed using the primitive evaluation function $f^{\mathsf{prim}}_{\mathbf{q}_P,\mathcal{T}_P}$.

Theorem 8. *Let $P = (H_P, \mathfrak{l}_P)$ be a THGP. For every input α for P, $f_P(\alpha) = 1$ iff $f^{\mathsf{prim}}_{\mathbf{q}_P,\mathcal{T}_P}(\gamma) = 1$, where γ is defined as follows: $\gamma(B_e) = 1$, $\gamma(R_e) = \gamma(R'_e) = 0$, and $\gamma(S_{ij}) = \gamma(S'_{ij}) = \alpha(\mathfrak{l}_P(\{v_i, v_j\}))$.*

C. Bounded Treewidth Queries, THGPs, and SAC^1

We next show how the modified tree witness functions associated with bounded treewidth queries and bounded depth ontologies can be computed using THGPs. We then relate THGPs to the non-uniform complexity class SAC^1.

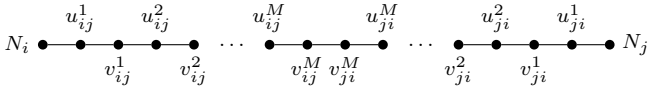
Suppose we are given a TBox $\mathcal{T}$ of depth d, a CQ $\mathbf{q}$, and a tree decomposition (T, λ) of $G_\mathbf{q}$ of width t, and we wish to define a THGP that computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$. In order to more easily refer to the variables in $\lambda(N)$, we construct functions $\lambda_1, \dots, \lambda_t$ such that $\lambda_i(N) \in \lambda(N)$ and $\lambda(N) = \cup_i \lambda_i(N)$.

The basic idea underlying the construction of the THGP is as follows: for each node N in T, we select a data-independent description of the way the variables in $\lambda(N)$ are homomorphically mapped into the canonical model. These descriptions are given by tuples from $W_d^t = \{(w_1, \dots, w_t) \mid w_i \in (\mathsf{N}_2^\pm \cap \mathsf{sig}(\mathcal{T}))^*, |w_i| \leq d\}$, where the ith word $\mathbf{w}[i]$ of tuple $\mathbf{w} \in W_d^t$ indicates that variable $\lambda_i(N)$ is mapped to an element of the form $a\,\mathbf{w}[i]$. The tuple assigned to node N must be compatible with the restriction of $\mathbf{q}$ to $\lambda(N)$ and with the tuples of neighbouring nodes. Formally, we say $\mathbf{w} \in W_d^t$ is *compatible with node N* if the following conditions hold:

- if $A(\lambda_i(N)) \in \mathbf{q}$ and $\mathbf{w}[i] \neq \varepsilon$, then $\mathbf{w}[i] = w'\varrho$ for some $\varrho \in \mathsf{N}_2^\pm$ with $\mathcal{T} \models \varrho(y, x) \to A(x)$
- if $R(\lambda_i(N), \lambda_j(N)) \in \mathbf{q}$, then one of the following holds:
  - $\mathbf{w}[i] = \mathbf{w}[j] = \varepsilon$
  - $\mathbf{w}[j] = \mathbf{w}[i] \cdot \varrho$ with $\mathcal{T} \models \varrho(x, y) \to R(x, y)$
  - $\mathbf{w}[i] = \mathbf{w}[j] \cdot \varrho$ with $\mathcal{T} \models \varrho(x, y) \to R(y, x)$

A pair $(\mathbf{w}, \mathbf{w}')$ is *compatible with the pair of nodes* (N, N') if $\lambda_i(N) = \lambda_j(N')$ implies that $\mathbf{w}[i] = \mathbf{w}'[j]$.

Suppose $W_d^t = \{\mathbf{w}_1, \dots, \mathbf{w}_M\}$, and consider the tree T' obtained from T by replacing edge $\{N_i, N_j\}$ by the edges:



The desired THGP $(H_{\mathbf{q},\mathcal{T}}, \mathfrak{l}_{\mathbf{q},\mathcal{T}})$ is based upon T' and contains the following hyperedges:

- $E_i^k = \langle u^k_{ij_1}, \dots, u^k_{ij_n} \rangle$, if $\mathbf{w}_k \in W_d^t$ is compatible with N_i and $N_{j_1}, \dots, N_{j_n}$ are the neighbours of N_i;
- $E_{ij}^{km} = \langle v^k_{ij}, v^m_{ji} \rangle$, if $\{N_i, N_j\}$ is an edge in T and $(\mathbf{w}_k, \mathbf{w}_m)$ is compatible with (N_i, N_j).

Intuitively, E_i^k corresponds to assigning (compatible) tuple $\mathbf{w}_k$ to node N_i, and hyperedges of the form E_{ij}^{km} are used to ensure compatibility of choices at neighbouring nodes. Vertices of $H_{\mathbf{q},\mathcal{T}}$ (i.e. the edges in T') are labeled by $\mathfrak{l}_{\mathbf{q},\mathcal{T}}$ as follows: edges of the forms $\{N_i, u^1_{ij}\}$, $\{v^\ell_{ij}, u^{\ell+1}_{ij}\}$, and $\{v^M_{ij}, v^M_{ji}\}$ are labelled 0, and every edge $\{u^\ell_{ij}, v^\ell_{ij}\}$ is labelled by the conjunction of the following variables:

- p_η, if $\eta \in \mathbf{q}$, $\mathsf{vars}(\eta) \subseteq \lambda(N_i)$, and $\lambda_g(N_i) \in \mathsf{vars}(\eta)$ implies $\mathbf{w}_\ell[g] = \varepsilon$;
- p^ϱ_z, if $\mathsf{vars}(\eta) = \{z\}$, $z = \lambda_g(N_i)$, and $\mathbf{w}_\ell[g] = \varrho w'$;
- p^ϱ_z, $p^\varrho_{z'}$, and $p_{z=z'}$, if $\mathsf{vars}(\eta) = \{z, z'\}$, $z = \lambda_g(N_i)$, $z' = \lambda_{g'}(N_i)$, and either $\mathbf{w}_\ell[g] = \varrho w'$ or $\mathbf{w}_\ell[g'] = \varrho w'$.

We prove in the appendix that the THGP $(H_{\mathbf{q},\mathcal{T}}, \mathfrak{l}_{\mathbf{q},\mathcal{T}})$ computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$, which allows us to establish the following result:

Theorem 9. *Fix $t \geq 1$ and $d \geq 0$. For every ontology $\mathcal{T}$ of depth $\leq d$ and CQ $\mathbf{q}$ of treewidth $\leq t$, there is a monotone THGP that computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ of size polynomial in $|\mathcal{T}| + |\mathbf{q}|$.*

To characterize the power of tree hypergraph programs, we consider *semi-unbounded fan-in* circuits in which NOT gates are applied only to the inputs, AND gates have fan-in 2, and OR gates have unbounded fan-in. The complexity class SAC^1 [28] is defined by considering circuits of this type having polynomial size and logarithmic depth. SAC^1 is the non-uniform analog of the class LOGCFL [19], which will play a central role in our complexity analysis in Section IV.

We consider semi-unbounded fan-in circuits of size σ and depth $\log \sigma$, where σ is a parameter, and show that they are polynomially equivalent to THGPs by providing reductions in both directions (details can be found in the appendix).

Theorem 10. *There exist polynomials p, p' such that:*

- *Every function computed by a semi-unbounded fan-in circuit of size at most σ and depth at most $\log \sigma$ is computable by a THGP of size $p(\sigma)$.*
- *Every function computed by a THGP of size σ is computable by a semi-unbounded fan-in circuit of size at most $p'(\sigma)$ and depth at most $\log p'(\sigma)$.*

Both reductions preserve monotonicity.

D. Bounded Leaf Queries, Linear THGPs, & NBPs

For bounded leaf queries, we establish a tight connection to *non-deterministic branching programs* (NBPs), a well-known representation of Boolean functions situated between Boolean formulas and Boolean circuits [21], [29]. We recall that an NBP is defined as a tuple $P = (V_P, E_P, s, t, \mathfrak{l}_P)$, where (V_P, E_P) is a directed graph, $s, t \in V$, and $\mathfrak{l}_P$ is a function that labels every edge $e \in E_P$ with 0, 1, or a conjunction of propositional literals built from L_P. The NBP P induces the function f_P defined as follows: for every valuation α of the variables L_P, $f_P(\alpha) = 1$ iff there is a path from s to t in the graph (V_P, E_P) such that all labels along the path evaluate to 1 under α. The size $|P|$ of P is $|V_P| + |E_P| + |L_P|$. An NBP is *monotone* if neither of its labels contains negation.

The next theorem shows that tree witness functions of bounded leaf queries can be captured by polysize NBPs.

Theorem 11. *Fix $\ell \geq 2$. For every ontology $\mathcal{T}$ and CQ $\mathbf{q}$ with at most ℓ leaves, the function $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ is computable by a monotone NBP of size polynomial in $|\mathbf{q}|$ and $|\mathcal{T}|$.*

Proof. Consider an ontology $\mathcal{T}$, a tree-shaped CQ $\mathbf{q}$ with ℓ leaves, and its associated graph $G_\mathbf{q} = (V_\mathbf{q}, E_\mathbf{q})$. For every tree

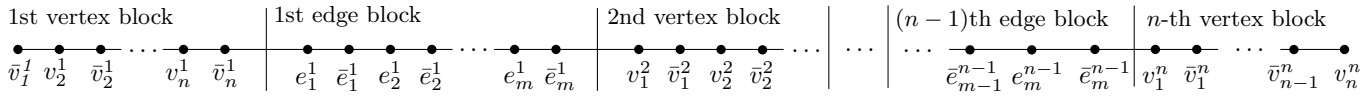


Fig. 3. The graph G underlying the interval hypergraph program from the proof of Theorem 12.

witness $\mathfrak{t} \in \Theta_{\mathcal{T}}^{\mathbf{q}}$, let $(V_\mathfrak{t}, E_\mathfrak{t})$ be the graph associated with $\mathbf{q}_\mathfrak{t}$, and for every subset $\Theta \subseteq \Theta_{\mathcal{T}}^{\mathbf{q}}$, let $V_\Theta = \bigcup_{\mathfrak{t}\in\Theta} V_\mathfrak{t}$ and $E_\Theta = \bigcup_{\mathfrak{t}\in\Theta} E_\mathfrak{t}$. Pick some vertex $v_0 \in V_{\mathbf{q}}$ and call an independent subset $\Theta \subseteq \Theta_{\mathcal{T}}^{\mathbf{q}}$ *flat* if every simple path in $G_{\mathbf{q}}$ with endpoint v_0 intersects at most one of the sets $E_\mathfrak{t}$, $\mathfrak{t} \in \Theta$. Note that every flat subset of $\Theta_{\mathcal{T}}^{\mathbf{q}}$ can contain at most ℓ tree witnesses, so the number of flat subsets is polynomially bounded in $|\mathbf{q}|$, when ℓ is a fixed constant. Flat subsets can be partially ordered as follows: $\Theta \prec \Theta'$ if every simple path between v_0 and a vertex $v' \in V_{\Theta'}$ intersects E_Θ.

The required NBP P is based upon the graph G_P with vertices $V_P = \{u_\Theta, v_\Theta \mid \Theta \subseteq \Theta_{\mathcal{T}}^{\mathbf{q}} \text{ is flat}\} \cup \{s, t\}$ and $E_P = \{(s, u_\Theta), (v_\Theta, t), (u_\Theta, v_\Theta) \mid \Theta \text{ flat}\} \cup \{(v_\Theta, u_{\Theta'}) \mid \text{flat } \Theta \prec \Theta'\}$. We label (u_Θ, v_Θ) with $\bigwedge_{\mathfrak{t}\in\Theta} p_\mathfrak{t}$ and label edges (s, u_Θ), (v_Θ, t), and (v_Θ, u_Θ) by conjunctions of variables p_η ($\eta \in \mathbf{q}$) corresponding respectively to the atoms in $\mathbf{q}$ that occur 'before' Θ, 'after' Θ, and 'between' Θ and Θ'. In the appendix, we detail the construction and show that $f_{\mathbf{q},\mathcal{T}}^{\mathsf{tw}}(\alpha) = 1$ iff there is a path from s to t in G_P all of whose labels evaluate to 1 under α. □

NBPs in turn can be translated into polysize interval HGPs.

Theorem 12. *Every function that is computed by a NBP P is computed by a interval HGP whose size is polynomial in $|P|$. The reduction preserves monotonicity.*

Proof. Consider an NBP $P = (V_P, E_P, v_1, v_n, \mathfrak{l}_P)$, where $V_P = \{v_1, \dots, v_n\}$ and $E_P = \{e_1, \dots, e_m\}$. We may assume w.l.o.g. that $e_m = (v_n, v_n)$ and $\mathfrak{l}_P(e_m) = 1$. This assumption ensures that if there is a path from v_1 to v_n whose labels evaluate to 1, then there is a (possibly non-simple) path with the same properties whose length is exactly $n-1$.

We now construct an interval HGP $(H, \mathfrak{l}_H)$ that computes the function f_P. In Figure 3, we display the graph $G = (V_G, E_G)$ that underlies the interval hypergraph H. Its vertices are arranged into n *vertex blocks* and $n-1$ *edge blocks* which alternate. The ℓth vertex block (resp. edge block) contains two copies, $v_i^\ell, \bar{v}_i^\ell$ (resp. $e_i^\ell, \bar{e}_i^\ell$), of every vertex $v_i \in V_P$ (resp. edge $e_i \in E_P$). We remove the first and last vertices v_1^1 and $\bar{v}_n^n$ and connect the remaining vertices as shown in Figure 3. The hypergraph $H = (V_H, E_H)$ is defined by setting $V_H = E_G$ and letting E_H be the set of all hyperedges $\zeta_{i,\ell} = \langle \bar{v}_j^\ell, e_i^\ell \rangle$ and $\zeta'_{i,\ell} = \langle \bar{e}_i^\ell, v_k^{\ell+1} \rangle$ where $e_i = (v_j, v_k) \in E_P$ and $1 \le \ell < n$. The function $\mathfrak{l}_H$ labels $\{e_i^\ell, \bar{e}_i^\ell\}$ with $\mathfrak{l}_P(e_i)$ and all other vertices of H (i.e. edges of G) with 0.

We claim that $(H, \mathfrak{l}_H)$ computes f_P. Indeed, if $f_P(\alpha) = 1$, then there is a path $e_{j_1}, e_{j_2} \dots, e_{j_{n-1}}$ from v_1 to v_n whose labels evaluate to 1 under α. It follows that $E' = \{\zeta_{j_\ell,\ell}, \zeta'_{j_\ell,\ell} \mid 1 \le \ell < n\}$ is an independent subset of E_H that covers all zeros. Conversely, if $E' \subseteq E_H$ is independent and covers all zeros under α, then it must contain exactly one pair of hyperedges $\zeta_{j_\ell,\ell}$ and $\zeta'_{j_\ell,\ell}$ for every $1 \le \ell < n$, and the corresponding sequence of edges $e_{j_1}, \dots, e_{j_{n-1}}$ defines a path from v_1 to v_n. Moreover, since E' does not cover $\{e_{j_\ell}^\ell, \bar{e}_{j_\ell}^\ell\}$, we know that $\mathfrak{l}_H(\{e_{j_\ell}^\ell, \bar{e}_{j_\ell}^\ell\}) = \mathfrak{l}_P(e_{j_\ell})$ evaluates to 1 under α, for every $1 \le \ell < n$. □

E. Succinctness Results

We now combine the correspondences from the preceding subsections with results from circuit complexity to derive upper and lower bounds on rewriting size for tree-like queries.

We start with what is probably our most surprising result: a super-polynomial lower bound on the size of PE-rewritings of linear queries and depth-2 ontologies. This result significantly improves upon earlier negative results for PE-rewritings [12], [13], which required either arbitrary queries or arbitrary ontologies. The proof utilizes Theorems 6, 8 and 12 and the well-known circuit complexity result that there is a sequence f_n of monotone Boolean functions that are computable by polynomial-size monotone NBPs, but all monotone Boolean formulas computing f_n are of size $n^{\Omega(\log n)}$ [30].

Theorem 13. *There is a sequence of linear CQs $\mathbf{q}_n$ and ontologies $\mathcal{T}_n$ of depth 2, both of polysize in n, such that any PE-rewriting of $\mathbf{q}_n$ and $\mathcal{T}_n$ is of size $n^{\Omega(\log n)}$.*

We obtain a positive result for NDL-rewritings of bounded-leaf queries using Theorems 4 and 11 and the fact that NBPs are representable as polynomial-size monotone circuits [29].

Theorem 14. *Fix a constant $\ell > 1$. Then all tree-shaped CQs with at most ℓ leaves and arbitrary ontologies have polynomial-size NDL-rewritings.*

As for FO-rewritings, we can use Theorems 4, 6, 12, and 8 to show that the existence of polysize FO-rewritings is equivalent to the open problem of whether $\mathrm{NL/poly} \subseteq \mathrm{NC}^1$.

Theorem 15. *The following are equivalent:*

1) *There exist polysize FO-rewritings for all linear CQs and depth 2 ontologies;*
2) *There exist polysize FO-rewritings for all tree-shaped CQs with at most ℓ leaves and arbitrary ontologies (for any fixed ℓ);*
3) *There exists a polynomial function p such that every NBP of size at most s is computable by a formula of size $p(s)$. Equivalently,* $\mathrm{NL/poly} \subseteq \mathrm{NC}^1$.

Turning next to bounded treewidth queries and bounded depth ontologies, Theorems 9 and 10 together provide a means of constructing a polysize monotone SAC^1 circuit that computes $f_{\mathbf{q},\mathcal{T}}^{\mathsf{tw}'}$. Applying Theorem 5, we obtain:

Theorem 16. *Fix $t > 0$ and $d > 0$. Then all CQs of treewidth $\le t$ and ontologies of depth $\le d$ have polysize NDL-rewritings.*

In the case of FO-rewritings, we can show that the existence of polysize rewritings corresponds to the open question of whether $\mathrm{SAC}^1 \subseteq \mathrm{NC}^1$.

Theorem 17. *The following are equivalent:*

1) *There exist polysize FO-rewritings for all tree-shaped CQs and depth 2 ontologies;*
2) *There exist polysize FO-rewritings for all CQs of treewidth at most t and ontologies of depth at most d (for fixed constants $t > 0$ and $d > 0$);*
3) *There exists a polynomial function p such that every semi-unbounded fan-in circuit of size at most σ and depth at most $\log \sigma$ is computable by a formula of size $p(\sigma)$. Equivalently, $\mathrm{SAC}^1 \subseteq \mathrm{NC}^1$.*

To complete the succinctness landscape, we generalize the result of [13] that says that all tree-shaped queries and depth 1 ontologies have polysize PE-rewritings by showing that this is also true for the wider class of bounded treewidth queries.

Theorem 18. *Fix $t > 0$. Then there exist polysize PE-rewritings for all CQs of treewidth $\leq t$ and depth 1 ontologies.*

IV. Complexity Results for Query Answering

To complement our succinctness results and to gain a better understanding of the inherent difficulty of query answering, we analyze the computational complexity of answering tree-like queries in the presence of OWL 2 QL ontologies.

A. Bounded Depth Ontologies

We begin by showing that the LOGCFL upper bound for bounded treewidth queries from [18] remains applicable in the presence of ontologies of bounded depth.

Theorem 19. *CQ answering is in* LOGCFL *for bounded treewidth queries and bounded depth ontologies.*

Proof. By Equation 1, $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{a})$ just in the case that $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \mathbf{q}(\mathbf{a})$. When $\mathcal{T}$ has finite depth, $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is a finite relational structure, so the latter problem is nothing other than standard conjunctive query evaluation over databases, which is LOGCFL-complete when restricted to bounded treewidth queries [14]. As LOGCFL is closed under $\mathrm{L}^{\mathrm{LOGCFL}}$ reductions [18], it suffices to show that $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ can be computed by means of an $\mathrm{L}^{\mathrm{LOGCFL}}$-transducer (that is, a deterministic logspace Turing machine with access to an LOGCFL oracle).

We briefly describe the $\mathrm{L}^{\mathrm{LOGCFL}}$-transducer that generates $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ when given a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ whose ontology $\mathcal{T}$ has depth at most k. First note that we need only logarithmically many bits to represent a predicate name or individual constant from $\mathcal{K}$. Moreover, as $\mathcal{T}$ has depth at most k, the domain of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is contained in the set $U = \{aw \mid aw \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}, |w| \leq k\}$. Since k is a fixed constant, each element in U can be stored using logarithmic space in $|\mathcal{K}|$. Finally, we observe that each of the following operations can be performed by making a call to an NL (hence LOGCFL) oracle:

- Decide whether $aw \in U$ belongs to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Decide whether $u \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ belongs to $A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Decide whether $(u, u') \in (\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}})^2$ belongs to $r^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

Indeed, all three problems can be decided using constantly many entailment checks, and entailment is in NL [1]. □

Procedure TreeQuery

Input: KB $(\mathcal{T}, \mathcal{A})$, tree-shaped query $\mathbf{q}$ with $\mathsf{avars}(\mathbf{q}) = (z_1, \ldots, z_n)$, tuple $\mathbf{b} = (b_1, \ldots, b_n) \in \mathsf{inds}(\mathcal{A})^n$

1: Fix a directed tree T compatible with $G_{\mathbf{q}}$. Let v_0 be the root variable. Set $U = \{aw \mid |w| \leq 2|\mathcal{T}| + |\mathbf{q}|\}$.
2: Guess $u_0 \in U$ and return **no** if either:
- $u_0 \in \mathsf{inds}(\mathcal{A})$ and MapCore$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_0, u_0) =$ **false**
- $u_0 = a_0 w_0 R$ and MapAnon$(\mathcal{T}, \mathbf{q}, v_0, R) =$ **false**

3: Initialize Frontier to $\{(v_0, u_0)\}$.
4: While Frontier $\neq \emptyset$
a: Remove (v_1, u_1) from Frontier.
b: For every child v_2 of v_1
i) Guess $u_2 \in U$. Return **no** if one of the following holds:
- $\mathbf{q}$ contains $P(v_1, v_2)$ ($P \in \mathsf{N}_2^{\pm}$), $\mathcal{C}_{\mathcal{T},\mathcal{A}} \not\models P(u_1, u_2)$
- $u_2 \in \mathsf{inds}(\mathcal{A})$ and MapCore$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_2, u_2) =$**false**
- $u_2 = a_2 w_2 R$ and MapAnon$(\mathcal{T}, \mathbf{q}, v_2, R) =$ **false**

ii) Add (v_2, u_2) to Frontier.
5: Return **yes**.

MapCore$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v, u)$

Return **false** iff one of the following holds:

- $v = z_i$ and $u \neq b_i$ for some $1 \leq i \leq n$
- $\mathbf{q}$ contains $A(v)$ and $\mathcal{T}, \mathcal{A} \not\models A(u)$
- $\mathbf{q}$ contains $P(v, v)$ and $\mathcal{T}, \mathcal{A} \not\models P(u, u)$.

MapAnon$(\mathcal{T}, \mathbf{q}, v, R)$

Return **false** iff one of the following holds:

- $v \in \mathsf{avars}(\mathbf{q})$
- $\mathbf{q}$ contains $A(v)$ and $\mathcal{T} \not\models \exists y R(y, x) \rightarrow A(x)$
- $\mathbf{q}$ contains some atom of the form $S(v, v)$.

Fig. 4. Non-deterministic procedure for answering tree-shaped queries

If we restrict the number of leaves in tree-shaped queries, then we can improve the preceding upper bound to NL:

Theorem 20. *CQ answering is* NL*-complete for bounded leaf queries and bounded depth ontologies.*

Proof. The lower bound is an immediate consequence of the NL-hardness of answering atomic queries in OWL 2 QL [1].

For the upper bound, we introduce in Figure 4 a non-deterministic procedure TreeQuery for deciding whether a tuple is a certain answer of a tree-shaped query. The procedure views the input query as a directed tree and constructs a homomorphism on-the-fly by traversing the tree from root to leaves. The set Frontier is initialized with a single pair (v_0, u_0), which represents the choice of where to map the root variable v_0. The possible choices for u_0 include all individuals from $\mathcal{A}$ as well as all elements aw that belong to the domain of the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ and have $|w| \leq 2|\mathcal{T}| + |q|$. The latter bound is justified by the well-known fact that if there

is a homomorphism of $\mathbf{q}$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, then there is one whose image only involves elements aw with $|w| \leq 2|\mathcal{T}| + |q|$. We use the sub-procedures MapCore or MapAnon to check that the guessed element u_0 is compatible with the variable v_0. If $u_0 \in \mathsf{inds}(\mathcal{A})$, then we use the first sub-procedure MapCore, which verifies that (i) if v_0 is an answer variable, then u_0 is the individual corresponding to v_0 in the tuple $\mathbf{b}$, and (ii) u_0 satisfies all atoms in $\mathbf{q}$ that involve only v_0. If $u_0 \notin \mathsf{inds}(\mathcal{A})$, then u_0 must take the form $a_0 w_0 R$. In this case, MapAnon is called and checks that v_0 is not an answer variable, $\mathbf{q}$ does not contain a reflexive loop at v_0, and $\mathcal{T} \models \exists y R(y,x) \rightarrow A(x)$ (equivalently, $a_0 w_0 R \in A^{\mathcal{C}}_{\mathcal{T},\mathcal{A}}$) for every $A(v_0) \in \mathbf{q}$. The remainder of the procedure consists of a while loop, in which we remove a pair (v_1, u_1) from Frontier, and if v_1 is not a leaf node, we guess where to map the children of v_1. We must then check that the guessed element u_2 for child v_2 is compatible with the role assertions linking v_1 to v_2 and the unary atoms concerning v_2 (using MapCore or MapAnon described earlier). If some check fails, we return **no**, and otherwise we add (v_2, u_2) to Frontier, for each child v_2 of v_1. We exit the while loop when Frontier is empty, i.e. when an element of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ has been assigned to every variable in $\mathbf{q}$.

Correctness and termination are straightforward to show and hold for arbitrary tree-shaped queries and OWL 2 QL ontologies. Membership in NL for bounded depth ontologies and bounded leaf queries relies upon the following observations:

- if $\mathcal{T}$ has depth k and $aw \in U$, then $|w| \leq k$
- if $\mathbf{q}$ has ℓ leaves, then |Frontier| never exceeds ℓ

which ensure Frontier can be stored in logarithmic space. □

B. Bounded Leaf Queries & Arbitrary Ontologies

The only remaining case is that of bounded leaf queries and arbitrary ontologies, for which neither the upper bounds from the preceding subsection, nor the NP lower bound from [12] can be straightforwardly adapted. We settle the question by showing LOGCFL-completeness.

Theorem 21. *CQ answering is* LOGCFL*-complete for bounded leaf queries and arbitrary ontologies. The lower bound holds already for linear queries.*

1) LOGCFL *upper bound:* The upper bound relies on a characterization of the class LOGCFL in terms of non-deterministic auxiliary pushdown automata (NAuxPDAs). We recall that an NAuxPDA [31] is a non-deterministic Turing machine that has an additional work tape that is constrained to operate as a pushdown store. Sudborough [32] proved that LOGCFL can be characterized as the class of problems that can be solved by NAuxPDAs that run in logarithmic space and in polynomial time (note that the space on the pushdown tape is not subject to the logarithmic space bound). Thus, to show membership in LOGCFL, it suffices to define a procedure for answering bounded leaf queries that can be implemented by such an NAuxPDA. We present such a procedure in Figure 5. The input query is assumed to be connected; this is w.l.o.g. since the connected components can be treated separately.

We start by giving an overview of the procedure BLQuery. Like TreeQuery, the idea is to view the input query $\mathbf{q}$ as a tree and iteratively construct a homomorphism of the query into the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, working from root to leaves. At the start of the procedure, we guess an element $a_0 w_0$ to which the root variable v_0 is mapped and check that the guessed element is compatible with v_0. However, instead of storing directly $a_0 w_0$ on Frontier, we push the word w_0 onto the stack (Stack) and record the height of the stack ($|w_0|$) in Height. We then initialize Frontier to the set of all 4-tuples $(v_0, v_i, a_0, \mathsf{Height})$ with v_i a child of v_0. Intuitively, a tuple (v, v', c, n) records that the variable v is mapped to the element $c\,\mathsf{Stack}[n]$ and that the child v' of v remains to be mapped (we use $\mathsf{Stack}[m]$ to denote the word consisting of the first m symbols of Stack).

In Step 4, we will remove one or more tuples from Frontier, choose where to map the variable(s) in the second component, and update Frontier, Stack, and Height accordingly. There are three options depending on how we map the variable. Option 1 will be used for tuples $(v, v', c, 0)$ in which both v and v' are mapped to named constants, while Option 2 (resp. Option 3) is used for tuples (v, v', c, n) in which we wish to map v' to a child (resp. parent) of v. Crucially, however, the order in which tuples are treated matters, due to the fact that several tuples are 'sharing' the single stack. Indeed, when applying Option 3, we pop a symbol from Stack, and may therefore lose some information that is needed for the processing of other tuples. To prevent this, Option 3 may only be applied to tuples whose last component is maximal (i.e. equals Height), and it must be applied to *all* such tuples. For Option 2, we will also impose that the selected tuple (v, v', c, n) is such that $n =$ Height. This is needed because Option 2 corresponds to mapping v' to an element $c\,\mathsf{Stack}[n]\,S$, and we need to access the nth symbol in Stack to determine the possible choices for S and to record the symbol chosen (by pushing it onto Stack).

The procedure terminates and returns **yes** when Frontier is empty, meaning that we have successfully constructed a homomorphism of the input query into the canonical model that witnesses that the input tuple is an answer. Conversely, given such a homomorphism, we can define a successful execution of BLQuery, as illustrated by the following example.

Example 22. *Reconsider the KB $(\mathcal{T}_0, \mathcal{A}_0)$, CQ $\mathbf{q}_0$, and homomorphism $\mathbf{q}_0(c, a) \rightarrow \mathcal{C}_{\mathcal{T}_0,\mathcal{A}_0}$ from Figure 2. We show in what follows how h_0 can be used to define an execution of BLQuery that outputs* **yes** *on input $(\mathcal{T}, \mathcal{A}, \mathbf{q}, (c, a))$.*

In Step 1, we will fix some variable, say y_1, as root. Since we wish to map y_1 to aP, we will guess in Step 2 the constant a and word P and verify using MapAnon that our choice is compatible with y_1. As the check succeeds, we proceed to Step 3, where we initialize Stack *to P,* Height *to* 1*, and* Frontier *to $\{(y_1, y_2, a, 1), (y_1, y_3, a, 1)\}$. Here the tuple $(y_1, y_2, a, 1)$ records that y_1 has been mapped to $a\,\mathsf{Stack}[1] = aP$ and the edge between y_1 and y_2 remains to be mapped.*

At the beginning of Step 4, Frontier *contains* 2 *tuples: $(y_1, y_2, a, 1)$ and $(y_1, y_3, a, 1)$. Since y_1, y_2, and y_3 are mapped to aP, a, and aPS respectively, we will use Op-*

Procedure BLQuery

Input: KB $(\mathcal{T}, \mathcal{A})$, connected tree-shaped query $\mathbf{q}$ with $\mathsf{avars}(\mathbf{q}) = (z_1, \ldots, z_n)$, tuple $\mathbf{b} = (b_1, \ldots, b_n) \in \mathsf{inds}(\mathcal{A})^n$

1: Fix a directed tree T, with root v_0, compatible with $G_{\mathbf{q}}$.
2: Guess $a_0 \in \mathsf{inds}(\mathcal{A})$ and $w_0 \in (\mathsf{N}_2^{\pm})^*$ with $|w_0| \leq 2|\mathcal{T}| + |\mathbf{q}|$ and $a_0 w_0 \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. Return **no** if either:
- $w_0 = \varepsilon$ and MapCore$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_0, a)$= **false**;
- $w_0 = w_0' R$ and MapAnon$(\mathcal{T}, \mathbf{q}, v_0, R)$= **false**.

3: Initialize Stack to w_0, Height to $|w_0|$, and Frontier to $\{(v_0, v_i, a_0, \mathsf{Height}) \mid v_i \text{ is a child of } v_0\}$.
4: While Frontier $\neq \emptyset$, do one of the following:

Option 1 *// Take one step in the core*

a: Remove $(v_1, v_2, c, 0)$ from Frontier.
b: Guess $d \in \mathsf{inds}(\mathcal{A})$. Return **no** if either
- $\mathbf{q}$ contains $P(v_1, v_2)$ $(P \in \mathsf{N}_2^{\pm})$, $\mathcal{C}_{\mathcal{T},\mathcal{A}} \not\models P(u_1, u_2)$
- MapCore$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_2, d)$= **false**.

c: For every child v_3 of v_2, add $(v_2, v_3, d, 0)$ to Frontier.

Option 2 *// Take one step 'forward' in anonymous part*

d: If Height $= 2|\mathcal{T}| + |\mathbf{q}|$, return **no**. Otherwise, remove $(v_1, v_2, c, \mathsf{Height})$ from Frontier.
e: Guess $S \in \mathsf{N}_2^{\pm}$. Return **no** if one of the following holds:
- Height $= 0$ and $\mathcal{T}, \mathcal{A} \not\models \exists x\, S(c, x)$
- Height > 0 and $\mathcal{T} \not\models \exists y R(y, x) \to \exists y S(x, y)$, where R is the top symbol of Stack
- $\mathbf{q}$ contains $P(v_1, v_2)$ and $\mathcal{T} \not\models S(x, y) \to P(x, y)$
- MapAnon$(\mathcal{T}, \mathbf{q}, v_2, S)$= **false**

f: If v_2 has at least one child in T, then
- Push S onto Stack, and increment Height.
- For every child v_3 of v_2 in T, add $(v_2, v_3, c, \mathsf{Height})$ to Frontier.

Else, pop $\delta = \mathsf{Height} - \mathbf{max}\{\ell \mid (v, v', d, \ell) \in \mathsf{Frontier}\}$ symbols from Stack and decrement Height by δ.

Option 3 *// Take one step 'backward' in anonymous part*

g: If Height $= 0$, return **no**. Else, remove Deepest $= \{(v_1, v_2, c, n) \in \mathsf{Frontier} \mid n = \mathsf{Height}\}$ from Frontier, pop R from Stack, and decrement Height.
h: Return **no** if for some $(v_1, v_2, c, n) \in$ Deepest, one of the following holds:
- Height $= 0$ and MapCore$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_2, c)$= **false**
- Height > 0 and MapAnon$(\mathcal{T}, \mathbf{q}, v_2, S)$= **false**, where S is the top symbol of Stack
- $\mathbf{q}$ contains $P(v_1, v_2)$ and $\mathcal{T} \not\models R(y, x) \to P(x, y)$

i: If there is some $(v_1, v_2, c, n) \in$ Deepest such that v_2 is a non-leaf node in T:
- For every $(v_1, v_2, c, n) \in$ Deepest and child v_3 of v_2 in T, add $(v_2, v_3, c, \mathsf{Height})$ to Frontier.

Else, pop $\delta = \mathsf{Height} - \mathbf{max}\{\ell \mid (v, v', d, \ell) \in \mathsf{Frontier}\}$ symbols from Stack and decrement Height by δ.

5: Return **yes**.

Fig. 5. Non-deterministic procedure for answering bounded leaf queries. Refer to Fig. 4 for the definitions of MapCore and MapAnon.

tion 3 ('step backward') for $(y_1, y_2, a, 1)$ and Option 2 ('step forward') for $(y_1, y_3, a, 1)$. If we were to apply Option 3 at this stage, then we would be forced to treat both tuples together, and the check in Step 4(h) would fail for $(y_1, y_3, a, 1)$ since $S(y_1, y_3) \in \mathbf{q}$ but $\mathcal{T} \not\models R(y, x) \to S(x, y)$. We will therefore choose to perform Option 2, removing $(y_1, y_3, a, 1)$ from Frontier *in Step 4(d) and guessing S in Step 4(e). As the check succeeds, we will proceed to 4(f), where we push S onto* Stack*, set* Height $= 2$*, and add tuples $(y_3, y_4, a, 2)$ and $(y_3, y_5, a, 2)$ to* Frontier*. Observe that from the tuples in* Frontier*, we can read off the elements aStack$[1]$ and aStack$[2]$ to which variables y_1 and y_3 are mapped.*

At the start of the second iteration of the while loop, we have Frontier $= \{(y_1, y_2, a, 1), (y_3, y_4, a, 2), (y_3, y_5, a, 2)\}$*,* Stack $= PS$*, and* Height $= 2$*. Note that since h_0 maps y_4 to aP and y_5 to $aPST^-$, we will use Option 3 to treat $(y_3, y_4, a, 2)$ and Option 2 for $(y_3, y_5, a, 2)$. It will again be necessary to start with Option 2. We will thus remove $(y_3, y_5, a, 2)$ from* Frontier*, and guess the relation T^- (which satisfies the required conditions). Since y_5 does not have any children and* Height $- \mathbf{max}\{\ell \mid (v, v', d, \ell) \in \mathsf{Frontier}\} = 2 - 2 = 0$*, we leave* Frontier*,* Stack*, and* Height *unchanged.*

At the start of the third iteration, we have Frontier $= \{(y_1, y_2, a, 1), (y_3, y_4, a, 2)\}$*,* Stack $= PS$*, and* Height $= 2$*. We have already mentioned that both tuples should be handled using Option 3. We will start by applying Option 3 to tuple $(y_3, y_4, a, 2)$ since its last component is maximal. We will thus remove $(y_3, y_4, a, 2)$ from* Frontier*, pop S from* Stack*, and decrement* Height*. As the checks succeed for S, we will add the tuple $(y_4, x_2, a, 1)$ to* Frontier *in Step 4(i).*

At the start of the fourth iteration, we have Frontier $= \{(y_1, y_2, a, 1), (y_4, x_2, a, 1)\}$*,* Stack $= P$*, and* Height $= 1$*. Since y_4 and x_2 are mapped respectively to aP and a, we should use Option 3 to handle the second tuple. We will thus apply Option 3 with* Deepest $= \{(y_1, y_2, a, 1), (y_4, x_2, a, 1)\}$*. This will lead to both tuples being removed from* Frontier*, P being popped from* Stack*, and* Height *being decremented. We next perform the required checks in Step 4(h), and in particular, we verify that the choice of where to map the answer variable x_2 agrees with the input vector $\mathbf{b}$ (which is indeed the case). In Step 4(i), we add $(y_2, x_1, a, 0)$ to* Frontier*.*

The final iteration of the while loop begins with Frontier $= \{(y_2, x_1, a, 0)\}$*,* Stack $= \epsilon$*, and* Height $= 0$*. Since h_0 maps x_1 to the constant c, we will choose Option 1. We thus remove $(y_2, x_1, a, 0)$ from* Frontier*, guess the constant c, and perform the required compatibility checks. As x_1 is a leaf, no new tuples are added to* Frontier*. We are thus left with* Frontier $= \emptyset$*, and so we continue on to Step 7, where we output* ***yes****.*

In the appendix, we argue that BLQuery can be implemented by an NAuxPDA, and we prove its correctness:

Proposition 23. *Every execution of BLQuery terminates.*

There exists an execution of BLQuery that returns **yes** *on input* $(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b})$ *just in the case that* $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{b})$.

2) LOGCFL *lower bound:* The proof is by reduction from the problem of deciding whether an input of length l is accepted by the lth circuit of a *logspace-uniform* family of SAC^1 circuits (proven LOGCFL-hard in [33]). This problem was used in [14] to establish the LOGCFL-hardness of evaluating tree-shaped queries over databases. We follow a similar approach, but with one crucial difference: the power of OWL 2 QL ontologies allows us to 'unravel' the circuit into a tree and to use linear queries instead of tree-shaped ones.

As in [14], we assume w.l.o.g. that the considered SAC^1 circuits adhere to the following *normal form*:

- fan-in of all AND gates is 2;
- nodes are assigned to levels, with gates on level i only receiving inputs from gates on level $i+1$;
- there are an odd number of levels with the output AND gate on level 1;
- all even-level gates are OR gates, and all odd-level (excepting the circuit inputs) gates are AND gates.

It is well known (cf. [14], [19]) and easy to see that a circuit in normal form accepts an input $\mathbf{x}$ iff there is a labelled rooted tree (called a *proof tree*) with the following properties:

- the root node is labelled with the output AND gate;
- if a node is labelled by an AND gate g_i, then it has two children labelled by the gates of g_i two predecessor gates;
- if a node is labelled by an OR gate g_i, then it has a unique child that is labelled by a predecessor of g_i;
- every leaf node is labelled by an input gate whose corresponding literal evaluates into 1 under $\mathbf{x}$.

For example, the circuit $\boldsymbol{C}^*$ in Fig. 6(a) accepts input $\mathbf{x}^* = (1,0,0,0,1)$, as witnessed by the proof tree in Fig. 6(b).

Importantly, while a circuit-input pair may admit multiple proof trees, they are all isomorphic modulo the labelling. Thus, with every circuit $\boldsymbol{C}$, we can associate a *skeleton proof tree* $T_{\boldsymbol{C}}$ such that $\boldsymbol{C}$ accepts input $\mathbf{x}$ iff some labelling of $T_{\boldsymbol{C}}$ is a proof tree for $\boldsymbol{C}$ and $\mathbf{x}$. The reduction in [14] encodes the circuit $\boldsymbol{C}$ and input $\mathbf{x}$ in the database and uses a Boolean tree-shaped query based upon the skeleton proof tree. More precisely, the database $D^{\mathbf{x}}_{\boldsymbol{C}}$ uses the gates of $\boldsymbol{C}$ as constants and contains the following facts[2]:

- $U(g_j, g_i)$, for every OR gate g_i with predecessor gate g_j;
- $L(g_j, g_i)$ (resp. $R(g_j, g_i)$), for every AND gate g_i with left (resp. right) predecessor g_j;
- $A(g_i)$, for every input gate g_i whose value is 1 under $\mathbf{x}$.

The query $\mathbf{q}_{\boldsymbol{C}}$ uses the nodes of $T_{\boldsymbol{C}}$ as variables, has an atom $U(n_j, n_i)$ (resp. $L(n_j, n_i)$, $R(n_j, n_i)$) for every node n_i with unique (resp. left, right) child n_j, and has an atom $A(n_i)$ for every leaf node n_i. It is proven in [14] that $D^{\mathbf{x}}_{\boldsymbol{C}} \models \mathbf{q}_{\boldsymbol{C}}$ just in the case that $\boldsymbol{C}$ accepts $\mathbf{x}$. Moreover, both $\mathbf{q}_{\boldsymbol{C}}$ and $D^{\mathbf{x}}_{\boldsymbol{C}}$ can be constructed by means of logspace transducers.

To adapt the preceding reduction to our setting, we will replace the tree-shaped query $\mathbf{q}_{\boldsymbol{C}}$ by a linear query $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ that is obtained, intuitively, by performing an ordered depth-first traversal of $\mathbf{q}_{\boldsymbol{C}}$. The new query $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ may give a different answer than $\mathbf{q}_{\boldsymbol{C}}$ when evaluated on $D^{\mathbf{x}}_{\boldsymbol{C}}$, but the two queries coincide if evaluated on the *unraveling of* $D^{\mathbf{x}}_{\boldsymbol{C}}$ *into a tree*. Thus, we will define a KB $(\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}})$ whose canonical model induces a tree that is isomorphic to the tree-unravelling of $D^{\mathbf{x}}_{\boldsymbol{C}}$.

To formally define the query $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$, consider the sequence of words inductively defined as follows: $w_0 = \epsilon$ and $w_{j+1} = L^- U^- w_j U L R^- U^- w_j U R$. Every word $w = \varrho_1 \varrho_2 \dots \varrho_k$ naturally gives rise to a linear query $\mathbf{q}_w = \bigwedge_{i=1}^{k} \varrho_i(y_{i-1}, y_i)$. We then take

$$\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}} = \exists y_1 \dots \exists y_k (\mathbf{q}_{w_d} \wedge \bigwedge_{w_n[i,i+1]=U^- U} A(y_i)).$$

where $k = |w_d|$ and d is such that $\boldsymbol{C}$ has $2d+1$ levels. The query $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}^*}$ for our example circuit $\boldsymbol{C}^*$ is given in Fig. 6(c).

We now proceed to the definition of the KB $(\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}})$. Suppose $\boldsymbol{C}$ has gates $g_1, g_2, \dots, g_m$, with g_1 the output gate. In addition to the predicates U, L, R, A from earlier, we introduce a unary predicate G_i for each gate g_i and a binary predicate P_{ij} for each gate g_i with predecessor g_j We set $\mathcal{A}_{\boldsymbol{C}} = \{G_1(a)\}$ and include in $\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}$ the following axioms:

- $G_i(x) \rightarrow \exists y P_{ij}(y, x)$ and $\exists y P_{ij}(x, y) \rightarrow G_j(x)$ for every gate g_i with predecessor g_j;
- $P_{ij}(x, y) \rightarrow S(x, y)$ for every $S \in \{U, L, R\}$ such that $S(g_j, g_i) \in D^{\mathbf{x}}_{\boldsymbol{C}}$;
- $G_i(x) \rightarrow A(x)$ whenever $A(g_i) \in D^{\mathbf{x}}_{\boldsymbol{C}}$.

In Fig. 6(d), we display (a portion of) the canonical model of the KB associated with circuit $\boldsymbol{C}^*$ and input $\mathbf{x}^*$. Observe that, when restricted to the predicates U, L, R, A, it is isomorphic to the unravelling of $D^{\mathbf{x}}_{\boldsymbol{C}}$ into a tree starting from g_1.

In the appendix, we argue $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ and $(\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}})$ can be constructed by logspace transducers, and we prove the following proposition that establishes the correctness of the reduction.

Proposition 24. $\boldsymbol{C}$ *accepts input* $\mathbf{x}$ *iff* $\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}} \models \mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}(a)$.

V. Conclusion

In this paper, we have clarified the impact of query topology and ontology depth on the worst-case size of query rewritings and the complexity of query answering in OWL 2 QL. Our results close an open question from [13] and yield a complete picture of the succinctness and complexity landscapes for the considered classes of queries and ontologies.

On the theoretical side, our results demonstrate the utility of using non-uniform complexity as a tool for studying succinctness. In future work, we plan to utilize the developed machinery to investigate additional dimensions of the succinctness landscape, with the hope of identifying other natural restrictions on queries and ontologies that guarantee small rewritings. We speculate that our techniques can be fruitfully applied to study succinctness in other logical settings.

Our results also have practical implications for querying OWL 2 QL KBs. Indeed, our succinctness analysis provides strong evidence in favour of adopting NDL as the target language for rewritings, since we have identified a range of

[2]For presentation purposes, we use a minor variant of the reduction in [14].

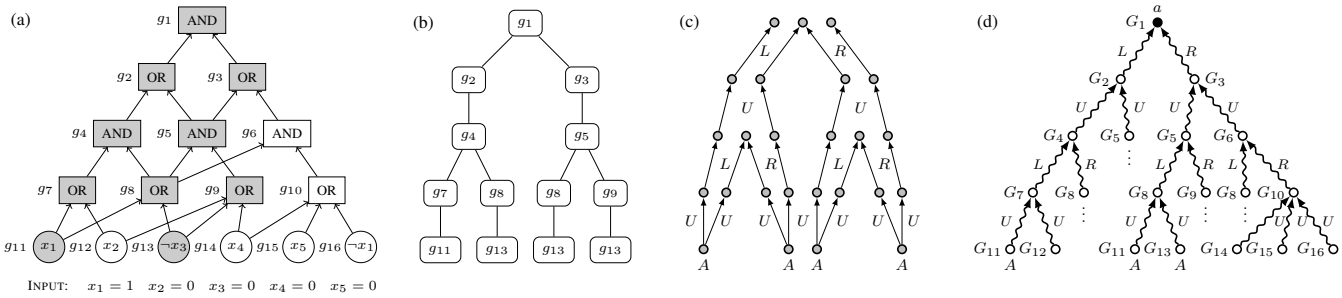


Fig. 6. (a) Example circuit $\boldsymbol{C}^*$ with input $\mathbf{x}^*$ (b) proof tree for $\boldsymbol{C}^*$ and $\mathbf{x}^*$ (c) query $\mathbf{q}_{\boldsymbol{C}^*}$ (d) canonical model for KB $(\mathcal{T}^{\mathbf{x}^*}_{\boldsymbol{C}^*}, \mathcal{A}_{\boldsymbol{C}^*})$.

query-ontology pairs for which polysize NDL-rewritings are guaranteed, but PE-rewritings may be of superpolynomial size. Interestingly, we have proved that for these same classes of queries and ontologies, query answering is tractable (either in NL or in LOGCFL). We plan to marry these positive succinctness and complexity results by developing concrete NDL-rewriting algorithms for OWL 2 QL for which both the rewriting and evaluation phases run in polynomial time (as was done in [16] for DL-Lite$_{\mathsf{core}}$). Moreover, since NL and LOGCFL are considered highly parallelizable, it would also be interesting to explore parallel query answering algorithms.

REFERENCES

[1] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family," *J. of Automated Reasoning*, vol. 39, no. 3, pp. 385–429, 2007.

[2] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, "OWL 2 Web Ontology Language profiles," W3C Recommendation, 11 December 2012, available at http://www.w3.org/TR/owl2-profiles/. [Online]. Available: http://www.w3.org/TR/owl2-profiles/

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

[4] R. Rosati and A. Almatelli, "Improving query answering over DL-Lite ontologies," in *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*. AAAI Press, 2010, pp. 290–300.

[5] A. Chortaras, D. Trivela, and G. Stamou, "Optimized query rewriting for OWL 2 QL," in *Proc. of the 23rd Int. Conf. on Automated Deduction (CADE-23)*, ser. LNCS, vol. 6803. Springer, 2011, pp. 192–206.

[6] G. Gottlob, G. Orsi, and A. Pieris, "Ontological queries: Rewriting and optimization," in *Proc. of the 27th Int. Conf. on Data Engineering (ICDE 2011)*. IEEE Computer Society, 2011, pp. 2–13.

[7] R. Rosati, "Prexto: Query rewriting under extensional constraints in DL-Lite," in *Proc. of the 9th Extended Semantic Web Conf. (EWSC 2012)*, ser. LNCS, vol. 7295. Springer, 2012, pp. 360–374.

[8] H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin, "Evaluation of query rewriting approaches for OWL 2," in *Proc. of SSWS+HPCSW 2012*, vol. 943. CEUR-WS, 2012.

[9] T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao, "Query rewriting for Horn-SHIQ plus rules," in *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012.

[10] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo, "A sound and complete backward chaining algorithm for existential rules," in *Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems (RR 2012)*, ser. LNCS, vol. 7497. Springer, 2012, pp. 122–138.

[11] M. Rodríguez-Muro, R. Kontchakov, and M. Zakharyaschev, "Ontology-based data access: Ontop of databases," in *Proc. of the 12th Int. Semantic Web Conf. (ISWC 2013)*, ser. LNCS, vol. 8218. Springer, 2013, pp. 558–573.

[12] S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev, "Exponential lower bounds and separation for query rewriting," in *Proc. of the 39th Int. Colloquium on Automata, Languages, and Programming (ICALP 2012), Part II*, ser. LNCS, vol. 7392. Springer, 2012, pp. 263–274.

[13] ——, "On the succinctness of query rewriting over OWL 2 QL ontologies with shallow chases," in *Proc. of the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2014)*. ACM Press, 2014.

[14] G. Gottlob, N. Leone, and F. Scarcello, "The complexity of acyclic conjunctive queries," *J. ACM*, vol. 48, no. 3, pp. 431–498, 2001.

[15] M. Yannakakis, "Algorithms for acyclic database schemes," in *Proc. of the 7th Int. Conf. on Very Large Data Bases (VLDB'81)*. IEEE Computer Society, 1981, pp. 82–94.

[16] M. Bienvenu, M. Ortiz, M. Simkus, and G. Xiao, "Tractable queries for lightweight description logics," in *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*. AAAI Press, 2013.

[17] C. Chekuri and A. Rajaraman, "Conjunctive query containment revisited," *Theoretical Computer Science*, vol. 239, no. 2, pp. 211–229, 2000.

[18] G. Gottlob, N. Leone, and F. Scarcello, "Computing LOGCFL certificates," in *ICALP-99*, 1999, pp. 361–371.

[19] H. Venkateswaran, "Properties that characterize LOGCFL," *J. Computer and System Sciences*, vol. 43, no. 2, pp. 380–404, 1991.

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

[21] S. Jukna, *Boolean Function Complexity: Advances and Frontiers*. Springer, 2012.

[22] G. Gottlob and T. Schwentick, "Rewriting ontological queries into small nonrecursive datalog programs," in *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012)*. AAAI Press, 2012, pp. 254–263.

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

[24] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev, "The combined approach to query answering in DL-Lite," in *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*. AAAI Press, 2010.

[25] "Hypergraphes arborés," *Discrete Mathematics*, vol. 21, no. 3, pp. 223 – 227, 1978.

[26] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[27] A. Bretto, *Hypergraph Theory: An Introduction*. Springer Publishing Company, Incorporated, 2013.

[28] H. Vollmer, *Introduction to circuit complexity - a uniform approach*, ser. Texts in theoretical computer science. Springer, 1999.

[29] A. Razborov, "Lower bounds for deterministic and nondeterministic branching programs," in *Proc. of the 8th Int. Symposium on Fundamentals of Computation Theory (FCT'91)*, ser. LNCS, vol. 529. Springer, 1991, pp. 47–60.

[30] M. Karchmer and A. Wigderson, "Monotone circuits for connectivity require super-logarithmic depth," in *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC 1988)*. ACM Press, 1988, pp. 539–550.

[31] S. A. Cook, "Characterizations of pushdown machines in terms of time-bounded computers," *J. ACM*, vol. 18, no. 1, pp. 4–18, 1971.

[32] I. H. Sudborough, "On the tape complexity of deterministic context-free languages," *Journal of the ACM*, vol. 25, no. 3, pp. 405–414, 1978.

[33] H. Venkateswaran, "Properties that characterize LOGCFL," *Journal of Computer and System Sciences*, vol. 43, no. 2, pp. 380–404, 1991.

APPENDIX

PROOFS FOR SECTION II

Theorem 5. Thm. 4 remains true if $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ is replaced by $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$:

$$f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}} = \bigvee_{\substack{\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}} \\ \text{independent}}} \Big(\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} p_\eta \wedge \bigwedge_{\mathfrak{t} \in \Theta} \big(\bigwedge_{z,z' \in \mathfrak{t}} p_{z=z'} \wedge \bigvee_{\substack{\varrho \in \mathsf{N}_2^{\pm}, \\ \mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]}} \bigwedge_{z \in \mathfrak{t}} p^{\varrho}_z \big) \Big)$$

Remark. In fact, Theorem 4 was proved in [13] only for *consistent* KBs. However, it is known that it is possible to define a short PE-query $\mathbf{q}^{\perp}_{\mathcal{T}}$ that when evaluated on $\mathcal{I}_{\mathcal{A}}$ returns all k-tuples of individual constants on $\mathcal{I}_{\mathcal{A}}$ if the KB $(\mathcal{T},\mathcal{A})$ is inconsistent, and returns no answers otherwise, cf. [33]. It follows that if $\mathbf{q}'$ is a rewriting for $\mathbf{q}$ and $\mathcal{T}$ for all data instances $\mathcal{A}$ that are consistent with $\mathcal{T}$, then we can obtain a rewriting for $\mathbf{q}$ and $\mathcal{T}$ (that works for all data instances) by taking the disjunction of $\mathbf{q}'$ and $\mathbf{q}^{\perp}_{\mathcal{T}}$. Therefore, to prove Theorem 5, it sufficient to show how to construct such "consistent rewritings".

Proof. Let $\mathcal{T}$ be an OWL 2 QL ontology and $\mathbf{q} = \exists \mathbf{y}\, \varphi(\mathbf{x},\mathbf{y})$ be a CQ with answer variables $\mathbf{x}$ and existential variables $\mathbf{y}$. (we will use z and z' when referring to variables of either type). We begin by recalling that every atom $\eta(\mathbf{u})$ has the following simple PE-rewriting:

$$\rho_\eta = \bigvee_{\mathcal{T} \models \xi(\mathbf{u}) \to \eta(\mathbf{u})} \xi(\mathbf{u})$$

where $\xi(\mathbf{u})$ ranges over $\varrho(\mathbf{u})$ ($\varrho \in \mathsf{N}_2^{\pm}$) when $|\mathbf{u}| = 2$ and over

$$\tau(u) \;::=\; A(u) \quad (A \in \mathsf{N}_1) \quad | \quad \exists v\, \varrho(u,v) \quad (\varrho \in \mathsf{N}_2^{\pm})$$

when $\mathbf{u}$ consists of the single variable u.

To show the first statement, consider a Boolean formula χ that computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$, and let $\mathbf{q}'$ be the FO-formula obtained from χ as follows:

- replace $p_{z=z'}$ by the equality $z = z'$;
- replace p_η by its PE-rewriting ρ_η;
- replace p^{ϱ}_z by the PE-rewriting $\rho_\varrho(z)$ of $\exists y \varrho(z,y)$;
- existentially quantify the variables $\mathbf{y}$.

Note that $\mathbf{q}'$ has the same answer variables as $\mathbf{q}$, and if χ is a monotone formula, then $\mathbf{q}'$ is a PE-formula.

We wish to show that $\mathbf{q}'$ is a consistent rewriting of $\mathbf{q}$ and $\mathcal{T}$ (cf. preceding remark). To do so, we let $\mathbf{q}''$ be the PE-formula obtained by applying the above transformation to the original monotone Boolean formula $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$:

$$\mathbf{q}'' = \exists \mathbf{y} \bigvee_{\substack{\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}} \\ \text{independent}}} \Big(\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} \rho_\eta \;\wedge \bigwedge_{\mathfrak{t} \in \Theta} \big(\bigwedge_{z,z' \in \mathfrak{t}} z = z' \wedge \bigvee_{\substack{\varrho \in \mathsf{N}_2^{\pm}, \\ \mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]}} \bigwedge_{z \in \mathfrak{t}} \rho_\varrho(z) \big) \Big).$$

We know that χ and $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ compute the same Boolean function. It follows that $\mathbf{q}'$ and $\mathbf{q}''$ are equivalent FO-formulas. It thus suffices to show that $\mathbf{q}''$ is a consistent rewriting of $\mathbf{q}$ and $\mathcal{T}$. This is easily seen by comparing $\mathbf{q}''$ to the following query

$$\mathbf{q}''' = \exists \mathbf{y}' \bigvee_{\substack{\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}} \\ \text{independent}}} \Big(\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} \rho_\eta \;\wedge \bigwedge_{\mathfrak{t} \in \Theta} \big(\bigvee_{\substack{\varrho \in \mathsf{N}_2^{\pm}, \\ \mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]}} \exists z \, (\rho_\varrho(z) \wedge \bigwedge_{z' \in \mathfrak{t}_r} z' = z) \big) \Big)$$

which was proven in [13] to be a consistent FO-rewriting of $\mathbf{q}$ and $\mathcal{T}$ (here $\mathbf{y}'$ is the restriction of $\mathbf{y}$ to the variables in $\mathbf{q}'''$).

The proof of the second statement concerning NDL-rewritings closely follows the proof of Theorem 4 from [], but we include it for the sake of completeness. First, we define a unary predicate D_0 that contains all individual constants of the given data instance. This is done by taking the rules

$$\varrho(u) \to D_0(u), \tag{2}$$

where $\varrho(u)$ is of the form $S(u)$, $S(u,v)$ and $S(v,u)$, for some predicate $S \in \mathsf{sig}(\mathcal{T}) \cup \mathsf{sig}(\mathbf{q})$. Next, we let $\mathbf{z} = \mathbf{x} \cup \mathbf{y}$ and define a $|\mathbf{z}|$-ary predicate D using the following rule:

$$\bigwedge_{z \in \mathbf{z}} D_0(z) \to D(\mathbf{z}). \tag{3}$$

We need the predicate D to ensure that all the rules in our NDL program are safe, i.e. every variable that appears in the head of a rule also occurs in the body.

Now let $\boldsymbol{C}$ be a monotone circuit for $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ whose gates are $g_1, \dots, g_n$, with g_n the output gate. For input gates g_i whose variable is $p_{z=z'}$, we take the rule[3]

$$z = z' \land D(\mathbf{z}) \to G_i(\mathbf{z}). \tag{4}$$

For every input gate g_i whose variable is p_η, we include the rule

$$\xi \land D(\mathbf{z}) \to G_i(\mathbf{z}), \tag{5}$$

for every disjunct ξ of the rewriting $\rho_\eta(\mathbf{z})$ of η and $\mathcal{T}$ (here we assume w.lo.g. that any variable in ξ that does not appear in ν does not belong to $\mathbf{z}$). If instead g_i is associated with variable p^ϱ_z, then we use the rules

$$\xi \land D(\mathbf{z}) \to G_i(\mathbf{z}), \tag{6}$$

where ξ is a disjunct of the rewriting ρ_ϱ of $\varrho(z)$ and $\mathcal{T}$ (here again we assume that every variable that appears both in ξ and $\mathbf{z}$ also appears in the atom ϱ). The remaining (AND and OR) gates are encoded using the following rules:

$$G_{j_1}(\mathbf{z}) \land G_{j_2}(\mathbf{z}) \land D(\mathbf{z}) \to G_i(\mathbf{z}) \quad \text{if } g_i = g_{j_1} \land g_{j_2}; \tag{7}$$

$$\left.\begin{array}{l} G_{j_1}(\mathbf{z}) \land D(\mathbf{z}) \to G_i(\mathbf{z}) \\ G_{j_2}(\mathbf{z}) \land D(\mathbf{z}) \to G_i(\mathbf{z}) \end{array}\right\} \quad \text{if } g_i = g_{j_1} \lor g_{j_2}. \tag{8}$$

Denote the resulting set of rules (2)–(8) by Π. We note that Π is of size $O(|\boldsymbol{C}| \cdot |\mathcal{T}|)$ and further claim that (Π, G_n) is an NDL-rewriting of $\mathbf{q}$ and $\mathcal{T}$. To see why, observe that by "unfolding" these rules in the standard way, we can transform (Π, G_n) into an equivalent PE-formula of the form

$$\exists \mathbf{y} \Big[\psi(\mathbf{x}, \mathbf{y}) \land \bigwedge_{z \in Z} \Big(\bigvee_{\varrho(u) \to D_0(u) \in \Pi} \varrho(z) \Big) \Big],$$

where $Z \subseteq \mathbf{x} \cup \mathbf{y}$ and $\exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ can be constructed by taking the Boolean formula representing $\boldsymbol{C}$ and replacing p_η with ρ_η, $p_{z=z'}$ with $z = z'$ and p^ϱ_z with ρ_ϱ. We have already shown that $\exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ is a rewriting of $\mathbf{q}$ and $\mathcal{T}$ in the first half of the proof, and the additional conjuncts asserting that the variables in $\mathbf{z}$ appear in some predicate are trivially satisfied. □

Theorem 6 If $\mathbf{q}'$ is a (PE-) FO-rewriting of $\mathbf{q}$ and $\mathcal{T}$, then there is a (monotone) Boolean formula χ of size $O(|\mathbf{q}'|)$ which computes $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$. If (Π, G) is an NDL-rewriting of $\mathbf{q}$ and $\mathcal{T}$, then $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ is computed by a monotone Boolean circuit $\boldsymbol{C}$ of size $O(|\Pi|)$.

Proof (implicit in [13]). Given a PE-, FO- or NDL-rewriting $\mathbf{q}'$ of $\mathbf{q}$ and $\mathcal{T}$, we show how to construct, respectively, a monotone Boolean formula, a Boolean formula or a monotone Boolean circuit for the function $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ of size $|\mathbf{q}'|$.

Suppose $\mathbf{q}'$ is a PE-rewriting of $\mathbf{q}$ and $\mathcal{T}$. We eliminate the quantifiers in $\mathbf{q}'$ by first replacing every subformula of the form $\exists x\, \psi(x)$ in $\mathbf{q}'$ with $\psi(a)$, and then replacing each atom of the form $A(a)$ and $P(a, a)$ with the corresponding propositional variable. One can verify that the resulting propositional monotone Boolean formula computes $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$. If $\mathbf{q}'$ is an FO-rewriting of $\mathbf{q}$, then we eliminate the quantifiers by replacing $\exists x\, \psi(x)$ and $\forall x\, \psi(x)$ in $\mathbf{q}'$ with $\psi(a)$. We then proceed as before, replacing atoms $A(a)$ and $P(a, a)$ by the corresponding propositional variables, to obtain a Boolean formula computing $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$.

If (Π, G) is an NDL-rewriting of $\mathbf{q}$, then we replace all the variables in Π with a and then perform the replacement described above. Denote the resulting propositional NDL-program by Π'. The program Π' can now be transformed into a monotone Boolean circuit computing $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$. For every (propositional) variable p occurring in the head of a rule in Π', we introduce an OR-gate whose output is p and inputs are the bodies of the rules with head p; for each such body, we introduce an AND-gate whose inputs are the propositional variables in the body. □

[3]For ease of notation, we use equality atoms in rule bodies, but these can be removed using standard (equality-preserving) transformations.

PROOFS FOR SECTION III

Theorem 8. Let $P = (H_P, \mathfrak{l}_P)$ be a THGP. For every input α for P, $f_P(\alpha) = 1$ iff $f^{\mathsf{prim}}_{\mathbf{q}_P, \mathcal{T}_P}(\gamma) = 1$, where γ is defined as follows: $\gamma(B_e) = 1$, $\gamma(R_e) = \gamma(R'_e) = 0$, and $\gamma(S_{ij}) = \gamma(S'_{ij}) = \alpha(\mathfrak{l}_P(\{v_i, v_j\}))$.

Proof. Consider a THGP $P = (H_P, \mathfrak{l}_P)$ whose underlying tree T has vertices $v_1, \ldots, v_n$, and let $T^{\downarrow}$ be the directed tree obtained from T by fixing one of its leaves v_1 as the root and orienting edges away from v_1. In what follows, we will say that a vertex $v \in V_T$ is an *internal vertex in* $e \in E_P$ *(w.r.t. T)* if it appears in e and is neither a leaf nor a boundary vertex of e w.r.t. T. Note that because we chose a leaf as root of $T^{\downarrow}$, we know that for every hyperedge e, the highest vertex in e (according to $T^{\downarrow}$) must be either a leaf or a boundary vertex of e.

Take some $\alpha : L_P \to \{0, 1\}$ and let γ be as defined in the theorem statement. Define the corresponding data instance:

$$\mathcal{A}_\gamma = \{B_e(a) \mid e \in E_P\} \cup \{S_{ij}(a, a), S'_{ij}(a, a) \mid \gamma(S_{ij}) = \gamma(S'_{ij}) = \alpha(\mathfrak{l}_P(\{v_i, v_j\})) = 1\}.$$

For the first direction, suppose that $f_P(\alpha) = 1$. Then we know that there exists $E' \subseteq E_P$ that is independent and covers all zeros of α. To show $f^{\mathsf{prim}}_{\mathbf{q}_P, \mathcal{T}_P}(\gamma) = 1$, we must show that $\mathcal{T}_P, \mathcal{A}_\gamma \models \mathbf{q}_P$. Define a mapping h as follows:

- $h(y_i) = aR_eR'_e$ if v_i is an internal vertex of $e \in E'$. Otherwise, $h(y_i) = a$.
- $h(y_{ij}) = aR_e$ if $\{v_i, v_j\} \in e$ and $e \in E'$. Otherwise, $h(y_{ij}) = a$.

Note that h is well-defined: since E' is independent, different hyperedges in E' cannot share internal vertices, and there can be at most one hyperedge $e \in E'$ that contains a given edge $\{v_i, v_j\}$.

It remains to show that h is a homomorphism from $\mathbf{q}_P$ to $\mathcal{C}_{\mathcal{T}_P, \mathcal{A}_\gamma}$. Consider a pair of atoms $S_{ij}(y_i, y_{ij}), S'_{ij}(y_{ij}, y_j)$ in $\mathbf{q}_P$. Then $(v_i, v_j) \in T^{\downarrow}$, so either $\alpha(\{v_i, v_j\}) = 1$ or there is some $e \in E'$ such that $\{v_i, v_j\} \in e$.

In the former case, we have $\gamma(S_{ij}) = \gamma(S'_{ij}) = 1$, so $\mathcal{A}_\gamma$ contains $S_{ij}(a, a)$ and $S'_{ij}(a, a)$. If there is no $e \in E'$ such that $\{v_i, v_j\} \in e$, then $h(y_i) = h(y_{ij}) = h(y_j) = a$, so the atoms $S_{ij}(y_i, y_{ij}), S'_{ij}(y_{ij}, y_j)$ are satisfied by h.

Now consider the alternative in which $e = \langle v_{k_1}, \ldots, v_{k_m} \rangle \in E'$ is such that $\{v_i, v_j\} \in e$ and $e \in E'$. Note that because boundary vertices must have degree 2 (recall that this condition is part of the definition of THGPs), we know that all boundary and leaf vertices of e must be among $v_{k_1}, \ldots, v_{k_m}$. Moreover, we may assume without loss of generality that $v_{k_1}, \ldots, v_{k_m}$ are all either boundary vertices or leaves of T (since any internal vertex v_{k_ℓ} can be dropped without changing the meaning of e). Note that this ensures that for all $v_i \in e$, $h(y_i) = a$ iff $v_i \in \{v_{k_1}, \ldots, v_{k_m}\}$. There are four possibilities to consider:

- Case 1: $\{v_i, v_j\} \subseteq \{v_{k_1}, \ldots, v_{k_m}\}$ *(i.e. neither of v_i and v_j is internal)*. We know that the boundary vertices of e have degree 2, so the only possibility is that $e = \{\{v_i, v_j\}\}$. We therefore have $h(v_i) = h(v_j) = a$ and $h(v_{ij}) = aR_e$, and the ontology $\mathcal{T}_P$ contains $R_e(x, y) \to S_{ij}(x, y)$ and $R_e(y, x) \to S'_{ij}(x, y)$.
- Case 2: $v_i \in \{v_{k_1}, \ldots, v_{k_m}\}$ but $v_j \notin \{v_{k_1}, \ldots, v_{k_m}\}$. *(i.e. v_i is a boundary vertex or leaf and v_j is internal)*
  We have $h(v_i) = a$, $h(v_{ij}) = aR_e$, and $h(v_j) = aR_eR'_e$, and the ontology contains $R_e(x, y) \to S_{ij}(x, y)$ and $R'_e(x, y) \to S'_{ij}(x, y)$.
- Case 3: $v_j \in \{v_{k_1}, \ldots, v_{k_m}\}$ but $v_i \notin \{v_{k_1}, \ldots, v_{k_m}\}$ *(i.e. v_j is a boundary vertex or leaf and v_i is internal)*
  Then we have $h(v_j) = a$, $h(v_{ij}) = aR_e$, and $h(v_i) = aR_eR'_e$, and the ontology contains $R_e(y, x) \to S'_{ij}(x, y)$ and $R'_e(y, x) \to S_{ij}(x, y)$.
- Case 4: $\{v_i, v_j\} \cap \{v_{k_1}, \ldots, v_{k_m}\} = \emptyset$ *(i.e. both are internal vertices)*. Then we have $h(v_i) = h(v_j) = aR_eR'_e$ and $h(v_{ij}) = aR_e$, and the ontology contains $R_e(y, x) \to S_{ij}(x, y)$ and $R'_e(x, y) \to S'_{ij}(x, y)$.

In all cases, we find that h satisfies the atoms $S_{ij}(y_i, y_{ij}), S'_{ij}(y_{ij}, y_j)$. We can thus conclude that h is indeed a homomorphism.

For the other direction, suppose that $f^{\mathsf{prim}}_{\mathbf{q}_P, \mathcal{T}_P}(\alpha) = 1$. Then we have $\mathcal{T}_P, \mathcal{A}_\gamma \models \mathbf{q}_P$, so there is a homomorphism $h : \mathbf{q}_P \to \mathcal{C}_{\mathcal{T}_P, \mathcal{A}_\gamma}$. We wish to show that there exists a subset of E_P that is independent and covers all zeros of α. Let us define E' as the set of all $e \in E$ such that $h^{-1}(aR_e) \neq \emptyset$ (that is, aR_e is in the image of h).

To show that E' is independent, we start by establishing the following claim:

Claim. If $h^{-1}(aR_e) \neq \emptyset$, $y_{ij} \in \mathsf{vars}(\mathbf{q}_H)$, and $\{v_i, v_j\} \in e$, then $h(y_{ij}) = aR_e$.

Proof of claim. Suppose that $h^{-1}(aR_e) \neq \emptyset$, where $e = \langle v_{k_1}, \ldots, v_{k_m} \rangle \in E'$. We may assume w.l.o.g. that v_{k_1} is the highest vertex in e according to T', and that none of $v_{k_1}, \ldots, v_{k_m}$ is an internal vertex. Now pick some variable $z \in h^{-1}(aR_e)$ such that there is no $z' \in h^{-1}(aR_e)$ that is higher than z in $\mathbf{q}_P$ (here we use the ordering of variables induced by the tree $T^{\downarrow}$). We first note that z cannot be of the form y_j, since then there is an atom in $\mathbf{q}_P$ of the form $S_{j\ell}(y_j, y_{j\ell})$ or $S'_{\ell j}(y_{\ell j}, y_j)$, and aR_e does not have any outgoing $S_{j\ell}$ or $S'^{-}_{\ell j}$ arcs in $\mathcal{C}_{\mathcal{T}_P, \mathcal{A}_\gamma}$. It follows that $z = y_{j\ell}$. By again considering the available arcs leaving aR_e, we can further see that $\{v_j, v_\ell\} \in e$. We next wish to show that $j = k_1$. Suppose that this is not the case. Then, we know that there must exist some edge $\{v_p, v_j\} \in e$ such that $(v_p, v_j) \in T^{\downarrow}$. A simple examination of the axioms in $\mathcal{T}_P$ shows that the only way for h to satisfy the atom $S_{j\ell}(y_j, y_{j\ell})$ is to map y_j to $aR_eR'_e$. It follows that to satisfy that atom $S'_{pj}(y_{pj}, y_j)$,

we must have $h(y_{pj}) = aR_e$. This contradicts our earlier assumption that $z = y_{j\ell}$ was a highest vertex in $h^{-1}(aR_e)$. We thus have $j = k_1$. Now using a simple inductive argument on the distance from y_{k_1}, and considering the possible ways of mapping the atoms of $\mathbf{q}_H$, we can show that $h(y_{ij}) = aR_e$ for every $\{v_i, v_j\} \in e$. (*end proof of claim*)

Suppose that there are two distinct hyperedges $e, e' \in E'$ that have a non-empty intersection: $\{v_i, v_j\} \in e \cap e'$. We know that either y_{ij} or y_{ji} belongs to $\mathsf{vars}(\mathbf{q}_P)$, and we can suppose w.l.o.g. that it is the former. We can thus apply the preceding claim to obtain $h(y_{ij}) = aR_e = aR_{e'}$, a contradiction. We have thus shown that E' is independent, and so it only remains to show it covers all zeros. To this end, let $\{v_i, v_j\}$ be such that $\alpha(\{v_i, v_j\}) = 0$ and again suppose w.l.o.g. that $y_{ij} \in \mathsf{vars}(\mathbf{q}_P)$. Then $\mathcal{A}_\gamma$ does not contain $S_{ij}(a, a)$, so the only way h can satisfy the query atom $S_{ij}(y_i, y_{ij})$ is by mapping y_{ij} to some element aR_e such that $\{v_i, v_j\} \in e$. It follows that there is some $e \in E'$ such that $\{v_i, v_j\} \in e$, so all zeros of α are covered by E'. We have thus shown that E' is an independent subset of E_P that covers all zeros of α, and hence we conclude that $f_P(\alpha) = 1$. □

Theorem 9. Fix $t \geq 1$ and $d \geq 0$. For every ontology $\mathcal{T}$ of depth $\leq d$ and CQ $\mathbf{q}$ of treewidth $\leq t$, there is a monotone THGP that computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ of size polynomial in $|\mathcal{T}| + |\mathbf{q}|$.

More specifically, we show the following:

Proposition. For every ontology $\mathcal{T}$ and CQ $\mathbf{q}$, the THGP $(H_{\mathbf{q},\mathcal{T}}, \mathfrak{l}_{\mathbf{q},\mathcal{T}})$ defined in Section III.C computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$. If $\mathcal{T}$ has depth d and $\mathbf{q}$ has treewidth t, then $H_{\mathbf{q},\mathcal{T}}$

- contains at most $(2M + 1)L$ vertices;
- contains at most $L(M + M^2)$ hyperedges;
- has labels with at most $(2|\mathcal{T}| + |\mathbf{q}| + 1)|\mathbf{q}|$ conjuncts.

where $L = (2|\mathbf{q}| - 1)^2$ and $M = |W^t_d| \leq (2|\mathcal{T}|)^d$.

Proof. Let (T, λ) be the tree decomposition of $G_{\mathbf{q}}$ of width t that was used to construct the THGP $(H_{\mathbf{q},\mathcal{T}}, \mathfrak{l}_{\mathbf{q},\mathcal{T}})$. We may assume w.l.o.g. that T contains at most $(2|\mathbf{q}| - 1)^2$ nodes, cf. [34]. Recall that to more easily refer to the variables in $\lambda(N)$, we make use of functions $\lambda_1, \ldots, \lambda_t$ such that $\lambda_i(N) \in \lambda(N)$ and $\lambda(N) = \cup_i \lambda_i(N)$. Further recall that the formula $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ is defined as follows:

$$f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}} = \bigvee_{\substack{\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}} \\ \text{independent}}} \left(\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} p_\eta \wedge \bigwedge_{\mathfrak{t} \in \Theta} \Big(\bigwedge_{z,z' \in \mathfrak{t}} p_{z=z'} \wedge \bigvee_{\substack{\varrho \in \mathsf{N}_2^\pm, \\ \mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]}} \bigwedge_{z \in \mathfrak{t}} p^\varrho_z \Big) \right)$$

where $\mathbf{q}_\Theta = \bigcup_{\mathfrak{t}} \mathbf{q}_{\mathfrak{t}}$. Throughout the proof, we use f_P to denote the function computed by the THGP $(H_{\mathbf{q},\mathcal{T}}, \mathfrak{l}_{\mathbf{q},\mathcal{T}})$. Note that by definition f_P uses exactly the same set of propositional variables as $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$.

To show the first direction of the first statement, let $\mathbf{v}$ be a valuation of the variables in $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ such that $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}(\mathbf{v}) = 1$. Then we can find an independent subset $\Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}}$ such that $\mathbf{v}$ satisfies the corresponding disjunct of $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$:

$$\bigwedge_{\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta} p_\eta \wedge \bigwedge_{\mathfrak{t} \in \Theta} \Big(\bigwedge_{z,z' \in \mathfrak{t}} p_{z=z'} \wedge \bigvee_{\substack{\varrho \in \mathsf{N}_2^\pm, \\ \mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho]}} \bigwedge_{z \in \mathfrak{t}} p^\varrho_z \Big) \tag{9}$$

For every $\mathfrak{t} \in \Theta$, we let $\varrho_{\mathfrak{t}}$ be a role that makes the final disjunction hold. Furthermore, we choose some homomorphism $h_{\mathfrak{t}} : \mathbf{q}_{\mathfrak{t}} \to \mathcal{C}_{\mathcal{T}_{\varrho_{\mathfrak{t}}}, \mathcal{A}_{\varrho_{\mathfrak{t}}}}$, where $\mathcal{T}_{\varrho_{\mathfrak{t}}} = \mathcal{T} \cup \{A_{\varrho_{\mathfrak{t}}}(x) \to \exists y \varrho_{\mathfrak{t}}(x, y)\}$ and $\mathcal{A}_{\varrho_{\mathfrak{t}}} = \{A_{\varrho_{\mathfrak{t}}}(a)\}$. Such homomorphisms are guaranteed to exist by the definition of tree witnesses.

Now for each node N in the tree decomposition T, we define $\mathbf{w}_N$ by setting:

- $\mathbf{w}_N[j] = \varepsilon$ if $\lambda_j(N) = z$ and either z appears in an atom η such that $\mathbf{v}(p_\eta) = 1$ or there is some $\mathfrak{t} \in \Theta$ such that $z \in \mathfrak{t}_r$.
- $\mathbf{w}_N[j] = w$ if $\lambda_j(N) = z$ and there is some $\mathfrak{t} \in \Theta$ such that $z \in \mathfrak{t}_i$ and $h_{\mathfrak{t}}(z) = aw$.

First note that $\mathbf{w}_N$ is well-defined since the independence of Θ guarantees that every variable in $\mathbf{q}$ can appear in $\mathfrak{t}_i$ for at most one tree witness $\mathfrak{t} \in \Theta$. Moreover, every variable in $\mathbf{q}$ must either belong to an atom η such that $\mathbf{v}(p_\eta) = 1$ or to an atom that is contained in $\mathbf{q}_{\mathfrak{t}}$ for some $\mathfrak{t} \in \Theta$.

Next we show that $\mathbf{w}_N$ is consistent with the node N. To show that the first condition holds, consider some atom $A(\lambda_i(N)) \in \mathbf{q}$ such that $\mathbf{w}_N[i] \neq \varepsilon$. Then there must be a tree witness $\mathfrak{t} \in \Theta$ such that $\lambda_i(N) \in \mathfrak{t}_i$, in which case we have that $h_{\mathfrak{t}}(\lambda_i(N)) = a\mathbf{w}_N[i]$. Let $\varsigma \in \mathsf{N}_2^\pm$ be the final symbol in $\mathbf{w}_N[i]$. Then since $h_{\mathfrak{t}}$ is a homomorphism from $\mathbf{q}_{\mathfrak{t}}$ into $\mathcal{C}_{\mathcal{T}_{\varrho_{\mathfrak{t}}}, \mathcal{A}_{\varrho_{\mathfrak{t}}}}$, it must be the case that $\mathcal{T} \models \exists y \varsigma(y, x) \to A(x)$.

To show the second condition holds, consider some atom $R(\lambda_i(N), \lambda_j(N)) \in \mathbf{q}$ such that either $\mathbf{w}_N[i] \neq \varepsilon$ or $\mathbf{w}_N[j] \neq \varepsilon$. We suppose w.l.o.g. that $\mathbf{w}_N[i] \neq \varepsilon$ (the other case is handled analogously). It follows from the definition of $\mathbf{w}_N$ that there

must exist a tree witness $\mathfrak{t} \in \Theta$ such that $\lambda_i(N) \in \mathfrak{t}_i$ and $h_{\mathfrak{t}}(\lambda_i(N)) = a\mathbf{w}_N[i]$. Since $\lambda_i(N) \in \mathfrak{t}_i$ and $R(\lambda_i(N), \lambda_j(N)) \in \mathbf{q}$, the definition of tree witnesses ensures that $\lambda_j(N) \in \mathfrak{t}$. Because $h_{\mathfrak{t}}$ is a homomorphism from $\mathbf{q}_{\mathfrak{t}}$ into $\mathcal{C}_{\mathcal{T}_{\varrho_{\mathfrak{t}}}, \mathcal{A}_{\varrho_{\mathfrak{t}}}}$, we know that one of the following must hold:

- $\lambda_j(N) \in t_r$, $\mathbf{w}_N[j] = \varepsilon$, and $\mathbf{w}_N[i] = \varsigma$ for some $\varsigma \in \mathsf{N}_2^{\pm}$ such that $\mathcal{T} \models \varsigma(y, x) \rightarrow R(x, y)$
- $\lambda_j(N) \in t_i$ and $\mathbf{w}_N[i] = \mathbf{w}_N[j] \cdot \varsigma$ for some $\varsigma \in \mathsf{N}_2^{\pm}$ such that $\mathcal{T} \models \varsigma(y, x) \rightarrow R(x, y)$
- $\lambda_j(N) \in t_i$ and $\mathbf{w}_N[j] = \mathbf{w}_N[i] \cdot \varsigma$ for some $\varsigma \in \mathsf{N}_2^{\pm}$ such that $\mathcal{T} \models \varsigma(x, y) \rightarrow R(x, y)$

This establishes the second consistency condition.

We must also show that the pairs associated with different nodes in T are compatible. To this end, consider a pair of nodes N_1 and N_2 and the corresponding tuples of words $\mathbf{w}_{N_1}$ and $\mathbf{w}_{N_2}$. It is clear from the way we defined $\mathbf{w}_{N_1}$ and $\mathbf{w}_{N_2}$ that if $\lambda_i(N_1) = \lambda_j(N_2)$, then we must have $\mathbf{w}_{N_1}[i] = \mathbf{w}_{N_2}[j]$.

Now consider the set E' of hyperedges in $H_{\mathbf{q},\mathcal{T}}$ that contains:

- for every N_i in T, the hyperedge $E_i^k = \langle u_{ij_1}^k, \ldots, u_{ij_n}^k \rangle$, where k is such that $\xi_k = \mathbf{w}_{N_i}$, and $N_{j_1}, \ldots, N_{j_n}$ are the neighbours of N_i;
- for every pair of adjacent nodes N_i, N_j in T, the hyperedge $E_{ij}^{km} = \langle v_{ij}^k, v_{ji}^m \rangle$, where k and m are such that $\xi_k = \mathbf{w}_{N_i}$ and $\xi_m = \mathbf{w}_{N_j}$.

Note that the aforementioned hyperedges all belong to $H_{\mathbf{q},\mathcal{T}}$ since we showed that each $\mathbf{w}_{N_i}$, is consistent with node N_i, and that $\mathbf{w}_{N_i}$ and $\mathbf{w}_{N_j}$ are compatible with (N_i, N_j) for all pairs of nodes (N_i, N_j) in T. It is easy to see that E' is independent, since whenever we include E_i^k or E_{ij}^{km}, we do not include any $E_i^{k'}$ or $E_{ij}^{k'm}$ for $k' \neq k$. To see why E' covers all zeros, consider a vertex F of $H_{\mathbf{q},\mathcal{T}}$ (= an edge in T') that evaluates to 0 under $\mathbf{v}$. There are several cases to consider:

- $F = \{N_i, u_{ij}^1\}$: F is covered by the hyperedge in E' of the form E_i^k
- $F = \{v_{ij}^\ell, u_{ij}^{\ell+1}\}$: then F is either covered by the hyperedge in E' of the form E_i^k (if $k \leq \ell + 1$) or by the hyperedge E_{ij}^{km} (if $k > \ell + 1$)
- $F = \{v_{ij}^M, v_{ji}^M\}$: then F is covered by the hyperedge in E' of the form E_{ij}^{km}
- $F = \{u_{ij}^\ell, v_{ij}^\ell\}$ with $\xi_\ell = \mathbf{w}$: then F is either covered by the hyperedge in E' of the form E_i^k (if $\ell < k$) or by the hyperedge E_{ij}^{km} (if $k > \ell$), or we have $\mathbf{w} = \mathbf{w}_{N_i}$. In the latter case, we know that F is labeled by the conjunction of the following variables:
  - p_η, if $\mathsf{vars}(\eta) \subseteq \lambda(N_i)$ and $\lambda_g(N_i) \in \mathsf{vars}(\eta)$ implies $\mathbf{w}[g] = \varepsilon$
  - p_z^ϱ, if $\mathsf{vars}(\eta) = \{z\}$, $z = \lambda_g(N_i)$, and $\mathbf{w}[g] = \varrho w'$
  - p_z^ϱ, $p_{z'}^\varrho$, and $p_{z=z'}$, if $\mathsf{vars}(\eta) = \{z, z'\}$, $z = \lambda_g(N_i)$, $z' = \lambda_{g'}(N_i)$, and either $\mathbf{w}[g] = \varrho w'$ or $\mathbf{w}[g'] = \varrho w'$

  Since F evaluates to false, one of these variables must be be assigned 0 under $\mathbf{v}$. First suppose that p_η is in the label and $\mathbf{v}(p_\eta) = 0$. Then since Equation (9) is satisfied, it must be the case that η belongs to some $\mathbf{q}_{\mathfrak{t}}$, but the fact that $\lambda_g(N_i) \in \mathsf{vars}(\eta)$ implies $\mathbf{w}_{N_i}[g] = \varepsilon$ means that all variables in η must belong to $\mathfrak{t}_r$, contradicting the fact that $\mathbf{q}_{\mathfrak{t}}$ contains only atoms that have at least one variable in $\mathfrak{t}_i$. Next suppose that one of p_z^ϱ, $p_{z'}^\varrho$, and $p_{z=z'}$ is part of the label and evaluates to 0 under $\mathbf{v}$. We focus on the case where these variables came from a role atom with distinct variables, but the proof is entirely similar if p_z^ϱ is present because of a unary atom (item 2 above). Then we know that there is some atom η with $\mathsf{vars}(\eta) = \{z, z'\}$, $z = \lambda_g(N_i)$, $z' = \lambda_{g'}(N_i)$, and either $\mathbf{w}[g] = \varrho w'$ or $\mathbf{w}[g'] = \varrho w'$. It follows that there is a tree witness $\mathfrak{t} \in \Theta$ such that $z \in \mathfrak{t}$. This means that the atom $p_{z=z'}$ must be a conjunct of Equation (9), and so it must be satisfied under $\mathbf{v}$. Moreover, the fact that $\mathbf{w}[g] = \varrho w'$ or $\mathbf{w}[g'] = \varrho w'$ means that $\varrho_{\mathfrak{t}} = \varrho$, so the variables p_z^ϱ and $p_{z'}^\varrho$ are also satisfied under $\mathbf{v}$, contradicting our earlier assumption to the contrary.

We have thus shown that E' is independent and covers all zeros under $\mathbf{v}$, which means that $f_P(\mathbf{v}) = 1$.

For the other direction, suppose that $f_P(\mathbf{v}) = 1$, i.e. there is an independent subset E' of the hyperedges in $H_{\mathbf{q},\mathcal{T}}$ that covers all vertices that evaluate to 0 under $\mathbf{v}$. It is clear from the construction of $H_{\mathbf{q},\mathcal{T}}$ that the set E' must contain exactly one hyperedge of the form E_i^k for every node N_i in T, and exactly one hyperedge of the form E_{ij}^{km} for every edge $\{N_i, N_j\}$ in T. Moreover, if we have hyperedges E_i^k and $E_{ij}^{k'm}$ (resp. E_j^m and $E_{ij}^{km'}$), then it must be the case that $k = k'$ (resp. $m = m'$). We can thus associate with every node N_i the tuple $\mathbf{w}_{N_i} = \xi_k$. Since all zeros are covered, we know that for every node N_i, the following variables are assigned to 1 by $\mathbf{v}$:

- p_η, if $\mathsf{vars}(\eta) \subseteq \lambda(N_i)$ and $\lambda_g(N_i) \in \mathsf{vars}(\eta)$ implies $\mathbf{w}[g] = \varepsilon$
- p_z^ϱ, if $\mathsf{vars}(\eta) = \{z\}$, $z = \lambda_g(N_i)$, and $\mathbf{w}[g] = \varrho w'$ $\quad (\star)$
- p_z^ϱ, $p_{z'}^\varrho$, and $p_{z=z'}$, if $\mathsf{vars}(\eta) = \{z, z'\}$, $z = \lambda_g(N_i)$, $z' = \lambda_{g'}(N_i)$, and either $\mathbf{w}[g] = \varrho w'$ or $\mathbf{w}[g'] = \varrho w'$

We know from the definition of the set of hyperedges in $H_{\mathbf{q},\mathcal{T}}$ that every $\mathbf{w}_{N_i}$ is consistent with N_i, and for adjacent nodes N_i, N_j, pairs $\mathbf{w}_{N_j}$ and $\mathbf{w}_{N_j}$ are compatible. Using the consistency and compatibility properties, and the connectedness condition of tree decompositions, we can infer that the pairs assigned to *any two nodes* N_i, N_j in T are compatible. Since every variable

must appear in at least one node label, it follows that we can associate a *unique* word w_z to every variable z in $\mathbf{q}$. Now let $\equiv$ be the smallest equivalence relation on the atoms of $\mathbf{q}$ that satisfies the following condition:

$$\text{If } y \in \mathsf{vars}(\mathbf{q}),\ w_y \neq \varepsilon,\ y \in \eta_1, \text{ and } y \in \eta_2, \text{ then } \eta_1 \equiv \eta_2.$$

Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be the queries corresponding to the equivalence classes of $\equiv$. It is easily verified that the queries $\mathbf{q}_i$ are pairwise disjoint. Moreover, if $\mathbf{q}_i$ contains only variables z with $w_z = \varepsilon$, then $\mathbf{q}_i$ consists of a single atom. We can show that the remaining $\mathbf{q}_i$ correspond to tree witnesses:

Claim. For every $\mathbf{q}_i$ that contains a variable y with $w_y \neq \varepsilon$:

1) there is a role ϱ_i such that every $w_y \neq \varepsilon$ begins by ϱ_i
2) there is a homomorphism h_i from $\mathbf{q}_i$ into $C_{\mathcal{T}_i,\mathcal{A}_i}$ where $\mathcal{T}_i = \mathcal{T} \cup \{A_{\varrho_i}(x) \rightarrow \exists y \varrho_i(x,y)\}$ and $\mathcal{A}_i = \{A_{\varrho_i}(a)\}$ (with A_{ϱ_i} fresh)
3) there is a tree witness $\mathfrak{t}^i$ for $\mathbf{q}$ and $\mathcal{T}$ generated by ϱ_i such that $\mathbf{q}_i = \mathbf{q}_{\mathfrak{t}^i}$

Proof of claim. From the way we defined $\mathbf{q}_i$, we know that there exists a sequence $Q_0, \ldots, Q_m$ of subsets of $\mathbf{q}$ such that $Q_0 = \{\eta_0\} \subseteq \mathbf{q}_i$ contains a variable y_0 with $w_{y_0} \neq \varepsilon$, $Q_m = \mathbf{q}_i$, and for every $1 \leq \ell \leq m$, $Q_{\ell+1}$ is obtained from Q_ℓ by adding an atom $\eta \in \mathbf{q} \setminus Q_\ell$ that contains a variable y that appears in Q_ℓ and is such that $w_y \neq \varepsilon$. By construction, every atom in $\mathbf{q}_i$ contains a variable y with $w_y \neq \varepsilon$. Let ϱ_i be the first letter of the word w_{y_0}, and for every $0 \leq \ell \leq m$, let h_ℓ be the function mapping every variable z in Q_ℓ to aw_z.

Statements 1 and 2 can be shown by induction. The base case is trivial. For the induction step, suppose that at stage ℓ, we know that every variable y in Q_ℓ with $w_y \neq \varepsilon$ begins by ϱ_i, and that h_ℓ is a homomorphism of Q_ℓ into the canonical model $C_{\mathcal{T}_i,\mathcal{A}_i}$. We let η be the unique atom in $Q_{\ell+1} \setminus Q_\ell$. Then we know that η contains a variable y that appears in Q_ℓ and is such that $w_y \neq \varepsilon$. If $\eta = B(y)$, then Statement 1 is immediate. For Statement 2, we let N be a node in T such that $\mathsf{vars}(\eta) \subseteq \lambda(N)$ (such a node must exist by the definition of tree decompositions), and let j be such that $\lambda_j(N) = y$. We know that $\mathbf{w}_N$ is consistent with N, so $w_y = \mathbf{w}_N[j]$ must end by a role ς with $\mathcal{T} \models \exists y \varsigma(y,x) \rightarrow B(x)$, which proves Statement 2. Next consider the other case in which η contains a variable other than y. Then η must be a role atom of the form $\eta = R(y,z)$ or $\eta = R(z,y)$. We give the argument for the case where $\eta = R(y,z)$ (the argument for $\eta = R(z,y)$ is entirely similar). Let N be a node in T such that $\mathsf{vars}(\eta) \subseteq \lambda(N)$, and let j,k be such that $\lambda_j(N) = y$ and $\lambda_k(N) = z$. We know that $\mathbf{w}_N$ is consistent with N, so one of the following must hold:

- $\mathbf{w}_N[k] = \mathbf{w}_N[j] \cdot \varsigma$ with $\mathcal{T} \models \varsigma(x,y) \rightarrow R(x,y)$
- $\mathbf{w}_N[j] = \mathbf{w}_N[k] \cdot \varsigma$ with $\mathcal{T} \models \varsigma(x,y) \rightarrow R(y,x)$

By definition, we have $w_y = \mathbf{w}_N[j]$ and $w_z = \mathbf{w}_N[k]$. Since w_y begins with ϱ_i, it follows that the same holds for w_z unless $w_z = \varepsilon$, which shows Statement 1. Moreover, we either have (i) $w_z = w_y\varsigma$ and $\mathcal{T} \models \varsigma(x,y) \rightarrow R(x,y)$, or (ii) $w_y = w_z\varsigma$ and $\mathcal{T} \models \varsigma(x,y) \rightarrow R(y,x)$. In both cases, it is clear from the way we defined $h_{\ell+1}$ that it is homomorphism from $Q_{\ell+1}$ to $C_{\mathcal{T}_i,\mathcal{A}_i}$, so Statement 2 holds.

Statement 3 now follows from Statements 1 and 2, the definition of $\mathbf{q}_i$, and the definition of tree witnesses. *(end proof of claim)*

Let Θ consist of all the tree witnesses $\mathfrak{t}^i$ obtained from the preceding claim. As the $\mathbf{q}_i$ are known to be disjoint, we have that the set $\{\mathbf{q}_{\mathfrak{t}^i} \mid \mathfrak{t}^i \in \Theta\}$ is independent. We aim to show that $\mathbf{v}$ satisfies the disjunct of $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ that corresponds to Θ (cf. Equation (9)). First consider some $\eta \in \mathbf{q} \setminus \mathbf{q}_\Theta$. Then we know that for every variable z in η, we have $w_z = \varepsilon$. Let N be a node such that $\mathsf{vars}(\eta) \subseteq \lambda(N)$. Then we know that $\lambda_g(N) \in \mathsf{vars}(\eta)$ implies $\mathbf{w}_N[g] = \varepsilon$. It follows from $(\star)$ that $\mathbf{v}(p_\eta) = 1$. Next consider a variable $p_{z=z'}$ such that there is an atom $\eta \in \mathfrak{t}^i$ with $\mathsf{vars}(\eta) = \{z, z'\}$ such that $z \neq z'$. Then since $\eta \in \mathbf{q}_i$, we know that either $w_z \neq \varepsilon$ or $w_{z'} \neq \varepsilon$. It follows from $(\star)$ that $\mathbf{v}(p_{z=z'}) = 1$. Finally, consider some $p_z^{\varrho_i}$ such that $z \in \mathfrak{t}^i$. First suppose that there is a unary atom $B(z) \in \mathbf{q}_{\mathfrak{t}^i}$. Then we know that $w_z \neq \varepsilon$, and so by the above claim, we must have $w_z = \varrho_i w'$. It follows that there is a node N in T such that $z = \lambda_g(N)$ and $\mathbf{w}_N[g] = \varrho_i w'$. From $(\star)$, we can infer that $p_z^{\varrho_i}$ evaluates to 1 under $\mathbf{v}$. The other possibility is that there exists a binary atom $\eta \in \mathbf{q}_{\mathfrak{t}^i}$ such that $\mathsf{vars}(\eta) = \{z, z'\}$. Let N be a node in T such that $z = \lambda_g(N)$ and $z' = \lambda_{g'}(N)$. Since $\mathbf{q}_{\mathfrak{t}^i} = \mathbf{q}_i$, we know that either $w_z \neq \varepsilon$ or $w_{z'} \neq \varepsilon$. From the above claim, this yields $\mathbf{w}_N[g] = \varrho_i w'$ or $\mathbf{w}_N[g'] = \varrho_i w'$. We can thus apply $(\star)$ to obtain $\mathbf{v}(p_z^{\varrho_i}) = 1$. To conclude, we have shown that $\mathbf{v}$ satisfies one of the disjuncts of $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$, so $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}(\mathbf{v}) = 1$.

For the second statement of the theorem, we recall that the tree T in the tree decomposition of $\mathbf{q}$ has at most $(2|\mathbf{q}| - 1)^2$ nodes and that the set W_d^t consists of all tuples of words $\{(w_1, \ldots, w_t) \mid w_i \in (\mathsf{N}_2^{\pm} \cap \mathsf{sig}(\mathcal{T}))^*, |w_i| \leq d\}$. To simplify the counting, we let $L = (2|\mathbf{q}| - 1)^2$ and $M = |W_d^t| \leq (2|\mathcal{T}|)^d$. The vertices of the hypergraph $H_{\mathbf{q},\mathcal{T}}$ correspond to the edges of T', and there can be at most $L \cdot (2M + 1)$ of them, since there can be no more than L edges in T, and each is replaced by $2M + 1$ new edges. The hyperedges of $H_{\mathbf{q},\mathcal{T}}$ are of two types: E_i^k (where $1 \leq i \leq L$ and $1 \leq k \leq M$) and E_{ij}^{km} (where $1 \leq i \leq L$ and $1 \leq k \leq M$). It follows that the total number of hyperedges cannot exceed $L(M + M^2)$. Finally, a simple examination of the labelling function shows that there can be at most $(2|\mathcal{T}| + |\mathbf{q}| + 1)|\mathbf{q}|$ conjuncts in each label. $\square$

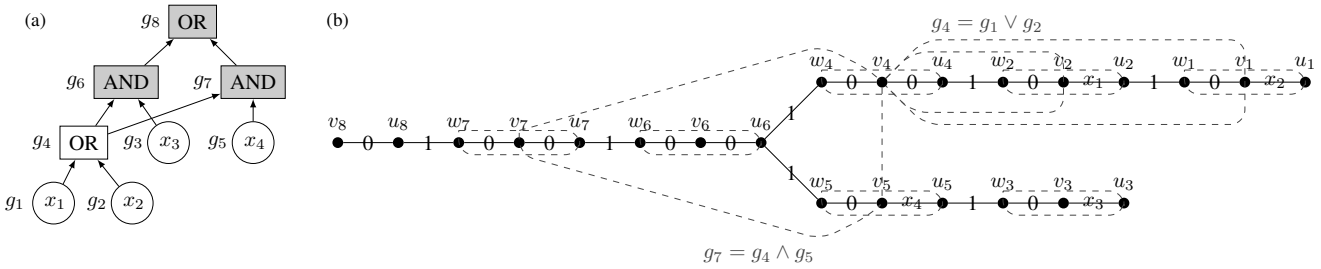


Fig. 7. (a) Example circuit, with nodes from S_1 in white, nodes from S_2 in grey (b) the tree underlying the corresponding THGP, together with labels and some hyperedges

Theorem 10. There exist polynomials p, p' such that:

- Every function computed by a semi-unbounded fan-in circuit of size at most σ and depth at most $\log \sigma$ is computable by a THGP of size $p(\sigma)$.
- Every function computed by a THGP of size σ is computable by a semi-unbounded fan-in circuit of size at most $p'(\sigma)$ and depth at most $\log p'(\sigma)$.

Both reductions preserve monotonicity.

Proof of the First Statement of Theorem 10: THGPs can simulate SAC^1 *circuits*

Consider a semi-unbounded fan-in circuit $\boldsymbol{C}$ of size at most σ and depth at most $\log \sigma$. Denote its gates by $g_1, \ldots, g_\sigma$, where g_σ is the output gate. We define the AND-depth of gates of $\boldsymbol{C}$ inductively. For the AND gate g_i, its AND-depth, $d(g_i)$, equals 1 if on each path from g_i to the input there are no AND gates. If there are AND gates on the paths from g_i to the input, consider one such gate g_j with maximal AND-depth, and let $d(g_i) = d(g_j) + 1$. For an OR gate g_i, we let $d(g_i)$ to be equal to the largest AND-depth of an AND gate on some path from g_i to the input. If there are no such AND gates, then the AND-depth of g_i is 0.

We denote by S_i the set of AND gates of the circuit of AND-depth i. Note that since the depth of $\boldsymbol{C}$ is at most $\log \sigma$, we have that the AND-depth of its gates is also at most $\log \sigma$. For each AND gate g_i of the circuit, we distinguish its first and its second input. We denote by $\mathsf{Left}(g_i)$ the subcircuit computing the first input of g_i, that is, the subcircuit consisting of the left input g_j of g_i and of all gates such that there is a path from them to g_j. Analogously, we use $\mathsf{Right}(g_i)$ to denote the subcircuit that computes the second input of g_i.

Lemma 25. *Any semi-unbounded fan-in circuit $\boldsymbol{C}$ of size σ and depth d is equivalent to a semi-unbounded fan-in circuit of size $2^d\sigma$ and depth d such that for each i*

$$\left(\bigcup_{g \in S_i} \mathsf{Left}(g)\right) \bigcap \left(\bigcup_{g \in S_i} \mathsf{Right}(g)\right) = \emptyset.$$

The proof of this lemma is standard, but we include it for the sake of completeness.

Proof. We show by induction on j that we can reconstruct the circuit in such a way that the property holds for all $i \leq j$, the depth of the circuit does not change, and the size of the circuit increases at most by the factor of 2^j.

Consider all AND gates in S_j, and consider a subcircuit $\bigcup_{g \in S_j} \mathsf{Left}(g)$. Construct a copy $\boldsymbol{C}'$ of this subcircuit separately and feed its output as first inputs to AND gates in S_j. This at most doubles the size of the circuit and ensures the property for S_j. Now for both circuits $\boldsymbol{C}'$ and $\bigcup_{g \in S_j} \mathsf{Right}(g)$ apply the induction hypothesis (note that the circuits do not intersect). The size of both circuits will increase at most by the factor of 2^{j-1} and the property for S_i for $i < j$ will be ensured. □

We can thus assume without loss of generality that the circuit $\boldsymbol{C}$ satisfies the property from the preceding lemma.

We now proceed to the construction of the THGP. We will begin by constructing a tree T and then afterwards define a hypergraph program based upon this tree. For each gate g_i in $\boldsymbol{C}$, we introduce three vertices w_i, v_i, u_i, and we arrange all these vertices into the tree from output gate to inputs. We construct the tree inductively from the root to leaves (see Figure 7). First we arrange vertices corresponding to the gates of AND-depth d into a path. Vertices are ordered according to the order of gates in $\boldsymbol{C}$. In each triple of vertices, the u-vertex preceds the v-vertex, which precedes the w-vertex. ‘ Next we branch the tree into two branches at the last u-vertex and associate subcircuit $\bigcup_{g \in S_d} \mathsf{Left}(g)$ to the left branch and the subcircuit of all other vertices to the right branch. We repeat the process for each subcircuit. This results in a tree, the number of vertices of which is 3σ. We remove from this tree the vertex w_σ.

We now define a hypergraph program based upon this tree. As before, the vertices of the hypergraph are the edges of the tree, and the hyperedges will take the form of generalized intervals. For each $i \neq \sigma$, we introduce a hyperedge $\langle w_i, u_i \rangle$. For each ANG gate g_i with $g_i = g_j \wedge g_k$, we add a hyperedge $\langle v_j, v_k, v_i \rangle$. For each OR gate $g_i = g_{k_1} \vee \ldots \vee g_{k_l}$, we add hyperedges $\langle v_{k_1}, v_i \rangle, \ldots, \langle v_{k_l}, v_i \rangle$.

For input gates, we label the corresponding $\{u, v\}$-edges by the corresponding literals (recall that in the circuit $\boldsymbol{C}$ negations are applied only to the inputs, so in this construction, we assume that the inputs are variables and their negations, and there are no NOT gates in the circuit). We label all other $\{u, v\}$-edges and $\{v, w\}$-edges of the tree by 0, and all remaining edges are labelled by 1.

The preceding construction clearly yields a THGP of size polynomial in the original circuit, and the construction is monotonicity-preserving. To complete the proof of the first statement of Theorem 10, we must show that the constructed THGP computes the same function as the circuit. This is established by the following claim:

Claim. For a given input x and for any i, the gate g_i outputs 1 iff the subtree with the root v_i can be covered (i.e. there is an independent subset of hyperedges that lies inside the subtree and covers all of the zeros in the subtree).

Proof. We prove the claim by induction on i. For input gates, the claim is trivial. If g_i is an AND gate, then both its inputs output 1. We cover both subtrees corresponding to the inputs (by induction hypothesis) and add a hyperedge $\langle v_j, v_k, v_i \rangle$. This covers the subtree rooted in v_i. If g_i is an OR gate, then there exists an input g_{k_j} of g_i which outputs 1. By the induction hypothesis, we can find a cover of its subtree and then add a hyperedge $\langle v_{k_j}, v_i \rangle$. All other edges of the subtree rooted in v_i can be covered by hyperedges of the form $\{u_p, w_p\}$. □

Proof of the Second Statement of Theorem 10: SAC^1 *circuits can simulate THGPs*

Now we proceed to the second part of Theorem 10. Suppose P is a THGP of size σ, and denote by T its underlying tree. We aim to construct a semi-unbounded fan-in circuit of size polynomial in σ. We first describe the idea of the construction, then do some preliminary work, and finally, detail the construction of the circuit.

First of all, we note that it is not convenient to think about covering all zero vertices of the hypergraph, and it is more convenient to think about partitioning the set of all vertices into disjoint hyperedges. To switch to this setting, for each vertex e of the hypergraph (recall it is an edge of T), we introduce a hyperedge $\{e\}$. Thus we arrive at the following problem:

(prob) given a tree hypergraph $H = (V_H, E_H)$, a labelling of its hyperedges $\mathfrak{l}$ and an input γ, decide if V_H can be partitioned into disjoint hyperedges, whose labels are evaluated into 1 under γ.

Before we proceed, we need to introduce some notation related to the trees. A vertex of a tree T is called a *branching point* if it has degree at least 3. A *branch* of the tree T is a simple path between two branching points which does not contain any other branching points. If v_1, v_2 are vertices of T we denote by T_{v_1,v_2} the subtree of T lying between the vertices v_1 and v_2. If v is a vertex of degree k with adjacent edges $e_1, \ldots, e_k$ then it splits T into k vertex-disjoint subtrees which we denote by $T_{v,e_1}, \ldots, T_{v,e_k}$. We call a vertex of a subtree T_1 a *boundary point* if it has a neighbour in T outside of T_1. The edges of T_1 adjacent to boundary points are called *boundary edges* of T_1. The *degree* of a subtree T_1 is the number of its boundary points. Note that there is only one subtree of T of degree 0 – the tree T itself.

Before we proceed with the proof of the theorem, we show the following technical lemma concerning the structure of tree hypergraph programs.

Lemma 26. *For any tree hypergraph program H with underlying tree T, there is an equivalent tree hypergraph program H' with underlying tree T' such that each hyperedge of H' covers at most one branching point of T', and the size of H' is at most $p(|H|)$ for some explicit polynomial p.*

Sketch. Let $h_1, \ldots, h_l$ be the hyperedges of H containing more than 2 branching points. Let $bp_1, \ldots, bp_l$ be the number of branching points in them. We prove the lemma by induction on $bp = \sum_i bp_i$. The base case is when this sum is 0.

For the induction step, consider h_1 and let v be one of the branching points of T in h_1. Denote by $e_1, \ldots, e_k$ the edges adjacent to v in T. On each e_i near vertex v, introduce two new adjacent edges e_{i1}, e_{i2}, edge e_{i1} closer to v, and label them by 0. Let v_i be the new vertex lying between e_{i1} and e_{i2}. Break h into $k+1$ hyperedges by vertices v_i and substitute h by these new hyperedges. Add hyperedges $\{e_{i1}, e_{i2}\}$ for all i. It is not hard to see that for each evaluation of variables there is a cover of all zeros in the original hypergraph iff there is a cover of all zeros in the new hypergraph. It is also not hard to see that bp has decreased during this operation and the size of the hypergraph program has increased by at most $2|T|$. So the lemma follows. □

Thus, in what follows, we can assume that each hyperedge of the hypergraph contains at most one branching point.

Now we are ready to proceed with the proof of the theorem. The main idea of the computation generalizes the polynomial size logarithmic depth semi-unbounded fan-in circuit for directed connectivity problem (discussed below).

In what follows, we say that some subtree T' of T can be partitioned into disjoint hyperedges if there is a set of disjoint hyperedges $h_1, h_2, \ldots, h_k$ in H such that they all lie in T', $\mathfrak{l}(h_i) = 1$ under γ for $1 \leq i \leq k$, and their union contains all edges of T'. Fix γ. Given the tree T underlying the hypergraph, we say that its vertices v_1 and v_2 are reachable from each other if the subtree lying between them can be partitioned into disjoint hyperedges. In this case, we let $\mathsf{Reach}(v_1, v_2) = 1$, otherwise we let $\mathsf{Reach}(v_1, v_2) = 0$. If v is a vertex of T and $e = \{v, u\}$ is an edge adjacent to it, we say that v is reachable from the side of e if the subtree $T_{v,e}$ can be partition into disjoint hyperedges. In this case, we let $\mathsf{Reach}(v, e) = 1$, otherwise we let $\mathsf{Reach}(v, e) = 0$. Our circuit will gradually compute the reachability relation Reach for more and more vertices, and in the end, we will compute whether the whole tree can be partitioned into hyperedges.

First, our circuit will compute the reachability relation for vertices on each branch of the tree T. If one of the endpoints of the branch is a leaf, we compute the reachability for the remaining vertex from the side containing the leaf. This is done just like for the usual reachability problem.

Next we proceed to compute reachability between vertices on different branches of T. For this, consider a tree D those vertices are branching points and leaves of the original tree T and those edges are branches of T. In D, each vertex is either a leaf or a branching point. We will consider subtrees of the tree D.

We describe a process of partitioning D into subtrees. At the end of the process, all subtrees will be individual edges. We have the following basic operation. Assume that we have already constructed a subtree D', consider some vertex $v \in D'$ and assume that it has k outgoing edges $e_1, \ldots, e_k$ within D', $e_i = \{v, v_i\}$. By partitioning D' in the vertex v, we call a substitution of D' by a set of disjoint subtrees $D_1, \ldots, D_k \subseteq D'$, where for all i, we let $D_i = D_{v,e} \cap D'$.

The following lemma helps us to apply our basic operation efficiently.

Lemma 27. *Consider a subtree D' of size m. If its degree is ≤ 1, then there is $v \in D'$ partitioning it into subtrees of size at most $m/2 + 1$ and degree at most 2 each. If the degree of D' is 2, then there is $v \in D'$ partitioning it into subtrees of size at most $m/2 + 1$ and degree at most 2 and possibly one subtree of size less than m and degree 1.*

Proof. If D' is of degree ≤ 1, then consider its arbitrary vertex v_1 and subtrees into which this vertex divides D'. If among them there is a subtree D_1 larger than $m/2 + 1$, then consider the (unique) vertex v_2 in this subtree adjacent to v_1. If we separate D' by v_2, then this partition will consist of the tree $D' \setminus D_1 \cup \{v_1, v_2\}$ and other trees lying inside of D_1. Thus, D_1 will be of size at most $m/2$ and other subtrees will be of size smaller than $|D_1|$. Thus, the size of the largest subtree decreased, and we repeat the process until the size of the largest subtree becomes at most $m/2 + 1$.

If D' has degree 2, consider its boundary points b_1 and b_2. Repeat the same process starting with $v_1 = b_1$. Once in this process the current vertex v tries to leave a path between b_1 and b_2, we stop. For this vertex v, it is not hard to see that all the resulting trees are of degree at most 2, and the only tree having the size larger than $m/2$ is of degree 1. □

With this lemma, the partitioning process works as follows. We start with the partition $\{D\}$ consisting of the tree D itself and repeatedly partition the current set of subtrees into smaller ones. At each step, we repeat the described procedure for each subtree separately. Note that after two steps the size of the largest subset decreases by the factor of 2. Thus in $O(\log \sigma)$ steps, we obtain the partition consisting of individual edges.

Now we are ready to describe the computational process of the circuit. The circuit will consider tree partitions described above in the reversed order. That is, we first have subtrees consisting of individual edges. Then on each step we merge some of them. In the end, we obtain the whole tree.

The intuition is that along with the construction of a subtree D_1, we compute the reachability for its boundary edges, that is, for example if the boundary edges of D_1 are b_1 and b_2 then we compute the reachability relation $\mathsf{Reach}(v_1, v_2)$ for all v_1 lying on the branch b_1 in T and v_2 lying on the branch b_2 in T.

Now we are ready to describe the circuit. First for each branch of the tree the circuit computes the reachability matrix for that branch. This is done by squaring the adjacency matrix $O(\log \sigma)$ times for all branches in parallel. Note that squaring a matrix requires only bounded fan-in AND-gates, and thus this step is readily computable by semi-unbounded circuit of size polynomial in σ and depth logarithmic in σ.

Thus the circuit computes the reachability matrix for the initial partition of D. Next the circuit computes the reachability matrix for larger subtrees of D following the process above. More specifically, suppose we merge subtrees $D_1, \ldots, D_k$ meeting in the vertex u to obtain a tree D'. For simplicity of notation, assume that there are two subtrees D_1 and D_2 having degree 2, denote by b, b' the boundary edges of D' and by $b_1, \ldots, b_k$ the boundary edges of $D_1, \ldots, D_k$ respectively adjacent to the vertex u.

It is not hard to see that for all vertices v in b and v' in b', it is true that

$$\mathsf{Reach}(v, v') = \bigvee_{h \ni u} \left(\mathsf{Reach}(v, v_1) \wedge \mathsf{Reach}(v', v_2) \wedge \bigwedge_{i=3}^{k} \mathsf{Reach}(v_i, e_i) \wedge \mathfrak{l}(h) \right),$$

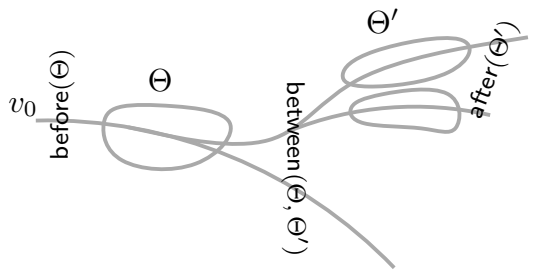


Fig. 8. In the above diagram, Θ consists of one tree witness, and Θ' consists of two tree witnesses. Both are independent and flat, and Θ precedes Θ'. Here $\mathsf{before}(\Theta)$ is the segment between v_0 and Θ, $\mathsf{between}(\Theta, \Theta')$ comprises the segment between Θ and Θ' as well as the downwards branch that exits Θ, and $\mathsf{after}(\Theta')$ consists of the two branches that leave Θ' on the side furthest from v_0.

where h ranges over all hyperedges of our hypergraph inside D, $v_1, \dots, v_k$ are boundary vertices of h lying in the branches $b_1, \dots, b_k$ respectively, for each i e_i is an edge adjacent to v_i and not contained in h.

The case when only one subtree among $D_1, \dots, D_k$ has degree 2 is analogous.

Thus we have described the circuit for solving (prob). Clearly that it is monotone provided that P is monotone. It is easy to see that its size is bounded by some fixed polynomial in σ. It is only left to show that this circuit can be arranged in such a way that it has depth $O(\log \sigma)$. This is not trivial since we have to show how to compute big AND in the formula above to make depth logarithmic. For this, we will use the following lemma.

Lemma 28. *Suppose the reachability relation for each branch is already computed. Then the subtree D' with m edges constructed on step i can be computed in the* AND*-depth at most* $\log m + i$.

Proof. The proof proceeds by induction. Suppose that to construct D', we unite subtrees $D_1, \dots, D_k$ of sizes $m_1, \dots, m_k$ respectively. Note that $m = m_1 + \ldots + m_k$.

By the induction hypothesis, we can compute each subtree D_j having AND-depth at most $\log m_j + i - 1$.

Consider a k-letter alphabet $A = \{a_1, \dots, a_k\}$ and assign to each letter a_j the probability m_j/m. It is well-known that there is a prefix binary code for this alphabet such that each letter a_j is encoded by a word of length $\lceil \log(m/m_j) \rceil$. This encoding can be represented by a rooted binary tree the leaves of which are labeled by letters of A and the length of the path from root to the leaf labeled by a_j is equal to $\lceil \log(m/m_j) \rceil$. Assigning the AND function to each vertex of the tree, we obtain the computation of AND in the formula above. The depth of this computation is the maximum over j of

$$\log m_j + (i-1) + \lceil \log(m/m_j) \rceil \leq \log m_j + (i-1) + \log(m/m_j) + 1 = \log m + i.$$

□

From this lemma and the fact that the computation stops after $O(\log \sigma)$ steps, we obtain that overall the AND-depth of the circuit is $O(\log \sigma)$ and thus the overall depth is $O(\log \sigma)$. This completes the proof of Theorem 10.

Theorem 11. Fix $\ell \geq 2$. For every ontology $\mathcal{T}$ and CQ $\mathbf{q}$ with at most ℓ leaves, the function $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$ is computable by a monotone NBP of size polynomial in $|\mathbf{q}|$ and $|\mathcal{T}|$.

Proof (continued). Recall that in the main text we chose a root variable v_0 in the query $\mathbf{q}$. We then defined flat sets Θ of tree witnesses, by requiring that every simple path starting from v_0 intersect at most one tree witness of Θ, and showed how flat sets could be ordered by the precedence relation $\prec$. This led us to construct the graph $G_P = (V_P, E_P)$ with $V_P = \{u_\Theta, v_\Theta \mid \text{ flat } \Theta \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}}\} \cup \{s, t\}$ and $E_P = \{(s, u_\Theta), (v_\Theta, t), (u_\Theta, v_\Theta) \mid \text{ flat } \Theta\} \cup \{(v_\Theta, u_{\Theta'}) \mid \text{ flat } \Theta \prec \Theta'\}$.

To formally define the labelling of the edges of G_P, we must first introduce some notation. For flat $\Theta \prec \Theta'$, we denote by $\mathsf{between}(\Theta, \Theta')$ the conjunction of p_η for atoms η 'between' Θ and Θ', that is those that lie outside of Θ and Θ' and are accessible from Θ via paths not passing through Θ' but are not accessible from v_0 via a path not passing through Θ. For flat Θ, we denote by $\mathsf{before}(\Theta)$ the conjunction of p_η for query atoms η which lie outside of Θ and are accessible from v_0 via paths not passing through Θ. By $\mathsf{after}(\Theta)$, we denote the conjunction of p_η for atoms η outside Θ which are accessible from v_0 only via paths passing through Θ. Now we are ready to define the labelling:

- edges of the form (u_Θ, v_Θ) are labelled $\bigwedge_{\mathfrak{t} \in \Theta} p_{\mathfrak{t}}$;
- edges of the form (s, u_Θ) are labelled with $\mathsf{before}(\Theta)$;
- edges of the form $(v_\Theta, u_{\Theta'})$ for $\Theta \prec \Theta'$ are labelled with $\mathsf{between}(\Theta, \Theta')$;
- edges of the form (v_Θ, t) are labelled with $\mathsf{after}(\Theta)$.

We claim that under any valuation of $p_{\mathfrak{t}}$ and p_η, the vertex t is accessible from s if and only if there is an independent subset $\hat{\Theta} \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}}$ (not necessarily flat) such that $p_{\mathfrak{t}} = 1$ for all $\mathfrak{t} \in \hat{\Theta}$ and $p_\eta = 1$ for all atoms η outside $\mathbf{q}_{\hat{\Theta}}$. Indeed, any such $\hat{\Theta}$ splits into flat "layers" $\Theta^1, \Theta^2, \dots \Theta^m$ which form a path $s \to u_{\Theta^1} \to v_{\Theta^1} \to u_{\Theta^2} \to \dots \to v_{\Theta^m} \to t$ in G_P and whose edge labels evaluate to 1: take Θ^1 to be the set of all edges from $\hat{\Theta}$ that are accessible from v_0 via paths which do not cross (that is come in and go out) any hyperedge of $\hat{\Theta}$; take Θ^2 to be the set of all edges from $\hat{\Theta} \setminus \Theta^1$ which are accessible from v_0 via paths which do not cross any hyperedge of $\hat{\Theta} \setminus \Theta^1$, and so on. Conversely, any path leading from s to t gives us a covering $\hat{\Theta}$ which is the union of all flat sets that occur in the subscripts of vertices on this path. □

Theorem 13. There is a sequence of linear CQs $\mathbf{q}_n$ and ontologies $\mathcal{T}_n$ of depth 2, both of polysize in n, such that any PE-rewriting of $\mathbf{q}_n$ and $\mathcal{T}_n$ is of size $n^{\Omega(\log n)}$.

Proof. It is known that there is a sequence f_n of monotone Boolean functions that are computable by polynomial-size monotone NBPs, but all monotone Boolean formulas computing f_n are of size $n^{\Omega(\log n)}$, e.g., s-t-reachability in a directed graph [30].

Apply Theorem 12 to the sequence f_n mentioned above to obtain a sequence of interval hypergraph programs P_n based on interval hypergraphs H_n which compute the functions f_n. By Theorem 8, there exist CQs $\mathbf{q}_n$ and ontologies $\mathcal{T}_n$ of depth 2 such that $f_n(\alpha) = 1$ iff $f^{\mathsf{prim}}_{\mathbf{q}_n,\mathcal{T}_n}(\gamma) = 1$, where γ is defined as follows: $\gamma(B_e) = 1$, $\gamma(R_e) = \gamma(R'_e) = 0$, and $\gamma(S_{ij}) = \gamma(S'_{ij}) = \alpha(\mathfrak{l}_P(\{v_i, v_j\}))$. We know from the construction that $\mathbf{q}_n$ is a linear CQ and $\mathbf{q}_n$ and $\mathcal{T}_n$ are both of polynomial size in n. Since f_n is obtained from $f^{\mathsf{prim}}_{\mathbf{q}_n,\mathcal{T}_n}$ through a simple substitution, the lower bound $n^{\Omega(\log n)}$ still holds for $f^{\mathsf{prim}}_{\mathbf{q}_n,\mathcal{T}_n}$. It remains to apply Theorem 6 to transfer this lower bound to PE-rewritings of $\mathbf{q}_n$ and $\mathcal{T}_n$. □

Theorem 14. Fix a constant $\ell > 1$. Then all tree-shaped CQs with at most ℓ leaves and arbitrary ontologies have polynomial-size NDL-rewritings.

Proof. Fix $\ell > 1$. By Theorem 11, there exists a polynomial p such that for every tree-shaped CQ $\mathbf{q}$ with at most ℓ leaves and every ontology $\mathcal{T}$, there is a monotone NBP of size at most $p(|q| + |\mathcal{T}|)$ that computes $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$. We also know from [29] that there is a polynomial p' such that every function f_P given by a monotone NBP P can be computed by a monotone Boolean circuit $\boldsymbol{C}_P$ of size at most $p'(P)$. By composing these two translations, we obtain polysize monotone Boolean circuits that compute the functions $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$, for the class of tree-shaped CQs with at most ℓ leaves. It then remains to apply Theorem 4. □

Theorem 15. The following are equivalent:

1) There exist polysize FO-rewritings for all linear CQs and depth 2 ontologies;
2) There exist polysize FO-rewritings for all tree-shaped CQs with at most ℓ leaves and arbitrary ontologies (for any fixed ℓ);
3) There exists a polynomial function p such that every NBP of size at most s is computable by a formula of size $p(s)$. Equivalently, NL/poly $\subseteq$ NC^1.

Proof.
(2) $\Longrightarrow$ (1): Trivial.

(1) $\Longrightarrow$ (3): Suppose (1) holds. In other words, there exists a polynomial p such that any linear query $\mathbf{q}$ and an ontology $\mathcal{T}$ of depth 2 have a rewriting of the size $p(|\mathbf{q}| + |\mathcal{T}|)$. Consider a sequence of functions f_n computing s-t-reachability in directed graphs, which is known to be NL/poly-complete under NC^1-reductions [35] (This function takes the adjacency matrix of an undirected graph G on n vertices with two distinguished vertices s and t and returns 1 iff t is accessible from s in G.) Clearly, the functions f_n are computed by a sequence of polynomial-size NBPs P_n. Theorem 12 gives us a sequence of hypergraph programs P'_n which compute the f_n. By Theorem 8, there exist CQs $\mathbf{q}_n$ and ontologies $\mathcal{T}_n$ such that $f_n(\alpha) = 1$ iff $f^{\mathsf{prim}}_{\mathbf{q}_n,\mathcal{T}_n}(\gamma) = 1$, for the valuation γ defined as follows: $\gamma(B_e) = 1, \gamma(R_e) = \gamma(R'_e) = 0$, and $\gamma(S_{ij}) = \gamma(S'_{ij}) = \alpha(\mathfrak{l}_P(\{v_i, v_j\}))$. By assumption, they have PE-rewritings $\mathbf{q}'_n$ of size $p(|\mathbf{q}| + |\mathcal{T}|)$ which is polynomial in n. Theorem 6 gives us a polysize Boolean formula for computing $f^{\mathsf{prim}}_{\mathbf{q}_n,\mathcal{T}_n}$. Since f^n is obtained from $f^{\mathsf{prim}}_{\mathbf{q}_n,\mathcal{T}_n}$ by some substitution, it follows that we have a polysize formula for f_n, hence for all functions in NL/poly.

(3) $\Longrightarrow$ (2): Suppose (3) holds. Fix some $\ell > 1$. Take a tree-shaped query $\mathbf{q}$ with at most ℓ leaves and an ontology $\mathcal{T}$. Since ℓ is fixed, by Theorem 11, there is a polysize NBP P which computes $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$. By assumption, there is a polysize FO formula computing $f^{\mathsf{tw}}_{\mathbf{q},\mathcal{T}}$, and Theorem 4 transforms it into a FO-rewriting of $\mathbf{q}$ and $\mathcal{T}$. □

Theorem 16. Fix constants $t > 0$ and $d > 0$. Then all CQs of treewidth $\leq t$ and ontologies of depth $\leq d$ have polysize NDL-rewritings.

Proof. Fix constants $t > 0$ and $d > 0$. By Theorem 9, we have that there is a polynomial p' such that for any CQ $\mathbf{q}$ of treewidth at most t and any ontology $\mathcal{T}$ of depth at most d the THGP P computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ and is of size at most $p'(|\mathbf{q}| + |\mathcal{T}|)$. Now we apply Theorem 10 and conclude that $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$ may be computed by a polysize semi-unbounded fan-in circuit. Therefore, by Theorem 5, there exists a polysize NDL-rewriting for $\mathbf{q}$ and $\mathcal{T}$. □

Theorem 17. The following are equivalent:

1) There exist polysize FO-rewritings for all tree-shaped CQs and depth 2 ontologies;
2) There exist polysize FO-rewritings for all CQs of treewidth at most t and ontologies of depth at most d (for fixed constants $t > 0$ and $d > 0$);
3) There exists a polynomial function p such that every semi-unbounded fan-in circuit of size at most σ and depth at most $\log \sigma$ is computable by a formula of size $p(\sigma)$. Equivalently, $\mathrm{SAC}^1 \subseteq \mathrm{NC}^1$.

Proof.
(2) $\Longrightarrow$ (1): Trivial.

(1) $\Longrightarrow$ (3): Suppose (1) holds. In other words, there exists a polynomial p'' such that every tree-shaped query $\mathbf{q}$ and ontology $\mathcal{T}$ of depth 2 has a rewriting of the size $p''(|\mathbf{q}| + |\mathcal{T}|)$. Consider a semi-unbounded fan-in circuit $\boldsymbol{C}$ of size σ and depth at most $\log \sigma$ that computes the Boolean function f. By Theorem 10, f is computed by a THGP P based on a tree hypergraph H of size at most $p(\sigma)$ for the polynomial p from Theorem 10. By Theorem 8, there exists a tree-shaped query $\mathbf{q}_P$ and an ontology $\mathcal{T}_P$ of depth 2 such that f is straightforwardly obtained from $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ via substitution. By the assumption, there exists an FO-rewriting for $\mathbf{q}_P$ and $\mathcal{T}_P$ of size at most $p''(|\mathbf{q}| + |\mathcal{T}|)$. This number is polynomial in σ (take the composition of p, the polynomial function from Theorem 8 and p''). Now by Theorem 6, there exists a polysize first-order formula for computing $f^{\mathsf{prim}}_{\mathbf{q},\mathcal{T}}$ and hence also for f.

(3) $\Longrightarrow$ (2): Suppose (3) holds. Fix $t > 0$ and $d > 0$. Take query $\mathbf{q}$ of treewidth at most t and an ontology $\mathcal{T}$ of depth at most d. Since t is fixed, by Theorem 9, there is a polysize THGP P that computes $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$. By assumption and Theorem 10, there is a polysize FO-formula computing $f^{\mathsf{tw}'}_{\mathbf{q},\mathcal{T}}$. We can then apply Theorem 5 to transform it into an FO-rewriting of $\mathbf{q}$ and $\mathcal{T}$. □

Theorem 18. Fix $t > 0$. Then there exist polysize PE-rewritings for all CQs of treewidth at most t and depth 1 ontologies.

Proof. Fix a constant $t > 0$. Throughout the proof, we will consider CQs of treewidth at most t, and for every such query $\mathbf{q}$, we will use $\mathsf{mtd}_t(\mathbf{q})$ (for 'minimal tree decomposition') to denote the minimum number of vertices over all tree decomposition of $\mathbf{q}$ that have width at most t.

We also fix an ontology $\mathcal{T}$ of depth 1, and as before, we use $\Theta^{\mathbf{q}}_{\mathcal{T}}$ to denote the set of all tree witnesses for the query $\mathbf{q}$ and ontology $\mathcal{T}$. Since $\mathcal{T}$ has depth 1, it is known from [13] that every tree witness $\mathfrak{t} = (\mathfrak{t}_r, \mathfrak{t}_i) \in \Theta^{\mathbf{q}}_{\mathcal{T}}$ contains a unique interior point (i.e. $|\mathfrak{t}_i| = 1$), and no two distinct tree witnesses may share the same interior point.

Given a set $U \subseteq \mathsf{vars}(\mathbf{q})$ of variables, we will use $\Theta^{\mathbf{q}}_{\mathcal{T}}(U)$ to refer to the tree witnesses whose interior point belongs to U. If Ω is an independent subset of $\Theta^{\mathbf{q}}_{\mathcal{T}}(U)$, then the set $\mathsf{border}(U, \Omega)$ of border variables for U and Ω is defined as follows:

$$\{u \in U \mid \text{ there is no } \mathfrak{t} \in \Omega \text{ with } \mathfrak{t}_i = \{u\}\} \cup \{z \mid z \in \mathfrak{t}_r \text{ for some } \mathfrak{t} \in \Omega\}.$$

We also define $\mathbf{q}^{U,\Omega} = \mathbf{q} \setminus \{\eta \in \mathbf{q} \mid \mathsf{vars}(\eta) \subseteq U \cup \mathsf{border}(U, \Omega)\}$. For $z, z' \in \mathsf{vars}(\mathbf{q}) \setminus (U \cup \mathsf{border}(U, \Omega))$, we set $z \sim z'$ if there is a path in $G_{\mathbf{q}}$ from z to z' that does not pass through $\mathsf{border}(U, \Omega)$ (recall that $G_{\mathbf{q}}$ is undirected). The $\sim$ relation can be lifted to the atoms in $\mathbf{q}^{U,\Omega}$ by setting $\eta_1 \sim \eta_2$ if all of the variables in $(\eta_1 \cup \eta_2) \setminus \mathsf{border}(U, \Omega)$ are $\sim$-equivalent. Let $\mathbf{q}^{U,\Omega}_1, \ldots, \mathbf{q}^{U,\Omega}_k$ denote the queries formed by the $\sim$-equivalence classes of atoms in $\mathbf{q}^{U,\Omega}$ with $\mathsf{avars}(\mathbf{q}^{U,\Omega}_i) = (\mathsf{avars}(\mathbf{q}) \cup \mathsf{border}(U, \Omega)) \cap \mathsf{vars}(\mathbf{q}^{U,\Omega}_i)$.

Claim. For every query $\mathbf{q}$ of treewidth t, there exists a subset U of $\mathsf{vars}(\mathbf{q})$ such that $|U| \leq t + 1$ and for all subqueries $\mathbf{q}^{U,\Omega}_i$, we have $\mathsf{mtd}_t(\mathbf{q}^{U,\Omega}_i) < \mathsf{mtd}_t(\mathbf{q})/2$.

Proof of claim. Consider some tree decomposition (T, λ) of $\mathbf{q}$ of width t with $T = (V, E)$ and $|V| = \mathsf{mtd}_t(\mathbf{q})$. It was shown in [13] that there exists a vertex $v \in V$ such that each connected component in the graph T_v obtained by removing v from T has at most $|V|/2 = \mathsf{mtd}_t(\mathbf{q})/2$ vertices. Consider the set of variables $U = \lambda(v)$. Since (T, λ) has width t, we have that $|U| \leq t + 1$. As to the second property, we observe that it is sufficient to consider queries of the form $\mathbf{q}^{U,\emptyset}_i$, since by definition, $\mathbf{q}^{U,\Omega}_i \subseteq \mathbf{q}^{U,\emptyset}_i$ for any $\Omega \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}}(U)$. We then remark that due to the connectedness condition on tree decompositions, and the fact that we only consider paths in $G_{\mathbf{q}}$ that do not pass by variables in $U = \lambda(v)$, every query $\mathbf{q}^{U,\emptyset}_i$ can be obtained by:

1) taking a connected component $C_i = (V_i, E_i)$ in the graph T_v;
2) considering the resulting tree decomposition (C_i, λ_i), where λ_i is the restriction of λ to the vertices in C_i;
3) taking a subset of the atoms in $\{\eta \in \mathbf{q} \mid \mathsf{vars}(\eta) \subseteq \lambda(v') \text{ for some } v' \in V_i\}$.

It then suffices to recall that each connected component of T_v contains at most $\mathsf{mtd}_t(\mathbf{q})/2$ vertices. (*end proof of claim*)

We now use the claim to define a recursive rewriting procedure. Given a query $\mathbf{q}$ of treewidth at most t, we choose a set U of variables that satisfies the properties of preceding claim, and define the rewriting $\mathbf{q}^\dagger$ of $\mathbf{q}$ w.r.t. $\mathcal{T}$ as follows:

$$\mathbf{q}^\dagger = \exists \mathbf{y} \bigvee_{\substack{\Omega \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}}(U) \\ \text{independent}}} \left(\mathsf{at}(U,\Omega) \wedge \mathsf{tw}(U,\Omega) \wedge \bigwedge_{i=1}^{k} (\mathbf{q}_i^{U,\Omega})^\dagger \right)$$

where

- $\mathbf{y}$ is the set of all existential variables in U,
- $\mathsf{at}(U,\Omega) = \{\eta \in \mathbf{q} \mid \mathsf{vars}(\eta) \subseteq U \text{ and there is no } \mathfrak{t} \in \Omega \text{ with } \eta \in \mathbf{q}_\mathfrak{t}\}$;
- $$\mathsf{tw}(U,\Omega) = \bigwedge_{\mathfrak{t} \in \Omega} \left(\exists z' (\bigvee_{\substack{\mathfrak{t} \in \Theta^{\mathbf{q}}_{\mathcal{T}}[\varrho] \\ \varrho \in \mathsf{N}_2^{\pm}}} \rho_\varrho(z')) \wedge \bigwedge_{z \in \mathfrak{t}_\mathsf{r}} (z = z') \right);$$
- $(\mathbf{q}_i^{U,\Omega})^\dagger$ are rewritings of the queries $\mathbf{q}_i^{U,\Omega}$, which are constructed recursively according to the same procedure.

The †-rewriting we have just presented generalizes the rewriting procedure for tree-shaped queries from [13], and correctness can be shown similarly to the original procedure.

As to the size of the obtained rewriting, we remark that since the set U is always chosen according to the claim, and $\mathsf{mtd}_t(\mathbf{q}) \leq (2|\mathbf{q}| - 1)^2$ (cf. [34]), we know that the depth of the recursion is logarithmic in $|\mathbf{q}|$. We also know that the branching is at most 2^{t+1} at each step, since there is at most one recursive call for each subset $\Omega \subseteq \Theta^{\mathbf{q}}_{\mathcal{T}}(U)$, and since $\mathcal{T}$ has depth 1, we have $\Theta^{\mathbf{q}}_{\mathcal{T}}(U) \leq |U|$. Thus, the resulting formula $\mathbf{q}^\dagger$ has the structure of a tree whose number of nodes is bounded by $O((2^{t+1})^{\log |\mathbf{q}|}) = O(|\mathbf{q}|^{t+1})$.

□

PROOFS FOR SECTION IV

Proposition 23. *Every execution of BLQuery terminates. There exists an execution of BLQuery that returns* ***yes*** *on input* $(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b})$ *just in the case that* $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{b})$.

Proof. The following two claims, which can be easily seen to hold by examination of the procedure and straightforward induction, resume some important properties of BLQuery.

Claim 1. Every execution of BLQuery satisfies the following statements:

- Frontier always contains tuples (v_1, v_2, c, n) such that v_2 is a child of v_1 in T.
- Once (v_1, v_2, c, n) is added to Frontier, no other tuple of the form (v_1, v_2, c', n') may ever be added to Frontier in future iterations.
- At every iteration of the while loop, at least one tuple is removed from Frontier.
- If (v_1, v_2, c, n) is removed from Frontier, then either the procedure returns **no** or for every child v_3 of v_2, a tuple whose first two arguments are (v_2, v_3) is added to Frontier.

Claim 2. The while loop in Step 4 has the following loop invariants:

- Height is equal to number of symbols on Stack.
- If Height > 0, then all tuples (u, v, c, n) with $n > 0$ have the same c.
- All tuples (u, v, c, n) in Frontier are such that $n \leq$ Height, and there exists at least one tuple with $n =$ Height.

We now show the first statement of the proposition.

Claim 3. Every execution of BLQuery terminates.

Proof of claim. A simple examination of BLQuery shows that the only possible source of non-termination is the while loop in Step 4, which continues so long as Frontier is non-empty. It follows from the first statement of Claim 1 that the total number of tuples that may appear in Frontier at some point cannot exceed the number of edges in T, which is itself bounded above by $|\mathbf{q}|$. We also know from the second and third statements of Claim 1 that every tuple is added at most once and is eventually removed from Frontier . Thus, either we will exit the while loop by returning **no** (if one of the checks fails), or we will eventually exit the while loop after reaching an empty Frontier. (*end proof of claim*)

The next two claims establish the second half of the proposition.

Claim 4. If $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{b})$, then some execution of BLQuery$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b})$ returns **yes**.

Proof of claim. Suppose that $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{b})$. Then there exists a homomorphism $h : \mathbf{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(\mathsf{avars}(\mathbf{q})) = \mathbf{b}$, and without loss of generality we may choose h so that the image of h consists of elements aw with $|w| \leq 2|\mathcal{T}| + |\mathbf{q}|$. We use h to specify an execution of BLQuery$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b})$ that returns **yes**. In Step 1, we fix some arbitrary variable v_0 as root, and in Step 2, we let choose the element $h(v_0) = a_0 w_0$. Since h defines a homomorphism of $\mathbf{q}(\mathbf{b})$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, the call to MapCore or MapAnon will return **true**. In Steps 3, we will initialize Stack to w_0, Height to $|w_0|$, and Frontier to $\{(v_0, v_i, a_0, \mathsf{Height}) \mid v_i \text{ is a child of } v_0\}$. In Step 4, we enter the while loop. Our aim will be to make the non-deterministic choices in such a way as to satisfy the following invariant:

Inv If (v, v', c, m) is in Frontier and $w = \mathsf{Stack}[m]$, then $h(v) = cw$.

Recall that $\mathsf{Stack}[m]$ designates the word obtained by concatenating the first m symbols of Stack. Observe that at the start of Step 4, property **Inv** is satisfied. At the start of each iteration of the while loop, we proceed as follows:

Case 1 Frontier contains an element $\tau = (v_1, v_2, c, 0)$ such that $h(v_2) \in \mathsf{inds}(\mathcal{A})$. In this case, we will choose Option 1. In Step 4(a), we will remove τ from Frontier, and in 4(b), we guess the individual $h(v_2)$. As $c = h(v_1)$ (by **Inv**) and h is a homomorphism, the calls to MapCore and MapEdge will both return **true**. We will thus continue to 4(c) where we will add $(v_2, v_3, h(v_2), 0)$ to Frontier for every child v_3 of v_2. Note that these additions to Frontier preserve the invariant.

Case 2 Frontier contains $\tau = (v_1, v_2, c, \mathsf{Height})$ such that $h(v_2) = h(v_1)S$. In this case, we choose Option 2 and remove τ from Frontier in 4(d). Note that we must have $\mathsf{Height} < 2|\mathcal{T}| + |\mathbf{q}|$ since (i) by the invariant **Inv**, $h(v_1) = cw$ where $w = \mathsf{Stack}[\mathsf{Height}]$, and (ii) by our choice of the homomorphism h, we have that $wS \leq 2|\mathcal{T}| + |\mathbf{q}|$. We will thus continue on to Step 4(e), where we choose the role S. Because of the invariant, the fact that $h(v_2) = h(v_1)S$, and that h is a homomorphism, one can show that none of the (undesired) properties in 4(e) holds, and so we will continue to 4(f). First consider the case in which v_2 has some child. In this case, we push S onto Stack, increment Height, and add $(v_2, v_3, c, \mathsf{Height})$ to Frontier for every child v_3 of v_2. Observe that **Inv** holds for the newly added tuples and continues to hold for existing tuples. If v_2 is a leaf in T, then no additions are made to Frontier, but we pop δ symbols from Stack and decrement Height by δ, where δ is the difference between Height and the maximal current value appearing in any tuple of Frontier. Since there are no additions, and the relevant initial segment of Stack remains unchanged, **Inv** continues to hold.

Case 3 Neither Case 1 nor Case 2 holds. In this case, we choose Option 3, and remove all elements in $\mathsf{Deepest} = \{(v_1, v_2, c, n) \in \mathsf{Frontier} \mid n = \mathsf{Height}\}$ from Frontier. Since neither Case 1 nor Case 2 applies, $\mathsf{Height} > 0$. Thus, in Step 4(g), we will not return **no** and will instead pop the top symbol R from Stack and decrement Height by 1. Since $\mathsf{Height} > 0$, it follows from Claim 2 that all tuples in Deepest have the same individual c in third position. By the invariant **Inv**, for every tuple $(v_1, v_2, c, n) \in \mathsf{Deepest}$ is such that $h(v_1) = cwR$ where $wR = \mathsf{Stack}[\mathsf{Height}]$. Moreover, since Case 2 was not applicable, we know that for every such tuple (v_1, v_2, c, n), we have $h(v_2) = cw$. Using the fact that h is a homomorphism, we can show that none of the undesired properties in Step 4(h) holds, and so we will continue on to 4(i), where we will set $\mathsf{Children} = \{(v_2, v_3) \mid (v_1, v_2, c, n) \in \mathsf{Deepest}, v_3 \text{ is a child of } v_2\}$. If Children is non-empty, then we will add the tuple $(v_2, v_3, c, \mathsf{Height})$ to Frontier for each pair $(v_2, v_3) \in \mathsf{Children}$. Note that the invariant **Inv** is satisfied by all the new tuples. Moreover, since we only removed the last symbol in Stack, all the remaining tuples in Frontier will continue to satisfy **Inv**. If Children is empty, then we pop δ symbols from Stack and decrement Height by δ, where $\delta = \mathsf{Height} - \mathbf{max}\{\ell \mid (v, v', d, \ell) \in \mathsf{Frontier}\}$. We can use the same reasoning as in Option 2 to show that **Inv** continues to hold.

Since we have shown how to make the non-deterministic choices in the while loop without returning **no**, we will eventually leave the while loop (by Claim 3), and return **yes** in Step 5. (*end proof of claim*)

Claim 5. If some execution of BLQuery$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b})$ returns **yes**, then $\mathcal{T}, \mathcal{A} \models \mathbf{q}(\mathbf{b})$.

Proof of claim. Consider an execution of BLQuery$(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b})$ that returns **yes**. Since **yes** can only be returned in Step 5, it follows that the while loop was successfully exited after reaching an empty Frontier. Let L be the total number of iterations of the while loop. We inductively define a sequence $h_0, h_1, \ldots, h_L$ of partial functions from $\mathsf{vars}(\mathbf{q})$ to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ by considering the guesses made during the different iterations of the while loop. We will ensure that the following properties hold for every $0 \leq i < L$:

P1 If $i > 0$, then $\mathsf{dom}(h_{i_1}) \subseteq \mathsf{dom}(h_i)$, and if $v \in \mathsf{dom}(h_{i-1})$ is defined, then $h_i(v) = h_{i-1}(v)$.

P2 If tuple (v_1, v_2, c, n) belongs to Frontier at the beginning of iteration $i + 1$, then:

(a) $h_i(v_1) = cw$ where $w = \mathsf{Stack}[n]$ (recall that $\mathsf{Stack}[n]$ consists of the first n symbols of Stack)

(b) neither v_2 nor any of its descendants belongs to $\mathsf{dom}(h_i)$.

P3 h_i is a homomorphism from $\mathbf{q}_i$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, where $\mathbf{q}_i$ is the restriction of $\mathbf{q}$ to the variables in $\mathsf{dom}(h_i)$.

Note that above and in what follows, we use $\mathsf{dom}(h_i)$ to denote the domain of the partial function h_i.

We begin by setting $h_0(v_0) = u_0$ (and leaving h_0 undefined for all other variables). Property **P1** is not applicable. Property **P2**(a) holds because of the initial values of Frontier, Stack, and Height, and **P2**(b) holds because only $v_0 \in \mathsf{dom}(h_0)$, and v_0 cannot be its own child (hence cannot appear in the second argument of a tuple in Frontier). To see why **P3** is satisfied, first suppose that $u_0 \in \mathsf{inds}(\mathcal{A})$. Then in Step 2, the subprocedure MapCore was called on input $(\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_0, u_0)$ and returned **yes**. It follows that

- if $v_0 = z_j$, then $u_0 = b_j$;
- if $\mathbf{q}$ contains $A(v_0)$, then $u_0 \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
- if $\mathbf{q}$ contains $r(v, v)$, then $(u_0, u_0) \in r^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;

and hence that h_0 defines a homomorphism of $\mathbf{q}_0$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. The other possibility is that $u_0 = a_0 w_0$ for some non-empty word $w_0 = w_0' R$, and so in Step 2, MapAnon was called on input $(\mathcal{T}, \mathbf{q}, v_0, R)$ and returned **yes**. It follows that

- $v_0 \notin \mathsf{avars}(\mathbf{q})$;
- if $\mathbf{q}$ contains $A(v)$, then $\mathcal{T} \models \exists y R(y, x) \rightarrow A(x)$ (hence: $u_0 \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$);
- $\mathbf{q}$ does not contain any atom of the form $S(v, v)$;

and hence h_0 maps all atoms of $\mathbf{q}_0$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. We have thus shown that the initial partial function h_0 satisfies the three requirements.

Next we show how to inductively define h_i from h_{i-1} while preserving properties **P1** –**P3**. The variables that belong to $\mathsf{dom}(h_i) \setminus \mathsf{dom}(h_{i-1})$ are precisely those variables that appear in the second position of a tuple removed from Frontier during iteration i (since these are the variables for which we guess a domain element). The choice of where to map these variables depends on which of three options was selected:

Option 1: In this case, we removed a tuple $(v_1, v_2, c, 0)$ and guessed an individual $d \in \mathsf{inds}(\mathcal{A})$. We set $h_i(v_2) = d$ and $h_i(v) = h_{i-1}(v)$ for all variables in $\mathsf{dom}(h_{i-1})$ (all other variables remain undefined). Property **P1** is trivially satisfied.

For property **P2**, let Stack_{i-1} designate Stack at the beginning of iteration i, and let Stack_i designate Stack at the beginning of iteration $i+1$. Consider some tuple $\tau = (v, v', a, p)$ that belongs to Frontier at the beginning of iteration $i+1$ (equivalently, the end of iteration i). If the tuple τ was already in Frontier at the beginning of iteration i, then we can use the fact that h_{i-1} satisfies **P2** to obtain that:

(a) $h_{i-1}(v) = cw$ where $w = \mathsf{Stack}_{i-1}[n]$
(b) neither v' nor any of its descendants belongs to $\mathsf{dom}(h_{i-1})$

Since $\mathsf{Stack}_i = \mathsf{Stack}_{i-1}$ and $h_i(v) = h_{i-1}(v)$, it follows that statement (a) continues to hold for τ. Moreover, since τ was not removed from Frontier during iteration i, we have that $\tau \neq (v_1, v_2, c, 0)$, and so using Claim 1, we can conclude that $v' \neq v_2$. It follows that neither v' nor any descendant is in $\mathsf{dom}(h_i)$. The other possibility is that the tuple τ was added to Frontier during iteration i, in which case $\tau = (v_2, v_3, d, 0)$ for some child v_3 of v_2. Condition (a) is clearly satisfied (since $\mathsf{Stack}_i[0] = \epsilon$). Since h_{i-1} satisfies **P2**, we know that v_3 (being a descendant of v_2) is not in $\mathsf{dom}(h_{i-1})$, and so remains undefined for h_i.

To show property **P3**, we first note that since h_i agrees with h_{i-1} on all variables in $\mathsf{dom}(h_i)$, it is only necessary to consider the atoms in $\mathbf{q}_i$ that do not belong to $\mathbf{q}_{i-1}$. There are four kinds of such atoms:

- Atoms of the form $A(v_2)$: if $A(v_2) \in \mathbf{q}$, then MapCore($\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_2, d$)=**true** implies that $h_i(v_2) = d \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $R(v_2, v_2)$: if $R(v_2, v_2) \in \mathbf{q}$, then we can again use the fact that MapCore($\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_2, d$)=**true** to infer that $(h_i(v_2), h_i(v_2)) = (d, d) \in R^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $R(v_2, v)$ with $v \neq v_2$: since $R(v_2, v) \in \mathbf{q}_i$, we know that v must belong to $\mathsf{dom}(h_i)$, so v must be the parent v_1 (rather than one of v_2's children). We can thus use the fact that MapEdge($\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_1, v_2, c, d$)=**true** to obtain $(h_i(v_2), h_i(v)) = (c, d) \in R^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $R(v, v_2)$ with $v \neq v_2$: analogous to the previous case.

We have thus shown that property **P3** holds for h_i.

Option 2: If Option 2 was selected during iteration i, then a tuple (v_1, v_2, c, n) was removed from Frontier with n equal to the value of Height, and then a role S was guessed. We set $h_i(v_2) = h_{i-1}(v_1)S$. Note that we are sure that $h_{i-1}(v_1)$ is defined, since h_{i-1} satisfies property **P2**. Moreover, the first two checks in Step 4(e) ensure that $h_{i-1}(v_1)S$ belongs to the domain of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. We also set $h_i(v) = h_{i-1}(v)$ for all variables in $\mathsf{dom}(h_{i-1})$ and leave the remaining variables undefined.

Property **P1** is immediate from the definition of h_i, and property **P2**(b) can be shown exactly as for Option 1. To show **P2**(a), we define Stack_{i-1} and Stack_i as in Option 1, and consider a tuple $\tau = (v, v', a, p)$ that belongs to Frontier at the beginning of iteration $i+1$. If τ was present in Frontier at the beginning of iteration i, then $h_{i-1}(v) = cw$ where $w = \mathsf{Stack}_{i-1}[p]$ (since h_{i-1} satisfies **P2**). Since $\mathsf{Stack}_i = \mathsf{Stack}_{i-1} S$, $p \leq |\mathsf{Stack}_{i-1}|$ and $h_i(v) = h_{i-1}(v)$, it follows that statement (a) continues to hold for τ. The other possibility is that τ was added to Frontier during iteration i, in which case τ must take the form

$(v_2, v_3, c, n+1)$ for some child v_3 of v_2. Since h_{i-1} satisfies **P2**, we know that $h_{i-1}(v_1) = c \cdot \mathsf{Stack}_{i-1}[n]$. Statement (a) follows then from the fact that $h_i(v_2) = h_{i-1}(v_1)S$ and $\mathsf{Stack}_i = \mathsf{Stack}_{i-1}\, S$.

We now turn to property **P3**. As explained in the proof for Option 1, it is sufficient to consider the atoms in $\mathbf{q}_i \setminus \mathbf{q}_{i-1}$, which can be of the following four types:

- Atoms of the form $A(v_2)$: if $A(v_2) \in \mathbf{q}$, then MapAnon($\mathcal{T}, \mathbf{q}, v_2, S$)=**true** implies that $\mathcal{T} \models \exists y S(y, x) \rightarrow A(x)$, hence $h_i(v_2) = h_{i-1}(v_1)S \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $R(v_2, v_2)$: MapAnon($\mathcal{T}, \mathbf{q}, v_2, S$)=**true** implies that no such atom occurs in $\mathbf{q}$.
- Atoms of the form $R(v_2, v)$ with $v \neq v_2$: if $R(v_2, v) \in \mathbf{q}_i$, the only possibility is that $v = v_1$ (cf. proof for Option 1). We know from the third check in Step 4(e) that $\mathcal{T} \models S(x, y) \rightarrow R(y, x)$, which shows that $(h_i(v_2), h_i(v)) = (h_{i-1}(v_1)S, h_{i-1}(v_1)) \in R^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $R(v, v_2)$ with $v \neq v_2$: analogous to the previous case.

This establishes that h_i is a homomorphism from $\mathbf{q}_i$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, so h_i satisfies **P3**.

Option 3: If it is Option 3 that was selected during iteration i, then the tuples in $\mathsf{Deepest} = \{(v_1, v_2, c, n) \in \mathsf{Frontier} \mid n = \mathsf{Height}\}$ were removed from Frontier, and the role R was popped from Stack. We know from Claim 2 that all tuples in Deepest contain the same individual c in their third position. For every variable $v \in \mathsf{DVars} = \{v_2 \mid v_1, v_2, c, n) \in \mathsf{Deepest}\}$, we set $h_i(v) = cw$, where w is equal to Stack after R has been popped. As for the other two options, we set $h_i(v) = h_{i-1}(v)$ for all variables in $\mathsf{dom}(h_{i-1})$ and leave the remaining variables undefined.

Property **P1** is again immediate, and the argument for property **P2**(b) is the same as for Option 1. For property **P2**(a), let Stack_{i-1} and Stack_i be defined as earlier, l and let $\tau = (v, v', a, p)$ be a tuple that in Frontier at the beginning of iteration $i+1$. If τ was present in Frontier at the beginning of iteration i, then $h_{i-1}(v) = cw$ where $w = \mathsf{Stack}_{i-1}[p]$, and p must be smaller than the value of Height at the start of iteration i. We know that Stack_i is obtained from Stack_{i-1} by popping one or more symbols, and that at the end of iteration i, Height is equal to the largest value appearing in a tuple of Frontier. We thus know that at the start of iteration $i+1$, $p \leq \mathsf{Height}$, and so **P2**(a) continues to hold for τ. Next consider the other possibility, which is that the tuple $\tau = (v, v', a, p)$ was added to Frontier during the ith iteration of the while loop. In this case, we know that $v \in \mathsf{DVars}$, $h_i(v) = c\mathsf{Stack}_i$, and $p = |\mathsf{Stack}_i|$, from which property **P2**(a) follows.

For property **P3**, the argument is similar to the other two options and involves considering the different types of atoms that may appear in $\mathbf{q}_i \setminus \mathbf{q}_{i-1}$:

- Atoms of the form $A(v)$ with $v \in \mathsf{DVars}$: if $A(v_2) \in \mathbf{q}$, then either
  - $|\mathsf{Stack}_{i-1}| = 1$ and MapCore($\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v_2, c$)= **true**, or
  - $|\mathsf{Stack}_{i-1}| > 1$ and MapAnon($\mathcal{T}, \mathbf{q}, v_2, S$)=**true**, where S is next-to-top symbol in Stack_{i-1}

  In both cases, we may infer $h_i(v) \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ (see Options 1 and 2).
- Atoms of the form $P(v, v)$ with $v \in \mathsf{DVars}$: in this case, we must have $\mathsf{Height} = 0$, $h_i(v) = c$, and MapCore($\mathcal{T}, \mathcal{A}, \mathbf{q}, \mathbf{b}, v, c$)=**true**. The latter implies that $(h_i(v), h_i(v)) = (c, c) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $P(v, v')$ with $v \neq v'$ and $v \in \mathsf{DVars}$: if $P(v, v') \in \mathbf{q}_i$, the only possibility is that v' is the parent of v (cf. proof for Option 1). We know from the third check in Step 4(h) that $\mathcal{T} \models R(y, x) \rightarrow P(x, y)$, which shows that $(h_i(v), h_i(v')) = (c\,\mathsf{Stack}_i, c\,\mathsf{Stack}_i\, R) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
- Atoms of the form $P(v', v)$ with $v \neq v'$ and $v \in \mathsf{DVars}$: analogous to the previous case.

We claim that the final partial function h_L is a homomorphism of $\mathbf{q}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Since h_L is a homomorphism of $\mathbf{q}_L$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, it suffices to show that $\mathbf{q} = \mathbf{q}_L$, or equivalently, that all variables of $\mathbf{q}$ are in $\mathsf{dom}(h_L)$. This follows from Claim 1 and the fact that $\mathsf{dom}(h_{i+1}) = \mathsf{dom}(h_i) \cup \{v' \mid (v, v', c, n)$ is removed from Frontier during iteration $i\}$. (*end proof of claim*) □

To complete our proof of the LOGCFL upper bound, we prove the following proposition.

Proposition 29. *BLQuery can be implemented by an NAuxPDA*

Proof. It suffices to show that BLQuery runs in non-deterministic logarithmic space and polynomial time.

In Step 1, we non-deterministically fix a root variable v_0, but do not actually need to store the induced directed tree T in memory, since it suffices to be able to decide given two variables v, v' whether v is the parent of v' in T, and the latter problem clearly belongs to NL.

In Step 2, we need only logarithmic space to store the individual a_0. The word $w_0 = \varrho_1 \dots \varrho_N$ can be guessed symbol by symbol and pushed onto Stack. We recall that $a_0 w_0 \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ just in the case that:

- $\mathcal{T}, \mathcal{A} \models \exists y \varrho_1(a, y)$ and $\mathcal{T}, \mathcal{A} \not\models \varrho_1(a, b)$ for any $b \in \mathsf{inds}(\mathcal{A})$;
- for every $1 \leq i < N$: $\mathcal{T} \models \exists y\, \varrho_i(y, x) \rightarrow \exists y\, \varrho_{i+1}(x, y)$ and $\mathcal{T} \not\models \varrho_i(x, y) \rightarrow \varrho_{i+1}(y, x)$.

Thus, it is possible to perform the required entailment checks incrementally as the symbols of w_i are guessed. Finally, to ensure that the guessed word w_0 does not exceed the length bound, each time we push a symbol onto Stack, we increment

Height by 1. If Height reaches $2|\mathcal{T}| + |\mathbf{q}|$, then no more symbols may be guessed. We next call either sub-procedure MapCore or MapAnon. It is easy to see that both can be made to run in non-deterministic logarithmic space.

The initializations of Stack and Height in Step 3 were already handled in our discussion of Step 2. Since the children of a node in T can be identified in NL, we can decide in non-deterministic logspace whether a tuple $(v_0, v_i, a_0, \mathsf{Height})$ should be included in Frontier. Moreover, since the input query $\mathbf{q}$ is a tree-shaped query with a bounded number of leaves, we know that only constantly many tuples can be added to Frontier in Step 4. Moreover, it is clear that every tuple can be stored using in logarithmic space. More generally, using Claims 1 and 2 from the proof of Proposition 23, one can show that $|\mathsf{Frontier}|$ is bounded by a constant throughout the execution of the procedure, and the tuples added during the while loop can also be stored using only logarithmically many bits.

Next observe that every iteration of while loop in Step 4 involves a polynomial number of the following elementary operations:

- remove a tuple from Frontier, or add a tuple to Frontier
- pop a role from Stack, or push a role onto Stack
- increment or decrement Height by a number bounded by $2|\mathcal{T}| + |q|$
- test whether Height is equal to 0 or to $2|\mathcal{T}| + |q|$
- guess a single individual constant or symbol
- identify the children of a given variable
- locate an atom in $\mathbf{q}$
- test whether $\mathcal{T} \models \alpha$, for some inclusion α involving symbols from $\mathcal{T}$
- make a call to one of the sub-procedures MapCore, MapAnon, or MapEdge

For each of the above operations, it is either easy to see, or has already been explained, that the operation can be performed in non-deterministic logarithmic space. To complete the argument, we note that it follows from Claim 1 (proof of Proposition 23) that there are at most $|\mathbf{q}|$ many iterations of the while loop. □

Proposition 30. *For a logspace-uniform family $\{\boldsymbol{C}_l\}_{l=1}^{\infty}$ of* SAC^1 *circuits in normal form, the sequences $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}_l}$ and $(\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}_l}, \mathcal{A}_{\boldsymbol{C}_l})$ are also logspace uniform.*

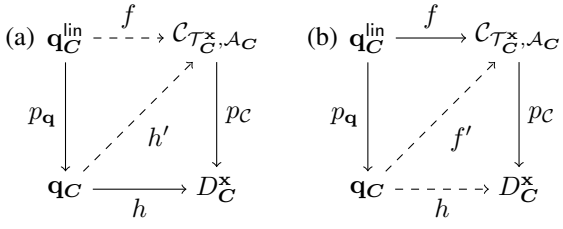
Proof. Consider a circuit $\boldsymbol{C}$ in normal form with $2d+1$ layers of gates, where d is logarithmic in number of its inputs l. We show that $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ and $(\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}})$ can be constructed using $O(\log(l))$ worktape memory.

- To produce the query $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$, we can generate the word w_d letter by letter and insert the corresponding variables. This can be done by a simple recursive procedure of depth d, using the worktape to remember the current position in the recursion tree as well as the index of the current variable y_i. Note that $|w_d|$ (hence the largest index of the query variables) may be exponential in d, but is only polynomial in l, and so we need only logarithmic space to store the index of the current variable.
- The ontology $\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}$ is obtained by making a single pass over a (graph representation) of the circuit and generating the axioms that correspond to the gates of $\boldsymbol{C}$ and the links between $\boldsymbol{C}$'s gates. To decide which axioms of the form $G_i(x) \to A(x)$ to include, we must also look up the value of the variables associated to the input gates under the valuation $\mathbf{x}$.
- $\mathcal{A}_{\boldsymbol{C}}$ consists of a single constant atom.

□

Proposition 24. *$\boldsymbol{C}$ accepts input $\mathbf{x}$ iff $\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}} \models \mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}(a)$.*

Proof. Denote by $p_{\mathbf{q}}$ the natural homomorphism from $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ to $\mathbf{q}_{\boldsymbol{C}}$, and by $p_{\mathcal{C}}$ the natural homomorphism from $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}}}$ to $D^{\mathbf{x}}_{\boldsymbol{C}}$. As it is proven in [14] that $\boldsymbol{C}$ accepts input $\mathbf{x}$ iff there is a homomorphism h from $\mathbf{q}_{\boldsymbol{C}}$ to $D^{\mathbf{x}}_{\boldsymbol{C}}$, it suffices to show that there exists a homomorphism f from $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ to $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}}}$ iff there is a homomorphism h from $\mathbf{q}_{\boldsymbol{C}}$ to $D^{\mathbf{x}}_{\boldsymbol{C}}$.



($\Rightarrow$) Suppose that h is a homomorphism from $\mathbf{q}_{\boldsymbol{C}}$ to $D^{\mathbf{x}}_{\boldsymbol{C}}$. We define the homomorphism $h' : \mathbf{q}_{\boldsymbol{C}} \to \mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}}}$ inductively moving from the root n_1 of $\mathbf{q}_{\boldsymbol{C}}$ to its leaves. First, we set $h'(n_1) = a$. Note that $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\boldsymbol{C}}, \mathcal{A}_{\boldsymbol{C}}} \models G_1(a)$. Then we proceed by

induction. Suppose that n_j is a child of n_i, $h'(n_i)$ is defined, $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}} \models G_{i'}(h(n_i))$ and $h(n_j) = g_{j'}$. In this case, we set $h'(n_j) = h'(n_i)P^{-}_{i'j'}$. It follows from the definition of $\mathcal{T}^{\mathbf{x}}_{\mathcal{C}}$ that $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}} \models G_{j'}(h'(n_j))$, which enables us to continue the induction. It should be clear that h' is indeed a homomorphism from $\mathbf{q}_{\boldsymbol{C}}$ into $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}}$. Since the composition of homomorphisms is again a homomorphism, we can obtain the desired homomorphism $f : \mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}} \to \mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}}$ by setting $f = p_{\mathbf{q}} \circ h'$. This is illustrated in diagram (a) above.

($\Leftarrow$) Suppose that f is a homomorphism from $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ to $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}}$. We prove that for all its variables y_i, y_j (with $i < j$) $p_{\mathbf{q}}(y_i) = p_{\mathbf{q}}(y_j)$ implies $f(y_i) = f(y_j)$ by induction on $|j - i|$. The base case ($|j - i| = 0$) is trivial. For the inductive step, we may assume without loss of generality that between y_i and y_j there are no intermediate variable y_k with $p_{\mathbf{q}}(y_i) = p_{\mathbf{q}}(y_k) = p_{\mathbf{q}}(y_j)$ (otherwise, we can simply use the induction hypothesis together with the transitivity of equality). It follows that $p_{\mathbf{q}}(y_{i+1}) = p_{\mathbf{q}}(y_{j-1})$, and the atom between y_{j-1} and y_j is oriented from y_{i-1} towards y_j, while the atom between y_i and y_{i+1} goes from y_{i+1} to y_i. Indeed, it holds if the node $n = p_{\mathbf{q}}(y_i) = p_{\mathbf{q}}(y_j)$ is an OR-node since there are exactly two variables in $\mathbf{q}^{\mathsf{lin}}_{\boldsymbol{C}}$ which are mapped to n, and they bound the subtree in $\mathbf{q}_{\boldsymbol{C}}$ generated by n. For an AND-node, this also holds because of our assumption about intermediate variables. By the induction hypothesis, we have $f(y_{i+1}) = f(y_{j-1}) = aw\varrho$ for some word $aw\varrho$. Since the only parent of $aw\varrho$ in $\mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}}$ is aw, all arrows in relations U, L and R are oriented towards the root, and f is known to be a homomorphism, it follows that $f(y_i) = f(y_j) = aw$. This concludes the inductive argument.

Next define the function $f' : \mathbf{q}_{\boldsymbol{C}} \to \mathcal{C}_{\mathcal{T}^{\mathbf{x}}_{\mathcal{C}},\mathcal{A}_{\boldsymbol{C}}}$ by setting $f'(x) = f(y)$ where y is such that $p_{\mathbf{q}}(y) = x$. Since $p_{\mathbf{q}}(y_i) = p_{\mathbf{q}}(y_j)$ implies $f(y_i) = f(y_j)$, we have that f' is well-defined, and because f is a homomorphism, the same holds for f'. To obtain the desired homomorphism from $\mathbf{q}_{\boldsymbol{C}}$ to $D^{\mathbf{x}}_{\mathcal{C}}$, it suffices to consider the composition h of f' and $p_{\mathcal{C}}$. □

REFERENCES

[33] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev, "The DL-Lite family and relations," *Journal of Artificial Intelligence Research (JAIR)*, vol. 36, pp. 1–69, 2009.

[34] T. Kloks, *Treewidth: Computations and Approximations*, ser. Lecture Notes in Computer Science. Springer, 1994, vol. 842.

[35] A. Wigderson, *The complexity of graph connectivity*. Springer, 1992.