Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity

MEGHYN BIENVENU, CNRS & University of Montpellier
STANISLAV KIKOT, Birkbeck, University of London
ROMAN KONTCHAKOV, Birkbeck, University of London
VLADIMIR PODOLSKII, Steklov Mathematical Institute, Moscow
MICHAEL ZAKHARYASCHEV, Birkbeck, University of London

We give solutions to two fundamental computational problems in ontology-based data access with the W3C standard ontology language *OWL 2 QL*: the succinctness problem for first-order rewritings of ontology-mediated queries (OMQs), and the complexity problem for OMQ answering. We classify OMQs according to the shape of their conjunctive queries (treewidth, the number of leaves) and the existential depth of their ontologies. For each of these classes, we determine the combined complexity of OMQ answering, and whether all OMQs in the class have polynomial-size first-order, positive existential, and nonrecursive datalog rewritings. We obtain the succinctness results using hypergraph programs, a new computational model for Boolean functions, which makes it possible to connect the size of OMQ rewritings and circuit complexity.

Categories and Subject Descriptors: I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; H.2.3 [**Database Management**]: Languages—Query Languages; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—Computational logic

General Terms: Theory, Languages, Algorithms.

Additional Key Words and Phrases: Ontology-Based Data Access, Description Logic, Ontology-Mediated Query, Query Rewriting, Succinctness, Computational Complexity, Circuit Complexity.

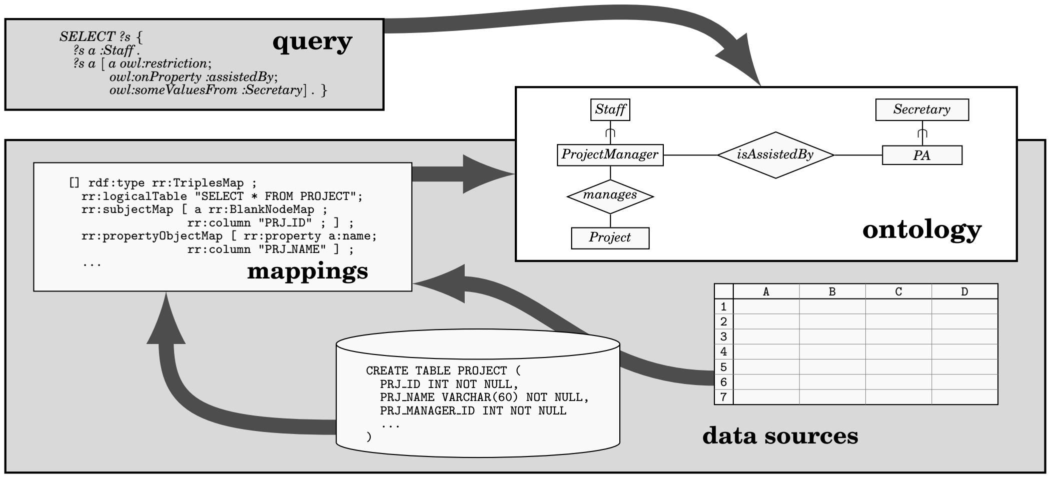
ACM Reference Format:
Meghyn Bienvenu, Stanislav Kikot, Roman Kontchakov, Vladimir Podolskii, and Michael Zakharyaschev, 2016. Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity *J. ACM* 9, 4, Article 39 (March 2010), 51 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

1.1. Ontology-based data access

Ontology-based data access (OBDA) via query rewriting was proposed by Poggi et al. [2008] with the aim of facilitating query answering over complex, possibly incomplete and heterogeneous data sources. In an OBDA system (see Fig. 1), the user does not have to be aware of the structure of data sources, which can be relational databases, spreadsheets, RDF triplestores, etc. Instead, the system provides the user with an

This work was supported by the French ANR JCJC grant 12-JS02-007-01 'PAGODA: Practical Algorithms for Ontology-Based Data Access', the UK EPSRC grant EP/M012670 'iTract: Islands of Tractability in Ontology-Based Data Access', and the Russian Foundation for Basic Research and grant MK-7312.2016.1.
Authors' addresses: M. Bienvenu, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier, 860 rue de St Priest, 34095 Montpellier cedex 5, France. S. Kikot, R. Kontchakov, M. Zakharyaschev, Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London WC1E 7HX, UK. V. Podolskii, Steklov Mathematical Institute of the Russian Academy of Sciences, 8 Gubkina str., 119991, Moscow, Russia.



```
SELECT ?s {
  ?s a :Staff .
  ?s a [ a owl:restriction;
         owl:onProperty :assistedBy;
         owl:someValuesFrom :Secretary] . }
```

```
[] rdf:type rr:TriplesMap ;
  rr:logicalTable "SELECT * FROM PROJECT";
  rr:subjectMap [ a rr:BlankNodeMap ;
                  rr:column "PRJ_ID" ; ] ;
  rr:propertyObjectMap [ rr:property a:name;
                  rr:column "PRJ_NAME" ] ;
  ...
```

```
CREATE TABLE PROJECT (
  PRJ_ID INT NOT NULL,
  PRJ_NAME VARCHAR(60) NOT NULL,
  PRJ_MANAGER_ID INT NOT NULL
  ...
)
```

Fig. 1. Ontology-based data access.

ontology that serves as a high-level conceptual view of the data, gives a convenient vocabulary for user queries, and enriches incomplete data with background knowledge. A snippet, $\mathcal{T}$, of such an ontology is shown below in the syntax of first-order (FO) logic:

$$\forall x\,\big(\textit{ProjectManager}(x) \to \exists y\,(\textit{isAssistedBy}(x,y) \land \textit{PA}(y))\big),$$
$$\forall x\,\big(\exists y\,\textit{managesProject}(x,y) \to \textit{ProjectManager}(x)\big),$$
$$\forall x\,\big(\textit{ProjectManager}(x) \to \textit{Staff}(x)\big),$$
$$\forall x\,\big(\textit{PA}(x) \to \textit{Secretary}(x)\big).$$

User queries are formulated in the signature of the ontology. For example, the conjunctive query (CQ)

$$\boldsymbol{q}(x) \;=\; \exists y\,(\textit{Staff}(x) \land \textit{isAssistedBy}(x,y) \land \textit{Secretary}(y)))$$

is supposed to find the staff assisted by secretaries. The ontology signature and data schemas are related by mappings designed by the ontology engineer and invisible to the user. The mappings allow the system to view the data sources as a single RDF graph (a finite set of unary and binary atoms), $\mathcal{A}$, in the signature of the ontology. For example, the global-as-view (GAV) mappings

$$\forall x,y,z\,\big(\texttt{PROJECT}(x,y,z) \to \textit{managesProject}(x,z)\big),$$
$$\forall x,y\,\big(\texttt{STAFF}(x,y) \land (y=2) \to \textit{ProjectManager}(x)\big)$$

populate the ontology predicates *managesProject* and *ProjectManager* with values from the database relations PROJECT and STAFF. In the query rewriting approach of Poggi et al. [2008], the OBDA system employs the ontology and mappings in order to transform the user query into a query over the data sources, and then delegates the actual query evaluation to the underlying database engines and triplestores.

For example, the first-order query

$$\boldsymbol{q}'(x) \;=\; \exists y\,\big[\textit{Staff}(x) \land \textit{isAssistedBy}(x,y) \land (\textit{Secretary}(y) \lor \textit{PA}(y))\big] \lor \textit{ProjectManager}(x) \lor \exists z\,\textit{managesProject}(x,z)$$

is an *FO-rewriting* of the *ontology-mediated query* (OMQ) $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ over any RDF graph $\mathcal{A}$ in the sense that a is an answer to $\boldsymbol{q}'(x)$ over $\mathcal{A}$ iff $\boldsymbol{q}(a)$ is a logical consequence of $\mathcal{T}$ and $\mathcal{A}$. As the system is not supposed to materialise $\mathcal{A}$, it uses the mappings to unfold the rewriting $\boldsymbol{q}'$ into an SQL (or SPARQL) query over the data sources.

Ontology languages suitable for OBDA via query rewriting have been identified by the Description Logic, Semantic Web, and Database/Datalog communities. The *DL-Lite* family of description logics, first proposed by Calvanese et al. [2007] and later extended by Artale et al. [2009], was specifically designed to ensure the existence of FO-rewritings for all conjunctive queries (CQs). Based on this family, the W3C defined a profile *OWL 2 QL*[1] of the Web Ontology Language *OWL 2* 'so that data [. . .] stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism.' Various dialects of tuple-generating dependencies (tgds) that admit FO-rewritings of CQs and extend *OWL 2 QL* have also been identified [Baget et al. 2011; Calì et al. 2012b; Civili and Rosati 2012]. We note in passing that while most work on OBDA (including the present paper) assumes that the user query is given as a CQ, other query languages, allowing limited forms of recursion and/or negation, have also been investigated [Rosati 2007; Gutiérrez-Basulto et al. 2015; Bienvenu et al. 2015; Kostylev et al. 2015]. SPARQL 1.1, the standard query language for RDF graphs, contains negation, aggregation and other features beyond first-order logic. The entailment regimes of SPARQL 1.1[2] also bring inferencing capabilities in the setting, which are, however, necessarily limited for efficient implementations.

By reducing OMQ answering to standard database query evaluation, which is generally regarded to be very efficient, OBDA via query rewriting has quickly become a hot topic in both theory and practice. A number of rewriting techniques have been proposed and implemented for *OWL 2 QL* (PerfectRef [Poggi et al. 2008], Presto/Prexto [Rosati and Almatelli 2010; Rosati 2012], tree witness rewriting [Kikot et al. 2012a]), sets of tuple-generating dependencies (Nyaya [Gottlob et al. 2011], PURE [König et al. 2015b]), and more expressive ontology languages that require recursive datalog rewritings (Requiem [Pérez-Urbina et al. 2009], Rapid [Chortaras et al. 2011], Clipper [Eiter et al. 2012] and Kyrie [Mora et al. 2014]). A few mature OBDA systems have also recently emerged: pioneering MASTRO [Calvanese et al. 2011], commercial Stardog [Pérez-Urbina et al. 2012] and Ultrawrap [Sequeda et al. 2014], and the Optique platform [Giese et al. 2015] based on the query answering engine Ontop [Rodriguez-Muro et al. 2013; Kontchakov et al. 2014]. By providing a semantic end-to-end connection between users and multiple distributed data sources (and thus making the IT expert middleman redundant), OBDA has attracted the attention of industry, with companies such as Siemens [Kharlamov et al. 2014] and Statoil [Kharlamov et al. 2015] experimenting with OBDA technologies to streamline the process of data access for their engineers.[3]

1.2. Succinctness and complexity

In this paper, our concern is two fundamental theoretical problems whose solutions will elucidate the computational costs required for answering OMQs with *OWL 2 QL* ontologies. The succinctness problem for FO-rewritings is to understand how difficult it is to construct FO-rewritings for OMQs in a given class and, in particular, to determine whether OMQs in the class have polynomial-size FO rewritings or not. In other words, the succinctness problem clarifies the computational costs of the *reduction* of OMQ answering to database query evaluation. On the other hand, it is also important to

[1]http://www.w3.org/TR/owl2-overview/#Profiles
[2]http://www.w3.org/TR/2013/REC-sparql11-entailment-20130321
[3]See, e.g., http://optique-project.eu.

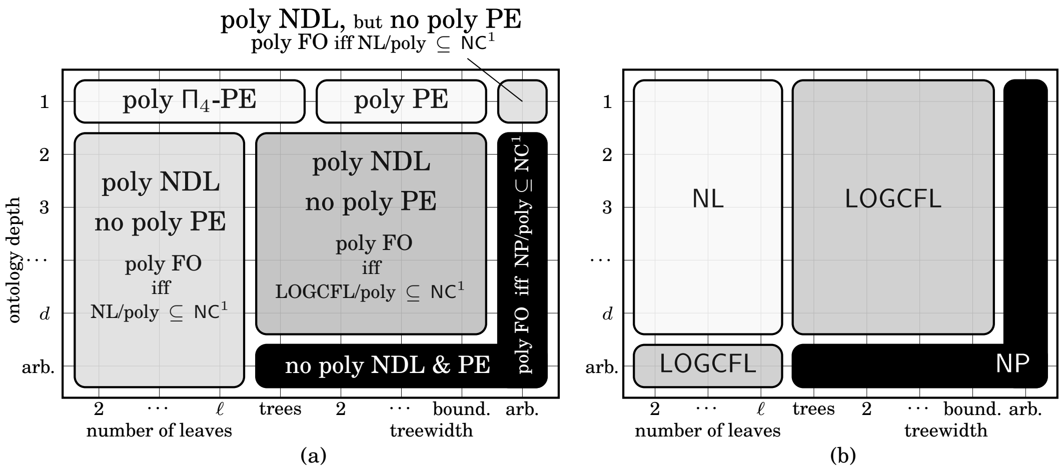


Fig. 2. (a) Succinctness of OMQ rewritings, and (b) combined complexity of OMQ answering (tight bounds).

measure the resources required to answer OMQs by a *best possible algorithm*, not necessarily a reduction to database query evaluation. Thus, we are interested in the combined complexity of the OMQ answering problem: given an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ from a certain class, a data instance $\mathcal{A}$ and a tuple $\boldsymbol{a}$ of constants from $\mathcal{A}$, decide whether $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$. The combined complexity of CQ evaluation has been thoroughly investigated in database theory; cf. [Grohe et al. 2001; Libkin 2004] and references therein. To slightly simplify the setting for our problems, we assume that data is given in the form of an RDF graph and leave mappings out of the picture (in fact, GAV mappings only polynomially increase the size of FO-rewritings over RDF graphs).

We suggest a 'two-dimensional' classification of OMQs. One dimension takes account of the shape of the CQs in OMQs by quantifying their treewidth (as in classical database theory) and the number of leaves in tree-shaped CQs. Note that, in SPARQL 1.1, the sub-queries that require rewriting under the *OWL 2 QL* entailment regime are always tree-shaped (they are, in essence, complex class expressions). The second dimension is the existential depth of ontologies, that is, the length of the longest chain of labelled nulls in the chase on any data. Thus, the NPD FactPages ontology,[4] which was designed to facilitate querying the datasets of the Norwegian Petroleum Directorate,[5] is of depth 5. A typical example of an ontology axiom causing infinite depth is $\forall x\, \big(\mathit{Person}(x) \to \exists y\, (\mathit{ancestor}(y,x) \land \mathit{Person}(y))\big)$.

1.3. Results

The results of our investigation are summarised in the succinctness and complexity landscapes of Fig. 2. In what follows, we discuss these results in more detail.

The *succinctness problem* we consider can be formalised as follows: given a sequence $\boldsymbol{Q}_n$ $(n < \omega)$ of OMQs whose size is polynomial in n, determine whether the size of minimal rewritings of $\boldsymbol{Q}_n$ can be bounded by a polynomial function in n. We distinguish between three types of rewritings: arbitrary FO-rewritings, positive existential (PE-) rewritings (in which only $\land$, $\lor$ and $\exists$ are allowed), and non-recursive datalog (NDL-) rewritings.[6] This succinctness problem was first considered by Kikot et al. [2012b] and

[4]http://sws.ifi.uio.no/project/npd-v2/
[5]http://factpages.npd.no/factpages/
[6]Domain-independent FO-rewritings correspond to SQL queries, PE-rewritings to SELECT-PROJECT-JOIN-UNION (or SPJU) queries, and NDL-rewritings to SPJU queries with views; see also Remark 2.3.

Gottlob and Schwentick [2012]. The former constructed a sequence $\boldsymbol{Q}_n$ of OMQs (with tree-shaped CQs) whose PE- and NDL-rewritings are of exponential size, while FO-rewritings are superpolynomial unless $\mathsf{NP} \subseteq \mathsf{P}/\mathsf{poly}$. Gottlob and Schwentick [2012] and Gottlob et al. [2014] showed that PE- (and so all other) 'rewritings' can be made polynomial under the condition that all relevant data instances contain two special constants. The 'succinctification' trick involves polynomially many extra existential quantifiers over these constants to guess a derivation of the given CQ in the chase, which makes such rewritings impractical (cf. NFAs vs DFAs, and [Avigad 2003]). In this paper, we stay within the classical OBDA setting that does not impose any extra conditions on the data and does not allow any special constants in rewritings.

Figure 2 (a) gives a summary of the succinctness results obtained in this paper. It turns out that polynomial-size PE-rewritings are guaranteed to exist—in fact, can be constructed in polynomial time—only for the class of OMQs with ontologies of depth 1 and CQs of bounded treewidth; moreover, tree-shaped OMQs have polynomial-size Π_4-PE-rewritings (with matrices of the form $\wedge\vee\wedge\vee$). Polynomial-size NDL-rewritings can be efficiently constructed for all tree-shaped OMQs with a bounded number of leaves, all OMQs with ontologies of bounded depth and CQs of bounded treewidth, and all OMQs with ontologies of depth 1. For OMQs with ontologies of depth 2 and arbitrary CQs, and OMQs with arbitrary ontologies and tree-shaped CQs, we have an exponential lower bound on the size of NDL- (and so PE-) rewritings. The existence of polynomial-size FO rewritings for all OMQs in each of these classes (save the first one) turns out to be equivalent to one of the major open problems in computational complexity such as $\mathsf{NC}^1 = \mathsf{NP}/\mathsf{poly}$.[7]

We obtain these results by establishing a connection between succinctness of rewritings and circuit complexity, a branch of computational complexity theory that classifies Boolean functions according to the size of circuits computing them. Our starting point is the observation that the tree-witness PE-rewriting of an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ introduced by Kikot et al. [2012a] defines a hypergraph whose vertices are the atoms in $\boldsymbol{q}$ and whose hyperedges correspond to connected sub-queries of $\boldsymbol{q}$ that can be homomorphically mapped to labelled nulls of some chases for $\mathcal{T}$. Based on this observation, we introduce a new computational model for Boolean functions by treating any hypergraph H, whose vertices are labelled by (possibly negated) Boolean variables or constants 0 and 1, as a program computing a Boolean function f_H that returns 1 on a valuation for the variables iff there is an independent subset of hyperedges covering all vertices labelled by 0 (under the valuation). We show that constructing short FO- (respectively, PE- and NDL-) rewritings of $\boldsymbol{Q}$ is (nearly) equivalent to finding short Boolean formulas (respectively, monotone formulas and monotone circuits) computing the hypergraph function for $\boldsymbol{Q}$.

For each of the OMQ classes in Fig. 2 (a), we characterise the computational power of the corresponding hypergraph programs and employ results from circuit complexity to identify the size of rewritings. For example, we show that OMQs with ontologies of depth 1 correspond to hypergraph programs of degree ≤ 2 (in which every vertex belongs to at most two hyperedges), and that the latter are polynomially equivalent to nondeterministic branching programs (NBPs). Since NBPs compute the Boolean functions in the class $\mathsf{NL}/\mathsf{poly} \subseteq \mathsf{P}/\mathsf{poly}$, the tree-witness rewritings for OMQs with ontologies of depth 1 can be equivalently transformed into polynomial-size NDL-rewritings. On the other hand, there exist monotone Boolean functions computable by polynomial-size NBPs but not by polynomial-size monotone Boolean formulas, which establishes

[7] C/poly is the non-uniform analogue of a complexity class C.

a superpolynomial lower bound for PE-rewritings. It also follows that all such OMQs have polynomial-size FO-rewritings iff $\mathsf{NC}^1 = \mathsf{NL/poly}$.

The succinctness results in Fig. 2 (a), characterising the complexity of the reduction to plain database query evaluation, are complemented by the combined complexity results in Fig. 2 (b). *Combined complexity* measures the time and space required for a best possible algorithm to answer an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ from the given class over a data instance $\mathcal{A}$, as a function of the size of $\boldsymbol{Q}$ and $\mathcal{A}$. It is known [Calvanese et al. 2007; Artale et al. 2009] that the general OMQ answering problem is NP-complete for combined complexity—that is, of the same complexity as standard CQ evaluation in databases. However, answering tree-shaped OMQs turns out to be NP-hard [Kikot et al. 2011] in contrast to the well-known tractability of evaluating tree-shaped and bounded-treewidth CQs [Yannakakis 1981; Chekuri and Rajaraman 2000; Gottlob et al. 1999]. Here, we prove that, surprisingly, answering OMQs with ontologies of bounded depth and CQs of bounded treewidth is no harder than evaluating CQs of bounded treewidth, that is, LOGCFL-complete. By restricting further the class of CQs to trees with a bounded number of leaves, we obtain an even better NL-completeness result, which matches the complexity of evaluating the underlying CQs. If we consider bounded-leaf tree-shaped CQs coupled with arbitrary *OWL 2 QL* ontologies, then the OMQ answering problem remains tractable, LOGCFL-complete to be more precise.

The plan of the paper is as follows. Section 2 gives formal definitions of *OWL 2 QL*, OMQs and rewritings. Section 3 defines the tree-witness rewriting. Section 4 reduces the succinctness problem for OMQ rewritings to the succinctness problem for hypergraph Boolean functions associated with tree-witness rewritings, and introduces hypergraph programs for computing these functions. Section 5 establishes a correspondence between the OMQ classes in Fig. 2 and the structure of the corresponding hypergraph functions and programs. Section 6 characterises the computational power of hypergraph programs in these classes by relating them to standard models of computation for Boolean functions. Section 7 uses the results of the previous three sections and some known facts from circuit complexity to obtain the upper and lower bounds on the size of PE-, NDL- and FO-rewritings in Fig. 2 (a). Section 8 establishes the combined complexity results in Fig. 2 (b). We conclude in Section 9 by discussing the obtained succinctness and complexity results and formulating a few open problems. All omitted proofs can be found in the appendix.

2. *OWL 2 QL* ONTOLOGY-MEDIATED QUERIES AND FIRST-ORDER REWRITABILITY

In first-order logic, any *OWL 2 QL ontology* (or *TBox* in description logic parlance), $\mathcal{T}$, can be given as a finite set of sentences (often called *axioms*) of the following forms

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

where the formulas $\tau(x)$ (called *classes* or *concepts*) and $\varrho(x,y)$ (called *properties* or *roles*) are defined, using unary predicates A and binary predicates P, by the grammars

$$\tau(x) \;::=\; \top \mid A(x) \mid \exists y\, \varrho(x,y) \qquad \text{and} \qquad \varrho(x,y) \;::=\; \top \mid P(x,y) \mid P(y,x). \tag{1}$$

(Strictly speaking, *OWL 2 QL* ontologies can also contain inequalities $a \neq b$, for constants a and b. However, they do not have any impact on the problems considered in this paper, and so will be ignored.)

Example 2.1. To illustrate, we show a snippet of the NPD FactPages ontology:

$$\forall x\, (\textit{GasPipeline}(x) \to \textit{Pipeline}(x)),$$

$$\begin{array}{l}\forall x\,(\boldsymbol{FieldOwner}(x) \leftrightarrow \exists y\, \boldsymbol{ownerForField}(x,y)),\\ \forall y\,(\exists x\, \boldsymbol{ownerForField}(x,y) \to \boldsymbol{Field}(y)),\\ \forall x,y\,(\boldsymbol{shallowWellboreForField}(x,y) \to \boldsymbol{wellboreForField}(x,y)),\\ \forall x,y\,(\boldsymbol{isGeometryOfFeature}(x,y) \leftrightarrow \boldsymbol{hasGeometry}(y,x)).\end{array}$$

To simplify presentation, in our ontologies we also use sentences of the form

$$\forall x\,\big(\tau(x) \to \zeta(x)\big), \tag{2}$$

where

$$\zeta(x) \ ::= \ \tau(x) \ \mid \ \zeta_1(x) \land \zeta_2(x) \ \mid \ \exists y\,\big(\varrho_1(x,y) \land \dots \land \varrho_k(x,y) \land \zeta(y)\big).$$

It is readily seen that such sentences are just syntactic sugar and can be eliminated by means of polynomially many fresh predicates. Indeed, any axiom of the form (2) with

$$\zeta(x) = \exists y\,\big(\varrho_1(x,y) \land \dots \land \varrho_k(x,y) \land \zeta'(y)\big)$$

can be (recursively) replaced by the following axioms, for a fresh P_ζ and $i = 1, \dots, k$:

$$\forall x\,\big(\tau(x) \to \exists y\, P_\zeta(x,y)\big), \quad \forall x,y\,\big(P_\zeta(x,y) \to \varrho_i(x,y)\big), \quad \forall y\,\big(\exists x\, P_\zeta(x,y) \to \zeta'(y)\big) \tag{3}$$

because any first-order structure is a model of (2) iff it is a restriction of some model of (3) to the signature of (2). The result of eliminating the syntactic sugar from an ontology $\mathcal{T}$ is called the *normalisation* of $\mathcal{T}$. We always assume that all of our ontologies are normalised even though this is not done explicitly; however, we stipulate (without loss of generality) that the normalisation predicates P_ζ never occur in the data.

When writing ontology axioms, we usually omit the universal quantifiers. We typically use the characters P, R to denote binary predicates, A, B, C for unary predicates, and S for either of them. For a binary predicate P, we write P^- to denote its inverse; that is, $P(x,y) = P^-(y,x)$, for any x and y, and $P^{--} = P$.

A *conjunctive query* (CQ) $\boldsymbol{q}(\boldsymbol{x})$ is a formula of the form $\exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$, where φ is a conjunction of atoms $S(\boldsymbol{z})$ all of whose variables are among $\boldsymbol{x}$, $\boldsymbol{y}$.

Example 2.2. Here is a (fragment of a) typical CQ from the NPD FactPages:

$$\begin{array}{l}\boldsymbol{q}(x_1,x_2,x_3) \ = \ \exists y,z\,\big[\boldsymbol{ProductionLicence}(x_1) \land \boldsymbol{ProductionLicenceOperator}(y) \land\\ \qquad \boldsymbol{dateOperatorValidFrom}(y,x_2) \land \boldsymbol{licenceOperatorCompany}(y,z) \land\\ \qquad\qquad \boldsymbol{name}(z,x_3) \land \boldsymbol{operatorForLicence}(y,x_1)\big].\end{array}$$

To simplify presentation and without loss of generality, we assume that CQs do not contain constants. Where convenient, we regard a CQ as the *set* of its atoms; in particular, $|\boldsymbol{q}|$ is the *size of* $\boldsymbol{q}$. The variables in $\boldsymbol{x}$ are called the *answer variables* of a CQ $\boldsymbol{q}(\boldsymbol{x})$. A CQ without answer variables is called *Boolean*. With every CQ $\boldsymbol{q}$, we associate its *Gaifman graph* $G_{\boldsymbol{q}}$ whose vertices are the variables of $\boldsymbol{q}$ and whose edges are the pairs $\{u,v\}$ such that $P(u,v) \in \boldsymbol{q}$, for some P. A CQ $\boldsymbol{q}$ is *connected* if the graph $G_{\boldsymbol{q}}$ is connected. We call $\boldsymbol{q}$ *tree-shaped* if $G_{\boldsymbol{q}}$ is a tree[8] and if $G_{\boldsymbol{q}}$ is a tree with at most two leaves, then $\boldsymbol{q}$ is said to be *linear*.

An *OWL 2 QL ontology-mediated query* (OMQ) is a pair $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, where $\mathcal{T}$ is an *OWL 2 QL* ontology and $\boldsymbol{q}(\boldsymbol{x})$ a CQ. The *size of* $\boldsymbol{Q}$ is defined as $|\boldsymbol{Q}| = |\mathcal{T}| + |\boldsymbol{q}|$, where $|\mathcal{T}|$ is the number of symbols in $\mathcal{T}$.

A *data instance*, $\mathcal{A}$, is a finite set of unary or binary ground atoms (called an *ABox* in description logic). We denote by $\mathsf{ind}(\mathcal{A})$ the set of individual constants in $\mathcal{A}$. Given

[8]Tree-shaped CQs also go by the name of *acyclic queries* [Yannakakis 1981; Bienvenu et al. 2013].

an OMQ $\boldsymbol{Q}(\boldsymbol{x})$ and a data instance $\mathcal{A}$, a tuple $\boldsymbol{a}$ of constants from $\mathsf{ind}(\mathcal{A})$ of length $|\boldsymbol{x}|$ is called a *certain answer to* $\boldsymbol{Q}(\boldsymbol{x})$ *over* $\mathcal{A}$ if $\mathcal{I} \models \boldsymbol{q}(\boldsymbol{a})$ for all models $\mathcal{I}$ of $\mathcal{T} \cup \mathcal{A}$; in this case we write $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$. If $\boldsymbol{q}(\boldsymbol{x})$ is Boolean, a certain answer to $\boldsymbol{Q}$ over $\mathcal{A}$ is 'yes' if $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}$, and 'no' otherwise. We remind the reader [Libkin 2004] that, for any CQ $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$, any first-order structure $\mathcal{I}$ and any tuple $\boldsymbol{a}$ from its domain Δ, we have $\mathcal{I} \models \boldsymbol{q}(\boldsymbol{a})$ iff there is a map $h\colon \boldsymbol{x} \cup \boldsymbol{y} \to \Delta$ such that (*i*) if $S(\boldsymbol{z}) \in \boldsymbol{q}$ then $\mathcal{I} \models S(h(\boldsymbol{z}))$, and (*ii*) $h(\boldsymbol{x}) = \boldsymbol{a}$. If (*i*) is satisfied then h is called a *homomorphism* from $\boldsymbol{q}$ to $\mathcal{I}$, and we write $h\colon \boldsymbol{q} \to \mathcal{I}$; if (*ii*) also holds, we write $h\colon \boldsymbol{q}(\boldsymbol{a}) \to \mathcal{I}$.

Central to OBDA is the notion of OMQ rewriting that reduces the problem of finding certain answers to standard query evaluation. More precisely, an FO-formula $\boldsymbol{q}'(\boldsymbol{x})$, possibly with equality, $=$, is an *FO-rewriting of an OMQ* $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ if, for *any* data instance $\mathcal{A}$ (without the normalisation predicates for $\mathcal{T}$) and any tuple $\boldsymbol{a}$ in $\mathsf{ind}(\mathcal{A})$,

$$\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a}) \qquad \text{iff} \qquad \mathcal{I}_{\mathcal{A}} \models \boldsymbol{q}'(\boldsymbol{a}), \tag{4}$$

where $\mathcal{I}_{\mathcal{A}}$ is the first-order structure over the domain $\mathsf{ind}(\mathcal{A})$ such that $\mathcal{I}_{\mathcal{A}} \models S(\boldsymbol{a})$ iff $S(\boldsymbol{a}) \in \mathcal{A}$, for any ground atom $S(\boldsymbol{a})$. As $\mathcal{A}$ is arbitrary, this definition implies, in particular, that the rewriting must be *constant-free*. If $\boldsymbol{q}'(\boldsymbol{x})$ is a positive existential formula—that is, $\boldsymbol{q}'(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$ with φ constructed from atoms (possibly with equality) using $\land$ and $\lor$ only—we call it a *PE-rewriting of* $\boldsymbol{Q}(\boldsymbol{x})$. A PE-rewriting whose matrix φ is a disjunction of conjunctions is known as a *UCQ-rewriting*; if φ takes the form $\land\lor\land\lor$ we call it a Π_4-*rewriting*. The size $|\boldsymbol{q}'|$ of $\boldsymbol{q}'$ is the number of symbols in it.

We also consider rewritings in the form of nonrecursive datalog queries. Recall [Abiteboul et al. 1995] that a *datalog program*, Π, is a finite set of Horn clauses $\forall \boldsymbol{x}\, (\gamma_1 \land \dots \land \gamma_m \to \gamma_0)$, where each γ_i is an atom $P(x_1, \dots, x_l)$ with $x_i \in \boldsymbol{x}$. The atom γ_0 is the *head* of the clause, and $\gamma_1, \dots, \gamma_m$ its (possibly empty) *body*. A predicate S *depends* on S' in Π if Π has a clause with S in the head and S' in the body; Π is *nonrecursive* if this dependence relation is acyclic.

Let $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ, Π a constant-free nonrecursive program, and $G(\boldsymbol{x})$ a predicate. The pair $\boldsymbol{q}'(\boldsymbol{x}) = (\Pi, G(\boldsymbol{x}))$ is an *NDL-rewriting of* $\boldsymbol{Q}$ if, for any data instance $\mathcal{A}$ and any tuple $\boldsymbol{a}$ in $\mathsf{ind}(\mathcal{A})$, we have $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\Pi(\mathcal{I}_{\mathcal{A}}) \models G(\boldsymbol{a})$, where $\Pi(\mathcal{I}_{\mathcal{A}})$ is the structure with domain $\mathsf{ind}(\mathcal{A})$ obtained by closing $\mathcal{I}_{\mathcal{A}}$ under the clauses in Π. Every PE-rewriting can clearly be represented as an NDL-rewriting of linear size.

Remark 2.3. As defined, FO- and PE-rewritings are not necessarily domain-independent queries, while NDL-rewritings are not necessarily safe [Abiteboul et al. 1995]. For example, $(x = x)$ is a PE-rewriting of the OMQ $(\{\forall x\, P(x, x)\}, P(x, x))$, and the program $(\{\top \to A(x)\}, A(x))$ is an NDL-rewriting of the OMQ $(\{\top \to A(x)\}, A(x))$. Rewritings can easily be made domain-independent and safe by relativising their variables to the predicates in the data signature (relational schema). For instance, if this signature is $\{A, P\}$, then a domain-independent relativisation of $(x = x)$ is the PE-rewriting $\big(A(x) \lor \exists y\, P(x, y) \lor \exists y\, P(y, x)\big) \land (x = x)$. Note that if we exclude from *OWL 2 QL* reflexivity statements and axioms with $\top$ on the left-hand side, then rewritings are guaranteed to be domain-independent, and no relativisation is required. In any case, rewritings are always interpreted under the *active domain semantics* adopted in databases; see (4).

As mentioned in the introduction, the *OWL 2 QL* profile of *OWL 2* was designed to ensure FO-rewritability of all OMQs with ontologies in the profile or, equivalently, OMQ answering in AC^0 for data complexity. It should be clear, however, that for the OBDA approach to work in practice, the rewritings of OMQs must be of 'reasonable shape and size'. Indeed, it was observed experimentally [Calvanese et al. 2011] and also established theoretically [Kikot et al. 2012b] that sometimes the rewritings are prohibitively large—exponentially-large in the size of the original CQ, to be more precise. These facts imply that, in the context of OBDA, we should actually be interested not in arbitrary but in *polynomial-size* rewritings. In complexity-theoretic terms, the focus should not only be on the data complexity of OMQ answering, which is an appropriate measure for database query evaluation (where queries are indeed usually small) [Vardi 1982], but also on the combined complexity that takes into account the contribution of ontologies and queries.

3. TREE-WITNESS REWRITING

Now we define one particular rewriting of *OWL 2 QL* OMQs that will play a key role in the succinctness and complexity analysis later on in the paper. This rewriting is a modification of the tree-witness PE-rewriting originally introduced by Kikot et al. [2012a] (cf. [Lutz 2008; Kontchakov et al. 2010; König et al. 2015b] for rewritings based on similar ideas).

We begin with two simple observations that will help us remove unneeded clutter from the definitions. Every *OWL 2 QL* ontology $\mathcal{T}$ consists of two parts: $\mathcal{T}^-$, which contains all the sentences with $\bot$, and the remainder, $\mathcal{T}^+$, which is consistent with every data instance. For any $\psi(\boldsymbol{z}) \to \bot$ in $\mathcal{T}^-$, consider the Boolean CQ $\exists \boldsymbol{z}\, \psi(\boldsymbol{z})$. It is not hard to see that, for any OMQ $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ and data instance $\mathcal{A}$, a tuple $\boldsymbol{a}$ is a certain answer to $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ over $\mathcal{A}$ iff either $\mathcal{T}^+, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ or $\mathcal{T}^+, \mathcal{A} \models \exists \boldsymbol{z}\, \psi(\boldsymbol{z})$, for some $\psi(\boldsymbol{z}) \to \bot$ in $\mathcal{T}^-$; see [Calì et al. 2012a] for more details. Thus, from now on we will assume that, in all our ontologies $\mathcal{T}$, the 'negative' part $\mathcal{T}^-$ is empty, and so they are *consistent* with all data instances.

The second observation will allow us to restrict the class of data instances we need to consider when rewriting OMQs. In general, if we only require condition (4) to hold for any data instance $\mathcal{A}$ from some class $\mathfrak{A}$, then we call $\boldsymbol{q}'(\boldsymbol{x})$ a *rewriting of* $\boldsymbol{Q}(\boldsymbol{x})$ *over* $\mathfrak{A}$. Such classes of data instances can be defined, for example, by the integrity constraints in the database schema and the mapping [Rodriguez-Muro et al. 2013]. We say that a data instance $\mathcal{A}$ is *complete*[9] for an ontology $\mathcal{T}$ if $\mathcal{T}, \mathcal{A} \models S(\boldsymbol{a})$ implies $S(\boldsymbol{a}) \in \mathcal{A}$, for any ground atom $S(\boldsymbol{a})$ with $\boldsymbol{a}$ from $\mathsf{ind}(\mathcal{A})$. The following proposition means that from now on we will only consider rewritings over complete data instances.

PROPOSITION 3.1. *If $\boldsymbol{q}'(\boldsymbol{x})$ is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ over complete data instances, then there is an NDL-rewriting $\boldsymbol{q}''(\boldsymbol{x})$ of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances with $|\boldsymbol{q}''| \le |\boldsymbol{q}'| \cdot |\mathcal{T}|$. A similar result holds for PE- and FO-rewritings.*

PROOF. Let $(\Pi, G(\boldsymbol{x}))$ be an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances. Denote by Π^* the result of replacing each predicate S in Π with a fresh predicate S^*. Define Π' to be the union of Π^* and the following clauses for predicates in Π:

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

(the empty body is denoted by $\top$). It is readily seen that $(\Pi', G^*(\boldsymbol{x}))$ is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances. The cases of PE- and FO-rewritings are similar except that we replace $A(x)$ and $P(x,y)$ with

$$\bigvee_{\mathcal{T} \models \tau(x) \to A(x)} \tau(x) \qquad \text{and} \qquad \bigvee_{\mathcal{T} \models \varrho(x,y) \to P(x,y)} \varrho(x,y) \quad \lor \bigvee_{\mathcal{T} \models P(x,x)} (x = y),$$

respectively (the empty disjunction is, by definition, $\bot$). □

[9]Rodriguez-Muro et al. [2013] used the term 'H-completeness'; see also [König et al. 2015a].

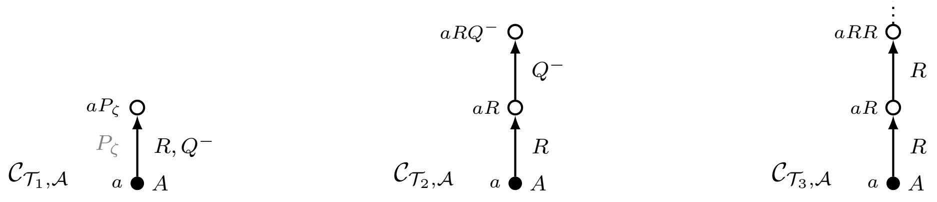

$\mathcal{C}_{\mathcal{T}_1,\mathcal{A}}$
aP_ζ
P_ζ
R, Q^-
a
A
$\mathcal{C}_{\mathcal{T}_2,\mathcal{A}}$
aRQ^-
Q^-
aR
R
a
A
$\mathcal{C}_{\mathcal{T}_3,\mathcal{A}}$
aRR
R
aR
R
a
A

Fig. 3. Canonical models in Example 3.2.

As is well-known [Abiteboul et al. 1995], every pair $(\mathcal{T}, \mathcal{A})$ of an ontology $\mathcal{T}$ and data instance $\mathcal{A}$ possesses a *canonical model* (or *chase*) $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(\boldsymbol{a})$, for all CQs $\boldsymbol{q}(\boldsymbol{x})$ and $\boldsymbol{a}$ in $\mathsf{ind}(\mathcal{A})$. In our proofs, we use the following definition of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, where without loss of generality we assume that $\mathcal{T}$ does not contain binary predicates P such that $\mathcal{T} \models \forall x, y\, P(x, y)$. Indeed, occurrences of such P in $\mathcal{T}$ can be replaced by $\top$ and occurrences of $P(x, y)$ in CQs can simply be removed without changing certain answers over any data instance (provided that x and y occur in the remainder of the query).

The domain $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ of the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ consists of $\mathsf{ind}(\mathcal{A})$ and the *witnesses*, or *labelled nulls*, introduced by the existential quantifiers in (the normalisation of) $\mathcal{T}$. More precisely, the labelled nulls in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ are finite words of the form $w = a\varrho_1 \dots \varrho_n$ $(n \geq 1)$ such that

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

Every individual name $a \in \mathsf{ind}(\mathcal{A})$ is interpreted in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ by itself, and unary and binary predicates are interpreted as follows: for any $u, v \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$,

– $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models A(u)$ iff either $u \in \mathsf{ind}(\mathcal{A})$ and $\mathcal{T}, \mathcal{A} \models A(u)$, or $u = w\varrho$, for some w and ϱ with $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$;
– $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models P(u, v)$ iff one of the following holds: (*i*) $u, v \in \mathsf{ind}(\mathcal{A})$ and $\mathcal{T}, \mathcal{A} \models P(u, v)$; (*ii*) $u = v$ and $\mathcal{T} \models P(x, x)$; (*iii*) $v = u\varrho$ and $\mathcal{T} \models \varrho(x, y) \to P(x, y)$; (*iv*) $u = v\varrho^-$ and $\mathcal{T} \models \varrho(x, y) \to P(x, y)$.

Example 3.2. Consider the following ontologies:

$$\begin{aligned}
\mathcal{T}_1 &= \{\ A(x) \to \exists y\, \big(R(x, y) \land Q(y, x)\big)\ \},\\
\mathcal{T}_2 &= \{\ A(x) \to \exists y\, R(x, y),\ \ \exists x\, R(x, y) \to \exists z\, Q(z, y)\ \},\\
\mathcal{T}_3 &= \{\ A(x) \to \exists y\, R(x, y),\ \ \exists x\, R(x, y) \to \exists z\, R(y, z)\ \}.
\end{aligned}$$

The canonical models of $(\mathcal{T}_i, \mathcal{A})$, for $\mathcal{A} = \{A(a)\}$, $i = 1, 2, 3$, are shown in Fig. 3, where $\zeta(x) = \exists y\, (R(x, y) \land Q(y, x))$ and P_ζ is the corresponding normalisation predicate. When depicting canonical models, we use ● for constants and ○ for labelled nulls.

For any ontology $\mathcal{T}$ and any formula $\tau(x)$ given by (1), we denote by $\mathcal{C}_{\mathcal{T}}^{\tau(a)}$ the canonical model of $(\mathcal{T} \cup \{A(x) \to \tau(x)\}, \{A(a)\})$, for a fresh unary predicate A. We say that $\mathcal{T}$ is *of depth* k, $1 \leq k < \omega$, if (*i*) there is no ϱ with $\mathcal{T} \models \varrho(x, x)$, (*ii*) at least one of the $\mathcal{C}_{\mathcal{T}}^{\tau(a)}$ contains a word $a\varrho_1 \dots \varrho_k$, but (*iii*) none of the $\mathcal{C}_{\mathcal{T}}^{\tau(a)}$ has such a word of greater length. Thus, $\mathcal{T}_1$ in Example 3.2 is of depth 1, $\mathcal{T}_2$ of depth 2, while $\mathcal{T}_3$ is not of any finite depth.

Ontologies of infinite depth generate infinite canonical models. However, *OWL 2 QL* has the *polynomial derivation depth property* (PDDP) in the sense that there is a polynomial p such that, for any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, data instance $\mathcal{A}$ and $\boldsymbol{a}$ in $\mathsf{ind}(\mathcal{A})$, we have $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\boldsymbol{q}(\boldsymbol{a})$ holds in the sub-model of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ whose domain consists of words of the form $a\varrho_1 \dots \varrho_n$ with $n \leq p(|\boldsymbol{Q}|)$ [Johnson and Klug 1982; Calì et al. 2012a]. (In general, the bounded derivation depth property of an ontology language is a necessary and sufficient condition of FO-rewritability [Gottlob et al. 2014].)

We call a set $\Omega_{\boldsymbol{Q}}$ of words of the form $w = \varrho_1 \dots \varrho_n$ *fundamental for* $\boldsymbol{Q}$ if, for any $\mathcal{A}$ and $\boldsymbol{a}$ in $\mathsf{ind}(\mathcal{A})$, we have $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\boldsymbol{q}(\boldsymbol{a})$ holds in the sub-model of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ with the domain $\{aw \mid a \in \mathsf{ind}(\mathcal{A}),\ w \in \Omega_{\boldsymbol{Q}}\}$. We say that a class $\mathcal{Q}$ of OMQs has the *polynomial fundamental set property* (PFSP) if there is a polynomial p such that every $\boldsymbol{Q} \in \mathcal{Q}$ has a fundamental set $\Omega_{\boldsymbol{Q}}$ with $|\Omega_{\boldsymbol{Q}}| \leq p(|\boldsymbol{Q}|)$. The class of *all* OMQs (even with ontologies of finite depth and tree-shaped CQs) does not have the PFSP [Kikot et al. 2012b]. On the other hand, it should be clear that the class of OMQs with ontologies of bounded depth does enjoy the PFSP. A less trivial example is given by the following theorem, which is an immediate consequence of Theorem 3.7 below:

THEOREM 3.3. *The class of OMQs whose ontologies do not contain axioms of the form* $\varrho(x, y) \to \varrho'(x, y)$ (*and syntactic sugar* (2)) *enjoys the PFSP.*

We are now in a position to define the tree-witness PE-rewriting of *OWL 2 QL* OMQs. Suppose we are given an OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$. For a pair $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ of disjoint sets of variables in $\boldsymbol{q}$, with $\mathfrak{t}_\mathsf{i} \subseteq \boldsymbol{y}$[10] and $\mathfrak{t}_\mathsf{i} \neq \emptyset$ ($\mathfrak{t}_\mathsf{r}$ can be empty), set

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

If $\boldsymbol{q}_\mathfrak{t}$ is a minimal subset of $\boldsymbol{q}$ for which there is a homomorphism $h\colon \boldsymbol{q}_\mathfrak{t} \to \mathcal{C}_\mathcal{T}^{\tau(a)}$ such that $\mathfrak{t}_\mathsf{r} = h^{-1}(a)$ and $\boldsymbol{q}_\mathfrak{t}$ contains every atom of $\boldsymbol{q}$ with at least one variable from $\mathfrak{t}_\mathsf{i}$, then we call $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ a *tree witness for Q generated by* τ (and *induced by* h). Observe that if $\mathfrak{t}_\mathsf{r} = \emptyset$ then $\boldsymbol{q}_\mathfrak{t}$ is a connected component of $\boldsymbol{q}$; in this case we call $\mathfrak{t}$ *detached*. Note also that the same tree witness $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ can be generated by different τ. Now, we set

$$\mathsf{tw}_\mathfrak{t}(\mathfrak{t}_\mathsf{r}) \;=\; \exists z\, \big(\bigwedge_{x \in \mathfrak{t}_\mathsf{r}} (x = z) \;\;\wedge \bigvee_{\mathfrak{t} \text{ generated by } \tau} \tau(z) \big). \tag{5}$$

The variables in $\mathfrak{t}_\mathsf{i}$ do not occur in $\mathsf{tw}_\mathfrak{t}$ and are called *internal*. The variables in $\mathfrak{t}_\mathsf{r}$, if any, are called *root variables*. Note that no answer variable in $\boldsymbol{q}(\boldsymbol{x})$ can be internal. The length $|\mathsf{tw}_\mathfrak{t}|$ of $\mathsf{tw}_\mathfrak{t}$ is $O(|\boldsymbol{Q}|)$. Tree witnesses $\mathfrak{t}$ and $\mathfrak{t}'$ are *conflicting* if $\boldsymbol{q}_\mathfrak{t} \cap \boldsymbol{q}_{\mathfrak{t}'} \neq \emptyset$. Denote by $\Theta_{\boldsymbol{Q}}$ the set of tree witnesses for $\boldsymbol{Q}(\boldsymbol{x})$. A subset $\Theta \subseteq \Theta_{\boldsymbol{Q}}$ is *independent* if no pair of distinct tree witnesses in it is conflicting. Let $\boldsymbol{q}_\Theta = \bigcup_{\mathfrak{t} \in \Theta} \boldsymbol{q}_\mathfrak{t}$. The following PE-formula is called the *tree-witness rewriting of* $\boldsymbol{Q}(\boldsymbol{x})$ *over complete data instances*:

$$\boldsymbol{q}_\mathsf{tw}(\boldsymbol{x}) \;=\; \bigvee_{\Theta \subseteq \Theta_{\boldsymbol{Q}} \text{ independent}} \exists \boldsymbol{y}\, \big(\bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta} S(\boldsymbol{z}) \;\wedge\; \bigwedge_{\mathfrak{t} \in \Theta} \mathsf{tw}_\mathfrak{t}(\mathfrak{t}_\mathsf{r}) \big). \tag{6}$$

Remark 3.4. As the normalisation predicates P_ζ cannot occur in data instances, we can omit from (5) all the disjuncts with P_ζ. For the same reason, the tree witnesses generated only by concepts with normalisation predicates will be ignored in the sequel.

Example 3.5. Consider the OMQ $\boldsymbol{Q}(x_1, x_2) = (\mathcal{T}, \boldsymbol{q}(x_1, x_2))$ with

$$\mathcal{T} \;=\; \big\{ A_1(x) \to \underbrace{\exists y\, \big(R_1(x, y) \wedge Q(x, y)\big)}_{\zeta_1(x)},\; A_2(x) \to \underbrace{\exists y\, \big(R_2(x, y) \wedge Q(y, x)\big)}_{\zeta_2(x)} \big\},$$

$$\boldsymbol{q}(x_1, x_2) \;=\; \exists y_1, y_2\, \big(R_1(x_1, y_1) \wedge Q(y_2, y_1) \wedge R_2(x_2, y_2)\big).$$

The CQ $\boldsymbol{q}$ is shown in Fig. 4 alongside $\mathcal{C}_\mathcal{T}^{A_1(a)}$ and $\mathcal{C}_\mathcal{T}^{A_2(a)}$. When depicting CQs, we use ● for answer variables and ○ for existentially quantified variables. There are two tree

[10]We (ab)use set-theoretic notation for lists and, for example, write $\mathfrak{t}_\mathsf{i} \subseteq \boldsymbol{y}$ to say that every element of $\mathfrak{t}_\mathsf{i}$ is an element of $\boldsymbol{y}$.

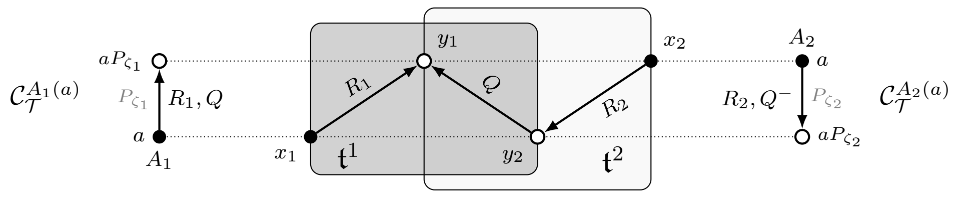


Fig. 4. Tree witnesses in Example 3.5.

witnesses, t^1 and t^2, for $\boldsymbol{Q}$ with

$$\boldsymbol{q}_{\mathsf{t}^1} = \{\, R_1(x_1,y_1), Q(y_2,y_1)\,\} \quad \text{and} \quad \boldsymbol{q}_{\mathsf{t}^2} = \{\, Q(y_2,y_1), R_2(x_2,y_2)\,\}$$

shown in Fig. 4 by the dark and light shading, respectively. The tree witness $\mathsf{t}^1 = (\mathsf{t}^1_\mathsf{r}, \mathsf{t}^1_\mathsf{i})$ with $\mathsf{t}^1_\mathsf{r} = \{x_1, y_2\}$ and $\mathsf{t}^1_\mathsf{i} = \{y_1\}$ is generated by $A_1(x)$, which gives

$$\mathsf{tw}_{\mathsf{t}^1}(x_1,y_2) = \exists z\,\big(A_1(z) \land (x_1 = z) \land (y_2 = z)\big).$$

(Recall that although t^1 is also generated by $\exists y\, P_{\zeta_1}(y,z)$, we do not include it in the disjunction in $\mathsf{tw}_{\mathsf{t}^1}$ because P_{ζ_1} cannot occur in data instances.) Symmetrically, the tree witness t^2 gives

$$\mathsf{tw}_{\mathsf{t}^2}(x_2,y_1) = \exists z\,\big(A_2(z) \land (x_2 = z) \land (y_1 = z)\big).$$

As t^1 and t^2 are conflicting, $\Theta_{\boldsymbol{Q}}$ contains three independent subsets: $\emptyset$, $\{\mathsf{t}^1\}$ and $\{\mathsf{t}^2\}$. Thus, we obtain the following tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}(x_1,x_2)$ of $\boldsymbol{Q}$ over complete data instances:

$$\exists y_1,y_2\,\big[\big(R_1(x_1,y_1) \land Q(y_2,y_1) \land R_2(x_2,y_2)\big) \lor \big(\mathsf{tw}_{\mathsf{t}^1} \land R_2(x_2,y_2)\big) \lor \big(R_1(x_1,y_1) \land \mathsf{tw}_{\mathsf{t}^2}\big)\big].$$

THEOREM 3.6 ([KIKOT ET AL. 2012A]). *For any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, any data instance $\mathcal{A}$, which is complete for $\mathcal{T}$, and any tuple $\boldsymbol{a}$ from $\mathsf{ind}(\mathcal{A})$, we have $\mathcal{T},\mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{I}_\mathcal{A} \models \boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{a})$. In other words, $\boldsymbol{q}_{\mathsf{tw}}$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.*

Intuitively, for every homomorphism $h\colon \boldsymbol{q}(\boldsymbol{a}) \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$, the sub-CQs of $\boldsymbol{q}$ mapped by h to sub-models of the form $\mathcal{C}_{\mathcal{T}}^{\tau(a)}$ define an independent set Θ of tree witnesses; see Fig. 5. Conversely, if Θ is such a set, then the homomorphisms corresponding to the tree witnesses in Θ can be pieced together into a homomorphism from $\boldsymbol{q}(\boldsymbol{a})$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$—provided that the $S(\boldsymbol{z})$ from $\boldsymbol{q} \setminus \boldsymbol{q}_\Theta$ and the $\mathsf{tw}_\mathsf{t}(\mathsf{t}_\mathsf{r})$ for $\mathsf{t} \in \Theta$ hold in $\mathcal{I}_\mathcal{A}$.

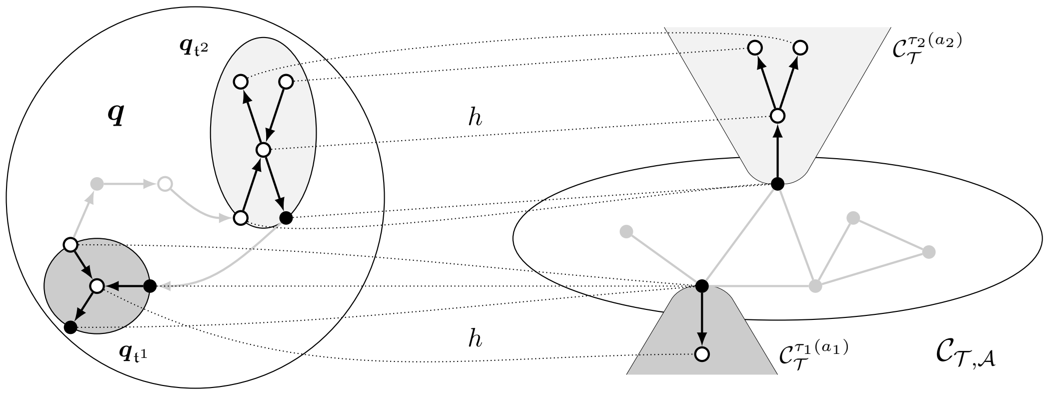

$\boldsymbol{q}_{\mathsf{t}^2}$
$\boldsymbol{q}$
h
$\mathcal{C}_{\mathcal{T}}^{\tau_2(a_2)}$
h
$\boldsymbol{q}_{\mathsf{t}^1}$
$\mathcal{C}_{\mathcal{T}}^{\tau_1(a_1)}$
$\mathcal{C}_{\mathcal{T},\mathcal{A}}$

Fig. 5. Tree-witness rewriting.

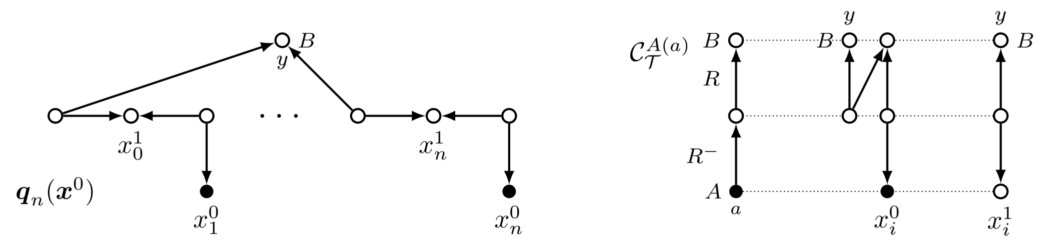


Fig. 6. The query $\boldsymbol{q}_n(\boldsymbol{x}^0)$ (all edges are labelled by R), the canonical model $\mathcal{C}_{\mathcal{T}}^{A(a)}$ (the normalisation predicates are not shown) and two ways of mapping a branch of the query to the canonical model in Example 3.8.

The *size* of the tree-witness PE-rewriting $\boldsymbol{q}_{\mathsf{tw}}$ depends on the number of tree witnesses in the given OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ and, more importantly, on the cardinality of $\Theta_{\boldsymbol{Q}}$ as we have $|\boldsymbol{q}_{\mathsf{tw}}| = O(2^{|\Theta_{\boldsymbol{Q}}|} \cdot |\boldsymbol{Q}|^2)$ with $|\Theta_{\boldsymbol{Q}}| \leq 3^{|\boldsymbol{q}|}$.

THEOREM 3.7. *OMQs $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$, in which $\mathcal{T}$ does not contain axioms of the form $\varrho(x, y) \to \varrho'(x, y)$ (and syntax sugar (2)), have at most $3|\boldsymbol{q}|$ tree witnesses.*

PROOF. As observed above, there can be only one detached tree witness for each connected component of $\boldsymbol{q}$. As $\mathcal{T}$ has no axioms of the form $\varrho(x, y) \to \varrho'(x, y)$, any two points in $\mathcal{C}_{\mathcal{T}}^{\tau(a)}$ can be R-related for at most one R, and so no point can have more than one R-successor, for any R. It follows that, for every atom $P(x, y)$ in $\boldsymbol{q}$, there can be at most one tree witness $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ with $P(x, y) \in \boldsymbol{q}_\mathfrak{t}$, $x \in \mathfrak{t}_\mathsf{r}$ and $y \in \mathfrak{t}_\mathsf{i}$ ($P^-(y, x)$ may give another tree witness). $\square$

OMQs with arbitrary axioms can have *exponentially many* tree witnesses:

Example 3.8. Consider the OMQ $\boldsymbol{Q}_n = (\mathcal{T}, \boldsymbol{q}_n(\boldsymbol{x}^0))$, where

$$\mathcal{T} = \big\{A(x) \to \exists y\,\big(R(y, x) \land \exists z\,(R(y, z) \land B(z))\big)\big\},$$
$$\boldsymbol{q}_n(\boldsymbol{x}^0) = \exists y, \boldsymbol{y}^1, \boldsymbol{x}^1, \boldsymbol{y}^2\,\Big(B(y) \;\land \bigwedge_{1 \leq k \leq n} \big(R(y_k^1, y) \land R(y_k^1, x_k^1) \land R(y_k^2, x_k^1) \land R(y_k^2, x_k^0)\big)\Big)$$

and $\boldsymbol{x}^i$ and $\boldsymbol{y}^i$ denote vectors of n variables x_k^i and y_k^i, for $1 \leq k \leq n$, respectively. The CQ is shown in Fig. 6 alongside the canonical model $\mathcal{C}_{\mathcal{T}}^{A(a)}$. OMQ $\boldsymbol{Q}_n$ has at least 2^n tree witnesses: for any $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \{0, 1\}^n$, there is a tree witness $(\mathfrak{t}_\mathsf{r}^{\boldsymbol{\alpha}}, \mathfrak{t}_\mathsf{i}^{\boldsymbol{\alpha}})$ with $\mathfrak{t}_\mathsf{r}^{\boldsymbol{\alpha}} = \{x_k^{\alpha_k} \mid 1 \leq k \leq n\}$. Note, however, that the tree-witness rewriting of $\boldsymbol{Q}_n$ can be equivalently transformed into the following *polynomial-size* PE-rewriting:

$$\boldsymbol{q}_n(\boldsymbol{x}^0) \;\lor\; \exists z\,\Big[A(z) \land \bigwedge_{1 \leq i \leq n} \big((x_i^0 = z) \lor \exists y\,(R(y, x_i^0) \land R(y, z))\big)\Big].$$

If any two tree witnesses for an OMQ $\boldsymbol{Q}$ are compatible in the sense that either they are non-conflicting or one is included in the other, then $\boldsymbol{q}_{\mathsf{tw}}$ can be equivalently transformed to the PE-rewriting

$$\exists \boldsymbol{y} \bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q}} \big(S(\boldsymbol{z}) \;\lor \bigvee_{\mathfrak{t} \in \Theta_{\boldsymbol{Q}} \text{ with } S(\boldsymbol{z}) \in \boldsymbol{q}_\mathfrak{t}} \mathsf{tw}_\mathfrak{t}(\mathfrak{t}_\mathsf{r})\big) \qquad \text{of size } O(|\Theta_{\boldsymbol{Q}}| \cdot |\boldsymbol{Q}|^2).$$

We now analyse transformations of this kind in the setting of Boolean functions.

4. OMQ REWRITINGS AS BOOLEAN FUNCTIONS

For any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, we define Boolean functions $f_{\boldsymbol{Q}}^{\triangledown}$ and $f_{\boldsymbol{Q}}^{\vartriangle}$ such that:

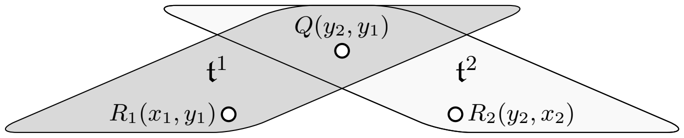


Fig. 7. The hypergraph $\mathcal{H}(\boldsymbol{Q})$ for $\boldsymbol{Q}$ from Example 3.5.

– if $f^{\triangledown}_{\boldsymbol{Q}}$ is computed by a Boolean formula (monotone formula or monotone circuit) Φ, then $\boldsymbol{Q}$ has an FO- (respectively, PE- or NDL-) rewriting of size $O(|\Phi| \cdot |\boldsymbol{Q}|)$;

– if $\boldsymbol{q}'$ is an FO- (PE- or NDL-) rewriting of $\boldsymbol{Q}$, then $f^{\vartriangle}_{\boldsymbol{Q}}$ is computed by a Boolean formula (respectively, monotone formula or monotone circuit) of size $O(|\boldsymbol{q}'|)$.

We remind the reader (for details see, e.g., [Arora and Barak 2009; Jukna 2012]) that an *n-ary Boolean function*, for $n \geq 1$, is any function from $\{0,1\}^n$ to $\{0,1\}$. A Boolean function f is *monotone* if $f(\boldsymbol{\alpha}) \leq f(\boldsymbol{\beta})$ for all $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$, where $\leq$ is the component-wise $\leq$ on vectors of $\{0,1\}$. A *Boolean circuit*, $\boldsymbol{C}$, is a directed acyclic graph whose vertices are called *gates*. Each gate is labelled with a propositional variable, a constant 0 or 1, or with NOT, AND or OR. Gates labelled with variables and constants have in-degree 0 and are called *inputs*; NOT-gates have in-degree 1, while AND- and OR-gates have in-degree 2 (unless otherwise specified). One of the gates in $\boldsymbol{C}$ is distinguished as the *output gate*. Given an assignment $\boldsymbol{\alpha} \in \{0,1\}^n$ to the variables, we compute the value of each gate in $\boldsymbol{C}$ under $\boldsymbol{\alpha}$ as usual in Boolean logic. The *output $\boldsymbol{C}(\boldsymbol{\alpha})$ of $\boldsymbol{C}$ on $\boldsymbol{\alpha} \in \{0,1\}^n$* is the value of the output gate. We usually assume that the gates $g_1, \dots, g_m$ of $\boldsymbol{C}$ are ordered in such a way that $g_1, \dots, g_n$ are input gates; each gate g_i, for $i \geq n$, gets inputs from gates $g_{j_1}, \dots, g_{j_k}$ with $j_1, \dots, j_k < i$, and g_m is the output gate. We say that $\boldsymbol{C}$ *computes* an n-ary Boolean function f if $\boldsymbol{C}(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha})$ for all $\boldsymbol{\alpha} \in \{0,1\}^n$. The *size* $|\boldsymbol{C}|$ of $\boldsymbol{C}$ is the number of gates in $\boldsymbol{C}$. A circuit is *monotone* if it contains only inputs, AND- and OR-gates. *Boolean formulas* can be thought of as circuits in which every logic gate has at most one outgoing edge. Any monotone circuit computes a monotone function, and any monotone Boolean function can be computed by a monotone circuit.

4.1. Hypergraph Functions

Let $H = (V, E)$ be a hypergraph with *vertices* $v \in V$ and *hyperedges* $e \in E \subseteq 2^V$. A subset $E' \subseteq E$ is said to be *independent* if $e \cap e' = \emptyset$, for any distinct $e, e' \in E'$. The set of vertices that occur in the hyperedges of E' is denoted by $V_{E'}$. For each vertex $v \in V$ and each hyperedge $e \in E$, we take propositional variables p_v and p_e, respectively. The *hypergraph function f_H for H* is given by the monotone Boolean formula

$$f_H \quad = \bigvee_{E' \text{ independent}} \Big(\bigwedge_{v \in V \setminus V_{E'}} p_v \;\wedge\; \bigwedge_{e \in E'} p_e \Big). \tag{7}$$

The tree-witness PE-rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ defines a hypergraph whose vertices are the atoms of $\boldsymbol{q}$ and hyperedges are the sets $\boldsymbol{q}_{\mathfrak{t}}$, where $\mathfrak{t}$ is a tree witness for $\boldsymbol{Q}$. We denote this hypergraph by $\mathcal{H}(\boldsymbol{Q})$ and call $f_{\mathcal{H}(\boldsymbol{Q})}$ the *tree-witness hypergraph function for $\boldsymbol{Q}$*. To simplify notation, we write $f^{\triangledown}_{\boldsymbol{Q}}$ instead of $f_{\mathcal{H}(\boldsymbol{Q})}$. Note that formula (7) defining $f^{\triangledown}_{\boldsymbol{Q}}$ is obtained from rewriting (6) by regarding the atoms $S(\boldsymbol{z})$ in $\boldsymbol{q}$ and tree-witness formulas $\mathsf{tw}_{\mathfrak{t}}$ as propositional variables. We denote these variables by $p_{S(\boldsymbol{z})}$ and $p_{\mathfrak{t}}$ (rather than p_v and p_e), respectively.

Example 4.1. For the OMQ $\boldsymbol{Q}$ in Example 3.5, the hypergraph $\mathcal{H}(\boldsymbol{Q})$ has 3 vertices (one for each atom in the query) and 2 hyperedges (one for each tree witness) shown in Fig. 7. The tree-witness hypergraph function for $\boldsymbol{Q}$ is as follows:

$$f^{\triangledown}_{\boldsymbol{Q}} \;=\; \big(p_{R_1(x_1,y_1)} \land p_{Q(y_2,y_1)} \land p_{R_2(x_2,y_2)}\big) \lor \big(p_{\mathfrak{t}^1} \land p_{R_2(x_2,y_2)}\big) \lor \big(p_{R_1(x_1,y_1)} \land p_{\mathfrak{t}^2}\big).$$

Suppose the function $f^{\triangledown}_{\boldsymbol{Q}}$ for an OMQ $\boldsymbol{Q}(\boldsymbol{x})$ is computed by a Boolean formula Φ. Consider the first-order formula $\Phi^*(\boldsymbol{x})$ obtained by replacing each $p_{S(\boldsymbol{z})}$ in Φ with $S(\boldsymbol{z})$, each $p_{\mathfrak{t}}$ with $\mathsf{tw}_{\mathfrak{t}}$, and adding the appropriate prefix $\exists \boldsymbol{y}$. By comparing (7) and (6), we see that $\Phi^*(\boldsymbol{x})$ is an FO-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over data instances that are complete over $\mathcal{T}$. This gives claim (i) of the following theorem:

THEOREM 4.2. (i) *If $f^{\triangledown}_{\boldsymbol{Q}}$ is computed by a (monotone) Boolean formula Φ, then there is a (PE-) FO-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ of size $O(|\Phi| \cdot |\boldsymbol{Q}|)$.*

(ii) *If $f^{\triangledown}_{\boldsymbol{Q}}$ is computed by a monotone Boolean circuit $\boldsymbol{C}$, then there is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ of size $O(|\boldsymbol{C}| \cdot |\boldsymbol{Q}|)$.*

PROOF. (ii) Let $\mathfrak{t}^1, \dots, \mathfrak{t}^l$ be tree witnesses for $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, where $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y} \bigwedge_{i=1}^{n} S_i(\boldsymbol{z}_i)$. We assume that the gates $g_1, \dots, g_n$ of $\boldsymbol{C}$ are the inputs $p_{S_1(\boldsymbol{z}_1)}, \dots, p_{S_n(\boldsymbol{z}_n)}$ for the atoms, the gates $g_{n+1}, \dots, g_{n+l}$ are the inputs $p_{\mathfrak{t}^1}, \dots, p_{\mathfrak{t}^l}$ for the tree witnesses and $g_{n+l+1}, \dots, g_m$ are AND- and OR-gates. Denote by Π the following NDL-program, where $\boldsymbol{z} = \boldsymbol{x} \cup \boldsymbol{y}$:

– $S_i(\boldsymbol{z}_i) \to G_i(\boldsymbol{z})$, for $0 < i \le n$;
– $\tau(u) \to G_{i+m}(\boldsymbol{z}[\mathfrak{t}^j_\mathsf{r}/u])$, for $0 < j \le l$ and τ generating $\mathfrak{t}^j$, where $\boldsymbol{z}[\mathfrak{t}^j_\mathsf{r}/u]$ is the result of replacing each $z \in \mathfrak{t}^j_\mathsf{r}$ in $\boldsymbol{z}$ with u;
– $\begin{cases} G_j(\boldsymbol{z}) \land G_k(\boldsymbol{z}) \to G_i(\boldsymbol{z}), & \text{if } g_i = g_j \land g_k, \\ G_j(\boldsymbol{z}) \to G_i(\boldsymbol{z}) \text{ and } G_k(\boldsymbol{z}) \to G_i(\boldsymbol{z}), & \text{if } g_i = g_j \lor g_k, \end{cases}$ for $n + l < i \le m$;
– $G_m(\boldsymbol{z}) \to G(\boldsymbol{x})$.

It is not hard to see that $(\Pi, G(\boldsymbol{x}))$ is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$. □

Thus, the problem of constructing polynomial-size rewritings of OMQs reduces to finding polynomial-size (monotone) formulas or monotone circuits for the corresponding functions $f^{\triangledown}_{\boldsymbol{Q}}$. Note, however, that $f^{\triangledown}_{\boldsymbol{Q}}$ contains a variable $p_{\mathfrak{t}}$ for every tree witness $\mathfrak{t}$, which makes this reduction useless for OMQs with exponentially many tree witnesses. To be able to deal with such OMQs, we slightly modify the tree-witness rewriting.

Suppose $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ is a tree witness for $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ induced by a homomorphism $h\colon \boldsymbol{q}_{\mathfrak{t}} \to \mathcal{C}^{\tau(a)}_{\mathcal{T}}$. We say that $\mathfrak{t}$ is *ϱ-initiated* if $h(z)$ is of the form $a\varrho w$, for every (equivalently, some) variable $z \in \mathfrak{t}_\mathsf{i}$. For such ϱ, we define a formula $\varrho^*(x)$ by taking the disjunction of $\tau(x)$ with $\mathcal{T} \models \tau(x) \to \exists y\, \varrho(x, y)$. Again, the disjunction includes only those $\tau(x)$ that do not contain normalisation predicates (even though ϱ itself can be one).

Example 4.3. Consider the OMQ $\boldsymbol{Q}(x) = (\mathcal{T}, \boldsymbol{q}(x))$ with

$$\mathcal{T} = \big\{\, \exists y\, Q(x,y) \to \exists y\, P(x,y),\;\; P(x,y) \to R(x,y) \,\big\} \quad \text{and} \quad \boldsymbol{q}(x) = \exists y\, R(x,y).$$

As shown in Fig. 8, the tree witness $\mathfrak{t} = (\{x\}, \{y\})$ for $\boldsymbol{Q}(x)$ is generated by $\exists y\, Q(x,y)$, $\exists y\, P(x,y)$ and $\exists y\, R(x,y)$; it is also P- and R-initiated, but not Q-initiated. We have:

$$P^*(x) = \exists y\, Q(x,y) \lor \exists y\, P(x,y) \text{ and } R^*(x) = \exists y\, Q(x,y) \lor \exists y\, P(x,y) \lor \exists y\, R(x,y).$$

The modified tree-witness rewriting for $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, denoted $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$, is obtained by replacing (5) in (6) with the formula

$$\mathsf{tw}'_{\mathfrak{t}}(\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i}) \;=\; \bigwedge_{R(z,z') \in \boldsymbol{q}_{\mathfrak{t}}} (z = z') \;\;\land \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \; \bigwedge_{z \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}} \varrho^*(z). \tag{5'}$$

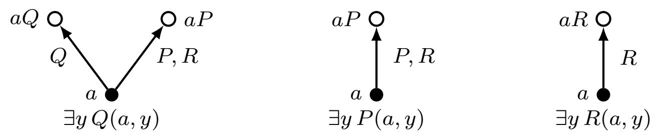


Fig. 8. Canonical models in Example 4.3.

Note that unlike (5), this formula contains the variables in both $\mathfrak{t}_\mathsf{i}$ and $\mathfrak{t}_\mathsf{r}$, which must be equal under every satisfying assignment. We associate with $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ the monotone Boolean function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ given by the formula obtained from (7) by replacing each variable p_v with the respective $p_{S(\boldsymbol{z})}$, for $S(\boldsymbol{z}) \in \boldsymbol{q}$, and each variable p_e with the formula

$$\bigwedge_{R(z,z') \in \boldsymbol{q}_\mathfrak{t}} p_{z=z'} \quad \wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \; \bigwedge_{z \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}} p_{\varrho^*(z)}, \tag{8}$$

for the respective tree witness $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ for $\boldsymbol{Q}(\boldsymbol{x})$, where $p_{z=z'}$ and $p_{\varrho^*(z)}$ are propositional variables. Clearly, the number of variables in $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is *polynomial* in $|\boldsymbol{Q}|$.

Example 4.4. For the OMQ $\boldsymbol{Q}(x)$ in the Example 3.5, we have:

$$\begin{aligned} f^{\blacktriangledown}_{\boldsymbol{Q}} \;=\; & \big(p_{R_1(x_1,y_1)} \wedge p_{Q(y_2,y_1)} \wedge p_{R_2(x_2,y_2)}\big) \;\vee \\ & \quad \Big(\big(p_{x_1=y_1} \wedge p_{y_2=y_1} \;\wedge \bigwedge_{z \in \{x_1,y_1,y_2\}} p_{P^*_{\zeta_1}(z)}\big) \;\wedge\; p_{R_2(x_2,y_2)}\Big) \;\vee \\ & \qquad \Big(p_{R_1(x_1,y_1)} \;\wedge\; \big(p_{y_2=y_1} \wedge p_{x_2=y_2} \;\wedge \bigwedge_{z \in \{y_1,y_2,x_2\}} p_{P^*_{\zeta_2}(z)}\big)\Big). \end{aligned}$$

The proof of the following theorem is given in Appendix A:

THEOREM 4.5. (*i*) *For any OMQ $\boldsymbol{Q}(\boldsymbol{x})$, the formulas $\boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{x})$ and $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ are equivalent, and so $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ is a PE-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.*
(*ii*) *Theorem 4.2 continues to hold for $f^{\triangledown}_{\boldsymbol{Q}}$ replaced by $f^{\blacktriangledown}_{\boldsymbol{Q}}$.*

Finally, we observe that although $f^{\triangledown}_{\boldsymbol{Q}}$ and $f^{\blacktriangledown}_{\boldsymbol{Q}}$ are defined by exponential-size formulas, each of these functions can be computed by a nondeterministic polynomial algorithm (in the number of propositional variables). Indeed, given truth-values for the $p_{S(\boldsymbol{z})}$ and $p_\mathfrak{t}$ in $f^{\triangledown}_{\boldsymbol{Q}}$, guess a set Θ of at most $|\boldsymbol{q}|$ tree witnesses and check whether (*i*) Θ is independent, (*ii*) $p_\mathfrak{t} = 1$ for all $\mathfrak{t} \in \Theta$, and (*iii*) every $S(\boldsymbol{z})$ with $p_{S(\boldsymbol{z})} = 0$ belongs to some $\mathfrak{t} \in \Theta$. The function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is computed similarly except that, in (*ii*), we check whether the polynomial-size formula (8) is true under the given truth-values for every $\mathfrak{t} \in \Theta$.

4.2. Primitive Evaluation Functions

To obtain lower bounds on the size of rewritings, we associate with every OMQ $\boldsymbol{Q}$ a third Boolean function $f^{\vartriangle}_{\boldsymbol{Q}}$ that describes the result of evaluating $\boldsymbol{Q}$ on data instances with a single individual constant. Let $\gamma \in \{0,1\}^n$ be a vector assigning the truth-value $\gamma(S_i)$ to each unary or binary predicate S_i in $\boldsymbol{Q}$. We associate with γ the data instance

$$\mathcal{A}(\gamma) \;=\; \big\{\, A_i(a) \mid \gamma(A_i) = 1 \,\big\} \;\cup\; \big\{\, P_i(a,a) \mid \gamma(P_i) = 1 \,\big\}$$

and set $f^{\vartriangle}_{\boldsymbol{Q}}(\gamma) = 1$ iff $\mathcal{T}, \mathcal{A}(\gamma) \models \boldsymbol{q}(\boldsymbol{a})$, where $\boldsymbol{a}$ is the $|\boldsymbol{x}|$-tuple of as. We call $f^{\vartriangle}_{\boldsymbol{Q}}$ the *primitive evaluation function* for $\boldsymbol{Q}$.

THEOREM 4.6. (*i*) *If $\boldsymbol{q}'$ is a (PE-) FO-rewriting of $\boldsymbol{Q}$, then $f^{\vartriangle}_{\boldsymbol{Q}}$ can be computed by a (monotone) Boolean formula of size $O(|\boldsymbol{q}'|)$.*

(*ii*) *If $\boldsymbol{q}'$ is an NDL-rewriting of $\boldsymbol{Q}$, then $f^{\vartriangle}_{\boldsymbol{Q}}$ can be computed by a monotone Boolean circuit of size* $O(|\boldsymbol{q}'|)$.

PROOF. (*i*) Let $\boldsymbol{q}'$ be an FO-rewriting of $\boldsymbol{Q}$. We eliminate the quantifiers in $\boldsymbol{q}'$ by replacing each subformula of the form $\exists x\, \psi(x)$ and $\forall x\, \psi(x)$ in $\boldsymbol{q}'$ with $\psi(a)$. We then replace each $a = a$ with $\top$ and each atom of the form $A(a)$ and $P(a, a)$ with the corresponding propositional variable. The resulting Boolean formula clearly computes $f^{\vartriangle}_{\boldsymbol{Q}}$. If $\boldsymbol{q}'$ is a PE-rewriting of $\boldsymbol{Q}$, then the result is a monotone Boolean formula computing $f^{\vartriangle}_{\boldsymbol{Q}}$.

(*ii*) If (Π, G) is an NDL-rewriting of $\boldsymbol{Q}$, then we replace all variables in Π with a and then perform the replacement described in (*i*). We now turn the resulting propositional NDL-program Π' into a monotone circuit computing $f^{\vartriangle}_{\boldsymbol{Q}}$. For every (propositional) variable p occurring in the head of a rule in Π', we take an appropriate number of OR-gates whose output is p and inputs are the bodies of the rules with head p; for every such body, we introduce an appropriate number of AND-gates whose inputs are the propositional variables in the body, or, if the body is empty, we take the gate for constant 1. $\square$

4.3. Hypergraph Programs

We introduced hypergraph functions as Boolean abstractions of the tree-witness rewritings. Our next aim is to define a model of computation for these functions.

A *hypergraph program* (HGP) P is a hypergraph $H = (V, E)$ each of whose vertices is labelled by 0, 1 or a literal over a list $p_1, \dots, p_n$ of propositional variables. (As usual, a *literal*, $\boldsymbol{l}$, is a propositional variable or its negation.) An *input* for P is a tuple $\boldsymbol{\alpha} \in \{0, 1\}^n$, which is regarded as a valuation for $p_1, \dots, p_n$. The output $P(\boldsymbol{\alpha})$ of P on $\boldsymbol{\alpha}$ is 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 $\boldsymbol{\alpha}$. We say that P *computes* an n-ary Boolean function f if $f(\boldsymbol{\alpha}) = P(\boldsymbol{\alpha})$, for all $\boldsymbol{\alpha} \in \{0, 1\}^n$. An HGP is *monotone* if its vertex labels do not have negated variables. The *size* $|P|$ of an HGP P is the size $|H|$ of the underlying hypergraph $H = (V, E)$, which is $|V| + |E|$.

The following observation shows that monotone HGPs capture the computational power of hypergraph functions. We remind the reader that a *subfunction* of a Boolean function f is obtained from f using two operations: (1) fixing some of its variables to 0 or 1, and (2) renaming (in particular, identifying) some of the variables in f. A hypergraph H is said to be of *degree at most* d if every vertex in it belongs to at most d hyperedges; H is of *degree* d if every vertex in it belongs to exactly d hyperedges.

PROPOSITION 4.7. (*i*) *Any monotone HGP based on a hypergraph H computes a subfunction of the hypergraph function f_H.*

(*ii*) *For any hypergraph H of degree at most d, there is a monotone HGP of size $O(|H|)$ that computes f_H and such that its hypergraph is of degree at most* $\max(2, d)$.

PROOF. To show (*i*), it is enough to replace the vertex variables p_v in f_H by the corresponding vertex labels of the given HGP and fix all the edge variables p_e to 1.

For (*ii*), given a hypergraph $H = (V, E)$, we label each $v \in V$ by the variable p_v. For each $e \in E$, we add a fresh vertex a_e labelled by 1 and a fresh vertex b_e labelled by p_e; then we create a new hyperedge $e' = \{a_e, b_e\}$ and add a_e to the hyperedge e. We claim that the resulting HGP P computes f_H. Indeed, for any input $\boldsymbol{\alpha}$ with $\boldsymbol{\alpha}(p_e) = 0$, we have to include the edge e' into the cover, and so cannot include the edge e itself. Thus, $P(\boldsymbol{\alpha}) = 1$ iff there is an independent set E of hyperedges with $\boldsymbol{\alpha}(p_e) = 1$, for all $e \in E$, covering all zeros of the variables p_v. $\square$

In some cases, it will be convenient to use *generalised HGPs* that allow hypergraph vertices to be labelled by conjunctions $\bigwedge_i \boldsymbol{l}_i$ of literals $\boldsymbol{l}_i$. The following proposition shows that this generalisation does not increase the computational power of HGPs.

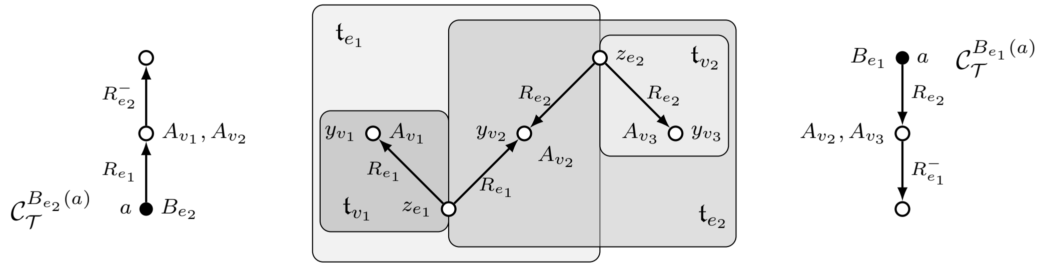


Fig. 9. The OMQ $\boldsymbol{Q}_H$ for H from Example 4.1 and its tree witnesses.

PROPOSITION 4.8. *For every generalised HGP P over n variables, there is an HGP P' computing the same function and such that $|P'| \leq n \cdot |P|$.*

PROOF. To construct P', we split every vertex v of P labelled with $\bigwedge_{i=1}^{k} \boldsymbol{l}_i$ into k new vertices $v_1, \dots, v_k$ and label v_i with $\boldsymbol{l}_i$, for $1 \leq i \leq k$ (without loss of generality, we can assume that $\boldsymbol{l}_i$ and $\boldsymbol{l}_j$ have distinct variables for $i \neq j$); each hyperedge containing v will now contain all the v_i. It is easy to see that $P(\boldsymbol{\alpha}) = P'(\boldsymbol{\alpha})$, for any input $\boldsymbol{\alpha}$. Since $k \leq n$, we have $|P'| \leq n \cdot |P|$. □

5. OMQS, HYPERGRAPHS AND MONOTONE HYPERGRAPH PROGRAMS

We now establish a correspondence between the structure of OMQs and hypergraphs.

5.1. OMQs with ontologies of depth 2

To begin with, we show that every hypergraph $H = (V, E)$ can be represented by a polynomial-size OMQ $\boldsymbol{Q}_H = (\mathcal{T}, \boldsymbol{q})$ with $\mathcal{T}$ of depth 2. With every vertex $v \in V$ we associate a unary predicate A_v, and with every hyperedge $e \in E$ a unary predicate B_e and a binary predicate R_e. We define $\mathcal{T}$ to be the set of the following axioms, for $e \in E$:

$$B_e(x) \rightarrow \exists y \big[\bigwedge_{e \cap e' \neq \emptyset,\ e \neq e'} R_{e'}(x, y) \land \bigwedge_{v \in e} A_v(y) \land \exists z\, R_e(z, y) \big].$$

Clearly, $\mathcal{T}$ is of depth 2. We also take the Boolean CQ $\boldsymbol{q}$ with variables y_v, for $v \in V$, and z_e, for $e \in E$:

$$\boldsymbol{q} = \big\{ A_v(y_v) \mid v \in V \big\} \cup \big\{ R_e(z_e, y_v) \mid v \in e, \text{ for } v \in V \text{ and } e \in E \big\}.$$

Example 5.1. Consider again the hypergraph from Example 4.1, which we now denote by $H = (V, E)$ with $V = \{v_1, v_2, v_3\}$, $E = \{e_1, e_2\}$, $e_1 = \{v_1, v_2\}$ and $e_2 = \{v_2, v_3\}$. The CQ $\boldsymbol{q}$ and the canonical models $\mathcal{C}_{\mathcal{T}}^{B_{e_i}(a)}$, for $i = 1, 2$, are shown in Fig. 9 along with four tree witnesses for $\boldsymbol{Q}_H$ (as explained in Remark 3.4, we ignore the two extra tree witnesses generated only by normalisation predicates).

It is not hard to see that the number of tree witnesses for $\boldsymbol{Q}_H$ does not exceed $|H|$. Indeed, all the tree witnesses for $\boldsymbol{Q}_H$ fall into two types:

$\mathfrak{t}_v = (\mathfrak{t}_\mathsf{i}, \mathfrak{t}_\mathsf{r})$ with $\mathfrak{t}_\mathsf{r} = \{z_e \mid v \in e\}$ and $\mathfrak{t}_\mathsf{i} = \{y_v\}$, for $v \in V$ that belong to a single $e \in E$;
$\mathfrak{t}_e = (\mathfrak{t}_\mathsf{i}, \mathfrak{t}_\mathsf{r})$ with $\mathfrak{t}_\mathsf{r} = \{z_{e'} \mid e \cap e' \neq \emptyset, e \neq e'\}$ and $\mathfrak{t}_\mathsf{i} = \{z_e\} \cup \{y_v \mid v \in e\}$, for $e \in E$.

We call a hypergraph H' a *subgraph* of a hypergraph $H = (V, E)$ if H' can be obtained from H by (i) removing some of its hyperedges and (ii) removing some of its vertices from both V and the hyperedges in E.

THEOREM 5.2. (i) *Any hypergraph H is isomorphic to a subgraph of $\mathcal{H}(\boldsymbol{Q}_H)$.*

(*ii*) *Any monotone HGP P based on a hypergraph H computes a subfunction of the primitive evaluation function $f^{\vartriangle}_{\boldsymbol{Q}_H}$.*

PROOF. (*i*) An isomorphism between H and a subgraph of $\mathcal{H}(\boldsymbol{Q}_H)$ can be established by the map $v \mapsto A_v(y_v)$, for $v \in V$, and $e \mapsto \boldsymbol{q}_{\mathfrak{t}_e}$, for $e \in E$.

(*ii*) Suppose that P is based on a hypergraph $H = (V, E)$. Given an input $\boldsymbol{\alpha}$ for P, we define an assignment $\boldsymbol{\gamma}$ for the predicates in $\boldsymbol{Q}_H = (\mathcal{T}, \boldsymbol{q})$ by taking $\boldsymbol{\gamma}(A_v)$ to be the value of the label of v under $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}(B_e) = 1$, $\boldsymbol{\gamma}(R_e) = 1$ (and of course $\boldsymbol{\gamma}(P_\zeta) = 0$, for all normalisation predicates P_ζ). By the definition of $\mathcal{T}$, for each $e \in E$, the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$ contains labelled nulls w_e and w'_e such that

$$\mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})} \models \bigwedge_{e \cap e' \neq \emptyset,\ e \neq e'} R_{e'}(a, w_e) \ \wedge\ \bigwedge_{v \in e} A_v(w_e) \ \wedge\ R_e(w'_e, w_e).$$

We now show that $P(\boldsymbol{\alpha}) = 1$ iff $f^{\vartriangle}_{\boldsymbol{Q}_H}(\boldsymbol{\gamma}) = 1$ (iff $\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma}) \models \boldsymbol{q}$). Suppose $P(\boldsymbol{\alpha}) = 1$, that is, there is an independent subset $E' \subseteq E$ such that the label of each $v \notin \bigcup E'$ evaluates to 1 under $\boldsymbol{\alpha}$. Then the map $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$ defined by taking

$$h(z_e) = \begin{cases} w'_e, & \text{if } e \in E', \\ a, & \text{otherwise,} \end{cases} \qquad h(y_v) = \begin{cases} w_e, & \text{if } v \in e \in E', \\ a, & \text{otherwise} \end{cases}$$

is a homomorphism witnessing $\mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})} \models \boldsymbol{q}$, whence $f^{\vartriangle}_{\boldsymbol{Q}_H}(\boldsymbol{\gamma}) = 1$.

Conversely, if $f^{\vartriangle}_{\boldsymbol{Q}_H}(\boldsymbol{\gamma}) = 1$ then there is a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$. For any hyperedge $e \in E$, there are only two options for $h(z_e)$: either a or w'_e. It follows that the set $E' = \{e \in E \mid h(z_e) = w'_e\}$ is independent and covers all zeros. Indeed, if $v \notin \bigcup E'$ then $h(y_v) = a$, and so the label of v evaluates to 1 under $\boldsymbol{\alpha}$ because $A_v(y_v) \in \boldsymbol{q}$. □

Next, we establish a tight correspondence between hypergraphs of degree at most 2 and OMQs with ontologies of depth 1.

5.2. Hypergraphs of Degree 2 and OMQs with Ontologies of Depth 1

THEOREM 5.3. *For any OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ with $\mathcal{T}$ of depth 1, the hypergraph $\mathcal{H}(\boldsymbol{Q})$ is of degree at most 2 and $|\mathcal{H}(\boldsymbol{Q})| \leq 2|\boldsymbol{q}|$.*

PROOF. We have to show that every atom in $\boldsymbol{q}$ belongs to at most two $\boldsymbol{q}_{\mathfrak{t}}$, $\mathfrak{t} \in \Theta_{\boldsymbol{Q}}$. Suppose $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ is a tree witness for $\boldsymbol{Q}$ and $y \in \mathfrak{t}_\mathsf{i}$. Since $\mathcal{T}$ is of depth 1, $\mathfrak{t}_\mathsf{i} = \{y\}$ and $\mathfrak{t}_\mathsf{r}$ consists of all the variables in $\boldsymbol{q}$ adjacent to y in the Gaifman graph $G_{\boldsymbol{q}}$ of $\boldsymbol{q}$. Thus, different tree witnesses have different internal variables y. An atom of the form $A(u) \in \boldsymbol{q}$ is in $\boldsymbol{q}_{\mathfrak{t}}$ iff $u = y$. An atom of the form $P(u, v) \in \boldsymbol{q}$ is in $\boldsymbol{q}_{\mathfrak{t}}$ iff either $u = y$ or $v = y$. Therefore, $P(u, v) \in \boldsymbol{q}$ can only be covered by the tree witness with internal u and by the tree witness with internal v. □

Conversely, we show now that any hypergraph H of degree 2 is isomorphic to $\mathcal{H}(\boldsymbol{S}_H)$, for some OMQ $\boldsymbol{S}_H = (\mathcal{T}, \boldsymbol{q})$ with $\mathcal{T}$ of depth 1. We can assume that $H = (V, E)$ comes with two fixed maps $i_1, i_2\colon V \to E$ such that for every $v \in V$, we have $i_1(v) \neq i_2(v)$, $v \in i_1(v)$ and $v \in i_2(v)$. For any $v \in V$, we fix a binary predicate R_v, and let the ontology $\mathcal{T}$ in $\boldsymbol{S}_H$ contain the following axioms, for $e \in E$:

$$A_e(x) \ \to\ \exists y\, \big[\bigwedge_{\substack{v \in V \\ i_1(v) = e}} R_v(y, x) \ \ \wedge \bigwedge_{\substack{v \in V \\ i_2(v) = e}} R_v(x, y) \big].$$

Clearly, $\mathcal{T}$ is of depth 1. The Boolean CQ $\boldsymbol{q}$ contains variables z_e, for $e \in E$, and is defined by taking

$$\boldsymbol{q} \ = \ \big\{ R_v(z_{i_1(v)}, z_{i_2(v)}) \mid v \in V \big\}.$$

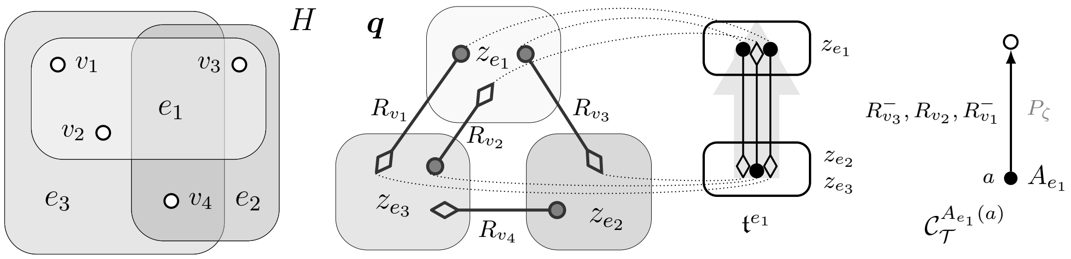


Fig. 10. Hypergraph H in Example 5.4, its CQ $\boldsymbol{q}$, tree witness $\mathfrak{t}^{e_1}$ for $\boldsymbol{S}_H$ and canonical model $\mathcal{C}_{\mathcal{T}}^{A_{e_1}(a)}$.

Example 5.4. Suppose that $H = (V, E)$, where $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3\}$ and $e_1 = \{v_1, v_2, v_3\}$, $e_2 = \{v_3, v_4\}$, $e_3 = \{v_1, v_2, v_4\}$. Let

$$i_1 \colon v_1 \mapsto e_1, \quad v_2 \mapsto e_3, \quad v_3 \mapsto e_1, \quad v_4 \mapsto e_2,$$
$$i_2 \colon v_1 \mapsto e_3, \quad v_2 \mapsto e_1, \quad v_3 \mapsto e_2, \quad v_4 \mapsto e_3.$$

The hypergraph H and the query $\boldsymbol{q}$ are shown in Fig. 10: each R_{v_k} is represented by an edge, $i_1(v_k)$ is indicated by the circle-shaped end of the edge and $i_2(v_k)$ by the diamond-shaped end of the edge; the e_j are shown as large grey squares. In this case,

$$\boldsymbol{q} \;=\; \exists z_{e_1}, z_{e_2}, z_{e_3}\, \big(R_{v_1}(z_{e_1}, z_{e_3}) \land R_{v_2}(z_{e_3}, z_{e_1}) \land R_{v_3}(z_{e_1}, z_{e_2}) \land R_{v_4}(z_{e_2}, z_{e_3})\big)$$

and $\mathcal{T}$ consists of the following axioms:

$$A_{e_1}(x) \to \exists y\, \big[R_{v_1}(y, x) \land R_{v_2}(x, y) \land R_{v_3}(y, x)\big],$$
$$A_{e_2}(x) \to \exists y\, \big[R_{v_3}(x, y) \land R_{v_4}(y, x)\big],$$
$$A_{e_3}(x) \to \exists y\, \big[R_{v_1}(x, y) \land R_{v_2}(y, x) \land R_{v_4}(x, y)\big].$$

The canonical model $\mathcal{C}_{\mathcal{T}}^{A_{e_1}(a)}$ is shown on the right-hand side of Fig. 10. Note that each z_e determines the tree witness $\mathfrak{t}^e$ with $\boldsymbol{q}_{\mathfrak{t}^e} = \{R_v(z_{i_1(v)}, z_{i_2(v)}) \mid v \in e\}$; distinct $\mathfrak{t}^e$ and $\mathfrak{t}^{e'}$ are conflicting iff $e \cap e' \neq \emptyset$. It follows that H is isomorphic to $\mathcal{H}(\boldsymbol{S}_H)$.

THEOREM 5.5. (*i*) *Any hypergraph H of degree 2 is isomorphic to $\mathcal{H}(\boldsymbol{S}_H)$.*
(*ii*) *Any monotone HGP P based on a hypergraph H of degree 2 computes a subfunction of the primitive evaluation function $f^{\vartriangle}_{\boldsymbol{S}_H}$.*

PROOF. (*i*) We show that the map $g \colon v \mapsto R_v(z_{i_1(v)}, z_{i_2(v)})$ is an isomorphism between H and $\mathcal{H}(\boldsymbol{S}_H)$. By the definition of $\boldsymbol{S}_H$, g is a bijection between V and the atoms of $\boldsymbol{q}$. For any $e \in E$, there is a tree witness $\mathfrak{t}^e = (\mathfrak{t}^e_{\mathsf{r}}, \mathfrak{t}^e_{\mathsf{i}})$ generated by $A_e(x)$ with $\mathfrak{t}^e_{\mathsf{i}} = \{z_e\}$ and $\mathfrak{t}^e_{\mathsf{r}} = \{z_{e'} \mid e \cap e' \neq \emptyset, e \neq e'\}$, and $\boldsymbol{q}_{\mathfrak{t}^e}$ consists of the $g(v)$, for $v \in e$. Conversely, every tree witness $\mathfrak{t}$ for $\boldsymbol{S}_H$ contains $z_e \in \mathfrak{t}_{\mathsf{i}}$, for some $e \in E$, and so $\boldsymbol{q}_{\mathfrak{t}} = \{g(v) \mid v \in e\}$.

(*ii*) By Proposition 4.7 (*i*), P computes a subfunction of f_H. Thus, it suffices to show that f_H is a subfunction of $f^{\vartriangle}_{\boldsymbol{S}_H}$. Let $H = (V, E)$ be a hypergraph of degree 2. For any $\boldsymbol{\alpha} \in \{0,1\}^{|H|}$, we define $\boldsymbol{\gamma}$ by taking $\boldsymbol{\gamma}(R_v) = \boldsymbol{\alpha}(p_v)$ for $v \in V$, $\boldsymbol{\gamma}(A_e) = \boldsymbol{\alpha}(p_e)$ for $e \in E$ (and $\boldsymbol{\gamma}(P_\zeta) = 0$ for all normalisation predicates P_ζ). We prove that $f_H(\boldsymbol{\alpha}) = 1$ iff $\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma}) \models \boldsymbol{q}$. By the definition of $\mathcal{T}$, for each $e \in E$ with $A_e(a) \in \mathcal{A}(\boldsymbol{\gamma})$ or, equivalently, $\boldsymbol{\alpha}(p_e) = 1$, the canonical model $\mathcal{C}_{\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma})}$ contains a labelled null w_e such that

$$\mathcal{C}_{\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma})} \models \bigwedge_{\substack{v \in V\\ i_1(v) = e}} R_v(w_e, a) \;\land\; \bigwedge_{\substack{v \in V\\ i_2(v) = e}} R_v(a, w_e).$$

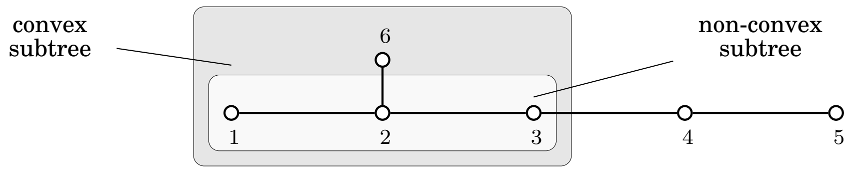


Fig. 11. Tree T in Example 5.6.

($\Rightarrow$) Let E' be an independent subset of E such that $\bigwedge_{v \in V \setminus V_{E'}} p_v \wedge \bigwedge_{e \in E'} p_e$ is true on $\boldsymbol{\alpha}$. Define $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$ by taking $h(z_e) = a$ if $e \notin E'$ and $h(z_e) = w_e$ otherwise. One can check that h is a homomorphism, and so $\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma}) \models \boldsymbol{q}$.

($\Leftarrow$) Given a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$, we show that $E' = \{e \in E \mid h(z_e) \neq a\}$ is independent. Indeed, if $e, e' \in E'$ and $v \in e \cap e'$, then h sends one variable of the R_v-atom to the labelled null w_e and the other end to $w_{e'}$, which is impossible. We claim that $f_H(\boldsymbol{\alpha}) = 1$. Indeed, for each $v \in V \setminus V_{E'}$, h sends both ends of the R_v-atom to a, and so $\boldsymbol{\alpha}(p_v) = 1$. For each $e \in E'$, we must have $h(z_e) = w_e$ because $h(z_e) \neq a$, and so $\boldsymbol{\alpha}(p_e) = 1$. It follows that $f_H(\boldsymbol{\alpha}) = 1$. □

5.3. Tree-Shaped OMQs and Tree Hypergraphs

We call an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ *tree-shaped* if the CQ $\boldsymbol{q}$ is tree-shaped. We now establish a close correspondence between tree-shaped OMQs and tree hypergraphs that are defined as follows.[11]

Suppose $T = (V_T, E_T)$ is an (undirected) tree. A leaf is a vertex of degree 1. A subtree $T' = (V'_T, E'_T)$ of T is said to be *convex* if, for any non-leaf vertex u in the subtree T', we have $\{u, v\} \in E'_T$ whenever $\{u, v\} \in E_T$. A hypergraph $H = (V, E)$ is called a *tree hypergraph* if there is a tree $T = (V_T, E_T)$ such that $V = E_T$ and every hyperedge $e \in E$ induces a convex subtree T_e of T. In this case, we call T the *underlying tree* of H. The *boundary* of a hyperedge e consists of all leaves of T_e; the interior of e is the set of non-leaves of T_e. A *tree hypergraph program* (THGP) is an HGP based on a tree hypergraph.

Example 5.6. Let T be the tree shown in Fig. 11. Any tree hypergraph with underlying tree T has the set of vertices $\{\{1,2\},\{2,3\},\{2,6\},\{3,4\},\{4,5\}\}$ (each vertex is an edge of T), and its hyperedges may include $\{\{1,2\},\{2,3\},\{2,6\}\}$ as the subtree of T induced by these edges is convex, but not $\{\{1,2\},\{2,3\}\}$.

THEOREM 5.7. *If an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ is tree-shaped, then $\mathcal{H}(\boldsymbol{Q})$ is isomorphic to a tree hypergraph. Furthermore, if $\boldsymbol{q}$ has at least one binary atom, then the number of leaves in the tree underlying $\mathcal{H}(\boldsymbol{Q})$ is the same as the number of leaves in $\boldsymbol{q}$.*

PROOF. The case when $\boldsymbol{q}$ has no binary atoms is trivial. Otherwise, let $G_{\boldsymbol{q}}$ be the Gaifman graph of $\boldsymbol{q}$ whose vertices u are labelled with the unary atoms $\xi(u)$ in $\boldsymbol{q}$ of the form $A(u)$ and $P(u,u)$, and whose edges $\{u, v\}$ are labelled with the atoms of the form $P(u,v)$ and $P'(v,u)$ in $\boldsymbol{q}$. We replace every edge $\{u,v\}$ labelled with $P_1(u'_1, v'_1), \dots, P_n(u'_n, v'_n)$, for $n \geq 2$, by a sequence of n edges forming a path from u to v and label them with $P_1(u'_1, v'_1), \dots, P_n(u'_n, v'_n)$, respectively. In the resulting tree, for every vertex u labelled with n unary atoms $\xi_1(u), \dots, \xi_n(u)$, for $n \geq 1$, we pick an edge $\{u, v\}$ labelled with some $P(u', v')$ and replace it by a sequence of $n+1$ edges forming a path from u to v and label them with $\xi_1(u), \dots, \xi_n(u), P(u', v')$, respectively. The resulting tree T has the same number of leaves as $\boldsymbol{q}$. It is readily checked that, for

[11]Our definition of tree hypergraph is a minor variant of the notion of (sub)tree hypergraph (aka hypertree) from graph theory [Flament 1978; Brandstädt et al. 1999; Bretto 2013].

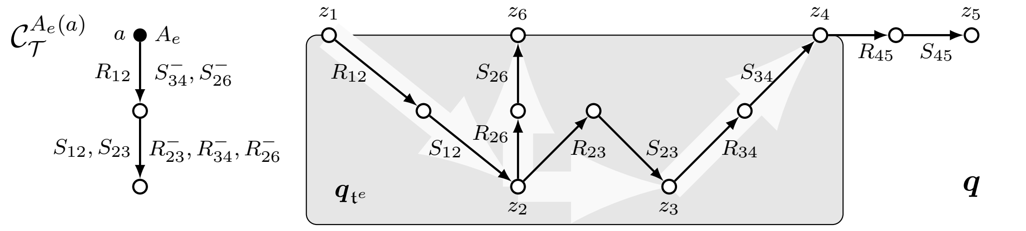


Fig. 12. The canonical model $\mathcal{C}_{\mathcal{T}}^{A_e(a)}$ and the query $\boldsymbol{q}$ (y_{ij} is the half-way point between z_i and z_j) for the tree hypergraph H in Example 5.8.

any tree witness $\mathfrak{t}$ for $\boldsymbol{Q}$, the set of edges in T labelled with atoms in $\boldsymbol{q}_{\mathfrak{t}}$ forms a convex subtree of T, which gives a tree hypergraph isomorphic to $\mathcal{H}(\boldsymbol{Q})$. □

Suppose $H = (V, E)$ is a tree hypergraph whose underlying tree $T = (V_T, E_T)$ has vertices $V_T = \{1, \dots, n\}$, for $n > 1$, and 1 is a leaf of T. Let $T^1 = (V_T, E_T^1)$ be the *directed* tree obtained from T by fixing 1 as the root and orienting the edges away from 1. We associate with H a tree-shaped OMQ $\boldsymbol{T}_H = (\mathcal{T}, \boldsymbol{q})$, in which $\boldsymbol{q}$ is the Boolean CQ

$$\boldsymbol{q} \;=\; \{\, R_{ij}(z_i, y_{ij}),\;\; S_{ij}(y_{ij}, z_j) \mid (i,j) \in E_T^1 \,\},$$

where the z_i, for $i \in V_T$, are the variables for vertices of the tree and the y_{ij}, for $(i,j) \in E_T^1$, are the variables for the edges of the tree. To define $\mathcal{T}$, suppose a hyperedge $e \in E$ induces a convex directed subtree $T_e = (V_e, E_e)$ of T^1 with root $r^e \in V_e$ and leaves $L_e \subseteq V_e$. Denote by $\mathcal{T}$ the ontology that contains the following axiom, for each $e \in E$:

$$A_e(x) \;\to\; \exists y\, \Big[\bigwedge_{(i,j)\in E_e,\; i=r^e} R_{r^e j}(x,y) \;\wedge \bigwedge_{(i,j)\in E_e,\; j\in L_e} S_{ij}(y,x) \;\wedge$$
$$\exists z\, \Big(\bigwedge_{(i,j)\in E_e,\; i\neq r^e} R_{ij}(z,y) \;\wedge \bigwedge_{(i,j)\in E_e,\; j\notin L_e} S_{ij}(y,z)\Big)\Big].$$

Since T_e is convex, its root, r_e, has only one outgoing edge, (r^e, j), for some j, and so the first conjunct above contains a single atom, $R_{r^e j}(x,y)$. These axioms (together with convexity of hyperedges) ensure that $\boldsymbol{T}_H$ has a tree witness $\mathfrak{t}^e = (\mathfrak{t}^e_{\mathsf{r}}, \mathfrak{t}^e_{\mathsf{i}})$, for $e \in E$, with

$$\mathfrak{t}^e_{\mathsf{r}} \;=\; \{\, z_i \mid i \text{ is on the boundary of } e \,\},$$
$$\mathfrak{t}^e_{\mathsf{i}} \;=\; \{\, z_i \mid i \text{ is in the interior of } e \,\} \;\cup\; \{\, y_{ij} \mid (i,j) \in e \,\}.$$

Note that $\mathcal{T}$ is of depth 2, and $\boldsymbol{T}_H$ is of polynomial size in $|H|$.

Example 5.8. Let H be the tree hypergraph whose underlying tree is as in Example 5.6 with fixed root 1 and whose only hyperedge is $e = \{\{1,2\}, \{2,3\}, \{3,4\}, \{2,6\}\}$. The CQ $\boldsymbol{q}$ and the canonical model $\mathcal{C}_{\mathcal{T}}^{A_e(a)}$ for this H are shown in Fig. 12. Note the homomorphism from $\boldsymbol{q}_{\mathfrak{t}^e}$ into $\mathcal{C}_{\mathcal{T}}^{A_e(a)}$.

The proofs of the following results (which are THGP analogues of Theorem 5.2 and Propositions 4.7 (*ii*) and 4.8, respectively) are given in Appendices B and C:

THEOREM 5.9. (*i*) *Any tree hypergraph H is isomorphic to a subgraph of $\mathcal{H}(\boldsymbol{T}_H)$.*
(*ii*) *Any monotone THGP based on a tree hypergraph H computes a subfunction of the primitive evaluation function $f^{\vartriangle}_{\boldsymbol{T}_H}$.*

PROPOSITION 5.10. (*i*) *For any tree hypergraph H of degree at most d, there is a monotone THGP of size $O(|H|)$ that computes f_H and such that its hypergraph is of degree at most* $\max(2, d)$.

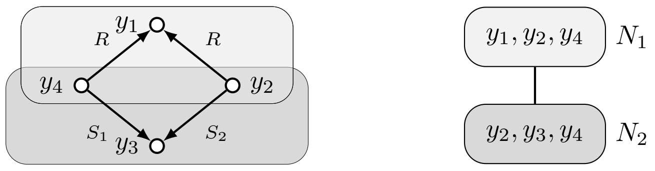


Fig. 13. Tree decomposition in Example 5.11.

(*ii*) *For every generalised THGP P over n variables, there is a THGP P' such that $|P'| \leq n \cdot |P|$ and P' has the same degree and number of leaves as P and computes the same function.*

5.4. OMQs with Bounded Treewidth CQs and Bounded Depth Ontologies

Recall (see, e.g., [Flum and Grohe 2006]) that a *tree decomposition* of an undirected graph $G = (V, E)$ is a pair (T, λ), where T is an (undirected) tree and λ a function from the set of nodes of T to 2^V such that

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

We call the set $\lambda(N) \subseteq V$ a *bag for* N. The *width of a tree decomposition* (T, λ) is the size of its largest bag minus one. The *treewidth* of G is the minimum width over all tree decompositions of G. The *treewidth of a CQ* $\boldsymbol{q}$ is the treewidth of its Gaifman graph $G_{\boldsymbol{q}}$.

Example 5.11. The Boolean CQ $\boldsymbol{q} = \{R(y_2, y_1),\ R(y_4, y_1),\ S_1(y_3, y_4),\ S_2(y_2, y_4)\}$ and its tree decomposition (T, λ) of width 2 are shown in Fig. 13, where T has two nodes, N_1 and N_2, connected by an edge, with bags $\lambda(N_1) = \{y_1, y_2, y_4\}$ and $\lambda(N_2) = \{y_2, y_3, y_4\}$.

Our aim in this section is to show that, for any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\boldsymbol{q}$ of bounded treewidth and a finite fundamental set $\Omega_{\boldsymbol{Q}}$, the modified tree-witness hypergraph function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ can be computed using a monotone THGP of size bounded by a polynomial in $|\boldsymbol{q}|$ and $|\Omega_{\boldsymbol{Q}}|$.

Let (T, λ) be a tree decomposition of $G_{\boldsymbol{q}}$ of width $m - 1$. In order to refer to the variables of $\boldsymbol{q}$, for each bag $\lambda(N)$, we fix an order of variables in the bag and define a injection $\nu_N \colon \lambda(N) \to \{1, \dots, m\}$ that gives the index of each z in $\lambda(N)$. A (*bag*) *type* is an m-tuple of the form $\boldsymbol{w} = (\boldsymbol{w}[1], \dots, \boldsymbol{w}[m])$, where each $\boldsymbol{w}[i] \in \Omega_{\boldsymbol{Q}}$. Intuitively, the ith component $\boldsymbol{w}[i]$ of $\boldsymbol{w}$ indicates that the ith variable in the bag is mapped to a domain element of the form $a\boldsymbol{w}[i]$ in the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. We say that a type $\boldsymbol{w}$ is *compatible with a node* N of T if the following conditions hold, for all $z, z' \in \lambda(N)$:

(1) if $A(z) \in \boldsymbol{q}$ and $\boldsymbol{w}[\nu_N(z)] \neq \varepsilon$, then $\boldsymbol{w}[\nu_N(z)] = w\varrho$ and $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$;
(2) if $P(z, z') \in \boldsymbol{q}$ and either $\boldsymbol{w}[\nu_N(z)] \neq \varepsilon$ or $\boldsymbol{w}[\nu_N(z')] \neq \varepsilon$, then
   – $\boldsymbol{w}[\nu_N(z)] = \boldsymbol{w}[\nu_N(z')]$ and $\mathcal{T} \models P(x, x)$, or
   – $\boldsymbol{w}[\nu_N(z')] = \boldsymbol{w}[\nu_N(z)]\varrho$ and $\mathcal{T} \models \varrho(x, y) \to P(x, y)$, or
   – $\boldsymbol{w}[\nu_N(z)] = \boldsymbol{w}[\nu_N(z')]\varrho^-$ and $\mathcal{T} \models \varrho(x, y) \to P(y, x)$.

Clearly, the type with all components equal to ε is compatible with any node N and corresponds to mapping all variables in $\lambda(N)$ to individuals in $\mathsf{ind}(\mathcal{A})$.

Example 5.12. Suppose $\mathcal{T} = \{\, A(x) \to \exists y\, R(x, y)\,\}$ and $\boldsymbol{q}$ is the same as in Example 5.11. Let ν_{N_1} and ν_{N_2} respect the order of the variables in the bags shown in Fig. 13. The only types compatible with N_1 are $(\varepsilon, \varepsilon, \varepsilon)$ and $(R, \varepsilon, \varepsilon)$, whereas the only type compatible with N_2 is $(\varepsilon, \varepsilon, \varepsilon)$.

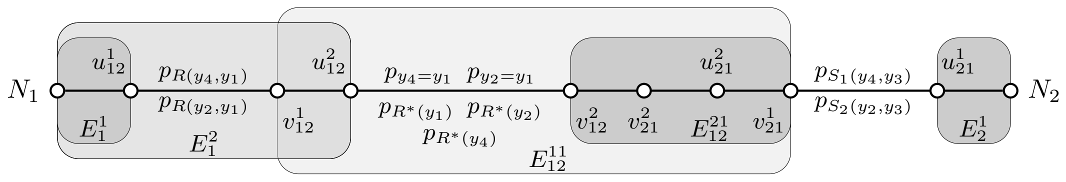


Fig. 14. THGP $P_{\boldsymbol{Q}}$ in Example 5.14: non-zero labels of vertices in $P_{\boldsymbol{Q}}$ are given on the edges of the tree.

Let $\boldsymbol{w}_1, \dots, \boldsymbol{w}_M$ be all the bag types for $\Omega_{\boldsymbol{Q}}$ ($M = |\Omega_{\boldsymbol{Q}}|^m$). Denote by T' the tree obtained from T by replacing every edge $\{N_i, N_j\}$ with the following sequence of edges:

$$\{N_i, u_{ij}^1\}, \qquad \{u_{ij}^k, v_{ij}^k\} \text{ and } \{v_{ij}^k, u_{ij}^{k+1}\}, \text{ for } 1 \le k < M, \qquad \{u_{ij}^M, v_{ij}^M\}, \qquad \{v_{ij}^M, v_{ji}^M\},$$
$$\{v_{ji}^M, u_{ji}^M\}, \qquad \{u_{ji}^{k+1}, v_{ji}^k\} \text{ and } \{v_{ji}^k, u_{ji}^k\}, \text{ for } 1 \le k < M, \qquad \{u_{ji}^1, N_j\},$$

for some fresh nodes u_{ij}^k, v_{ij}^k, u_{ji}^k and v_{ji}^k. We now define a generalised monotone THGP $P_{\boldsymbol{Q}}$ based on a hypergraph with the underlying tree T'. Denote by $[L]$ the set of nodes of the minimal convex subtree of T' containing all nodes of L. The hypergraph has the following hyperedges:

– $E_i^k = [N_i, u_{ij_1}^k, \dots, u_{ij_n}^k]$ if $N_{j_1}, \dots, N_{j_n}$ are the neighbours of N_i in T and $\boldsymbol{w}_k$ is compatible with N_i;
– $E_{ij}^{k\ell} = [v_{ij}^k, v_{ji}^\ell]$ if $\{N_i, N_j\}$ is an edge in T and $(\boldsymbol{w}_k, \boldsymbol{w}_\ell)$ is compatible with (N_i, N_j) in the sense that $\boldsymbol{w}_k[\nu_{N_i}(z)] = \boldsymbol{w}_\ell[\nu_{N_j}(z)]$, for all $z \in \lambda(N_i) \cap \lambda(N_j)$.

We label the vertices of the hypergraph—that is, the edges of T'—in the following way. The edges $\{N_i, u_{ij}^1\}$, $\{v_{ij}^k, u_{ij}^{k+1}\}$ and $\{v_{ij}^M, v_{ji}^M\}$ are labelled with 0, and every edge $\{u_{ij}^k, v_{ij}^k\}$ is labelled with the conjunction of the following variables:

– $p_{S(\boldsymbol{z})}$, whenever $S(\boldsymbol{z}) \in \boldsymbol{q}$, $\boldsymbol{z} \subseteq \lambda(N_i)$ and $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varepsilon$, for all $z \in \boldsymbol{z}$;
– $p_{\varrho^*(z)}$, whenever $A(z) \in \boldsymbol{q}$, $z \in \lambda(N_i)$ and $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varrho w$;
– $p_{\varrho^*(z)}$, $p_{\varrho^*(z')}$ and $p_{z=z'}$, whenever $R(z,z') \in \boldsymbol{q}$ (possibly with $z = z'$), $z, z' \in \lambda(N_i)$, and either $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varrho w$ or $\boldsymbol{w}_k[\nu_{N_i}(z')] = \varrho w$.

The following result is proved in Appendix D:

THEOREM 5.13. *For every OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ with a fundamental set $\Omega_{\boldsymbol{Q}}$ and with $\boldsymbol{q}$ of treewidth t, the generalised monotone THGP $P_{\boldsymbol{Q}}$ computes $f_{\boldsymbol{Q}}^{\blacktriangledown}$ and is of size polynomial in $|\boldsymbol{q}|$ and $|\Omega_{\boldsymbol{Q}}|^t$.*

Example 5.14. Let $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ be the OMQ from Example 5.12. As we have seen, there are only two types compatible with nodes in T: $\boldsymbol{w}_1 = (\varepsilon, \varepsilon, \varepsilon)$ and $\boldsymbol{w}_2 = (R, \varepsilon, \varepsilon)$. This gives us the generalised THGP $P_{\boldsymbol{Q}}$ shown in Fig. 14, where the omitted labels are all 0. To explain the meaning of $P_{\boldsymbol{Q}}$, suppose $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}$, for some data instance $\mathcal{A}$. Then there is a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$. This homomorphism defines the type of bag N_1, which can be either $\boldsymbol{w}_1$ (if $h(z) \in \mathsf{ind}(\mathcal{A})$ for all $z \in \lambda(N_1)$) or $\boldsymbol{w}_2$ (if $h(y_1) = aR$ for some $a \in \mathsf{ind}(\mathcal{A})$). These two cases are represented by the hyperedges $E_1^1 = [N_1, u_{12}^1]$ and $E_1^2 = [N_1, u_{12}^2]$. Since $\{N_1, u_{12}^1\}$ is labelled with 0, exactly one of them must be chosen to construct an independent subset of hyperedges covering all zeros. In contrast to that, there is no hyperedge E_2^2 because $\boldsymbol{w}_2$ is not compatible with N_2, and so $E_2^1 = [u_{21}^1, N_2]$ must be present in every covering of all zeros. Both $(\boldsymbol{w}_1, \boldsymbol{w}_1)$ and $(\boldsymbol{w}_2, \boldsymbol{w}_1)$ are compatible with (N_1, N_2), which gives $E_{12}^{11} = [v_{12}^1, v_{21}^1]$ and $E_{12}^{21} = [v_{12}^2, v_{21}^1]$. Thus, if N_1 is of type $\boldsymbol{w}_1$, then we include E_1^1 and E_{12}^{11} in the covering of all zeros, and so $p_{R(y_4,y_1)} \wedge p_{R(y_2,y_1)}$ should hold. If N_1 is of type $\boldsymbol{w}_2$, then instead of E_{12}^{11}, we take E_{12}^{21},

Table I. HGPs computing tree-witness hypergraph functions for OMQs.

OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$	$P_{\boldsymbol{Q}}$	of size	computes
$\mathcal{T}$ of depth 1	mHGP^2	$O(\lvert\boldsymbol{q}\rvert)$	$f^{\triangledown}_{\boldsymbol{Q}}$
tree-shaped $\boldsymbol{q}$ with ℓ leaves	$\mathsf{mTHGP}(\ell)$	$\lvert\boldsymbol{q}\rvert^{O(\ell)}$	$f^{\triangledown}_{\boldsymbol{Q}}$
$\boldsymbol{q}$ of treewidth t, $\Omega_{\boldsymbol{Q}}$ a fundamental set	mTHGP	$\lvert\boldsymbol{q}\rvert^{O(1)} \cdot \lvert\Omega_{\boldsymbol{Q}}\rvert^{O(t)}$	$f^{\blacktriangledown}_{\boldsymbol{Q}}$

Table II. Representation results for classes of hypergraphs.

hypergraph H	is isomorphic to	any mHGP based on H computes a subfunction of
any	a subgraph of $\mathcal{H}(\boldsymbol{Q}_H)$	$f^{\vartriangle}_{\boldsymbol{Q}_H}$
of degree 2	$\mathcal{H}(\boldsymbol{S}_H)$	$f^{\vartriangle}_{\boldsymbol{S}_H}$
tree hypergraph	a subgraph of $\mathcal{H}(\boldsymbol{T}_H)$	$f^{\vartriangle}_{\boldsymbol{T}_H}$

and so $p_{y_4=y_1} \land p_{y_2=y_1} \land p_{R^*(y_1)} \land p_{R^*(y_2)} \land p_{R^*(y_4)}$ should be true. Finally, since $\{v^1_{21}, u^1_{21}\}$ does not belong to any hyperedge, $p_{S_1(y_4,y_3)} \land p_{S_2(y_2,y_3)}$ must hold in either case.

5.5. Summary

In Tables I and II, we summarise the results of Section 5 that will be used in Section 7 to obtain lower and upper bounds for the size of OMQ rewritings. Table I shows how Theorems 5.3 and 5.7 (on the shape of tree-witness hypergraphs) combined with Proposition 4.7 (*ii*), as well as Theorem 5.13 provide us with hypergraph programs computing tree-witness hypergraph functions for OMQs. Table II contains the representation results of Theorems 5.2, 5.5 and 5.9 that show how abstract hypergraphs can be embedded into tree-witness hypergraphs of OMQs.

6. HYPERGRAPH PROGRAMS AND CIRCUIT COMPLEXITY

In the previous section, we saw how different classes of OMQs gave rise to different classes of monotone HGPs. Here we characterise the computational power of HGPs in these classes by relating them to standard models of computation for Boolean functions. Table III shows some of the obtained results. For example, its first row says that any Boolean function computable by a polynomial-size nondeterministic circuit can also be computed by a polynomial-size HGP of degree at most 3, and the other way round.

We remind the reader that the complexity classes in the table form the chain

$$\Pi_3 \subsetneqq \mathsf{AC}^0 \subsetneqq \mathsf{NC}^1 \subseteq \mathsf{NL/poly} \subseteq \mathsf{LOGCFL/poly} \subseteq \mathsf{P/poly} \subseteq \mathsf{NP/poly} \qquad (9)$$

and that whether any of the non-strict inclusions is actually strict remains a major open problem in complexity theory; see, e.g., [Arora and Barak 2009; Jukna 2012]. All these classes are non-uniform in the sense that they are defined in terms of polynomial-size non-uniform sequences of Boolean circuits of certain shape and depth. The suffix '/poly' comes from an alternative definition of $\mathsf{C/poly}$ in terms of Turing machines for the class C with an additional advice input of polynomial size.

When talking about complexity classes, instead of individual Boolean functions, we consider *sequences* of functions $f = \{f_n\}_{n<\omega}$ with $f_n \colon \{0,1\}^n \to \{0,1\}$. The same concerns circuits, HGPs and the other models of computation we deal with. For example, we say that a circuit $\boldsymbol{C} = \{\boldsymbol{C}_n\}_{n<\omega}$ *computes* a function $f = \{f_n\}_{n<\omega}$ if $\boldsymbol{C}_n$ computes f_n

Table III. Complexity classes, models of computation and corresponding classes of HGPs.

complexity class	model of computation	class of HGPs
NP/poly	nondeterministic Boolean circuits	$\mathsf{HGP} = \mathsf{HGP}^d$, $d \geq 3$
P/poly	Boolean circuits	—
LOGCFL/poly (SAC^1)	logarithmic-depth circuits with unbounded fan-in AND-gates and NOT-gates only on inputs	THGP
NL/poly	nondeterministic branching programs	$\mathsf{HGP}^2 = \mathsf{THGP}(\ell)$, $\ell \geq 2$
NC^1	Boolean formulas	THGP^d, $d \geq 3$
AC^0	constant-depth circuits with unbounded fan-in AND- and OR-gates, and NOT-gates only on inputs	—
Π_3	AC^0 circuits of depth 3 with output AND-gate	$\mathsf{THGP}^2 = \mathsf{THGP}^2(2)$

for every $n < \omega$. (It will always be clear from context whether f, $\boldsymbol{C}$, etc. denote an individual function, circuit, etc. or a sequence thereof.) A circuit $\boldsymbol{C}$ is said to be *polynomial* if there is a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that $|\boldsymbol{C}_n| \leq p(n)$, for every $n < \omega$. The *depth* of $\boldsymbol{C}_n$ is the length of the longest directed path from an input to the output of $\boldsymbol{C}_n$.

The complexity class P/poly can be defined as comprising those Boolean functions that are computed by polynomial circuits, and NC^1 consists of functions computed by polynomial formulas (that is, circuits every logic gate in which has at most one output). Alternatively, a Boolean function is in NC^1 iff it can be computed by a polynomial-size circuit of logarithmic depth, whose AND- and OR-gates have two inputs.

LOGCFL/poly (also known as SAC^1) is the class of Boolean functions computable by polynomial-size and logarithmic-depth circuits in which AND-gates have two inputs but OR-gates can have arbitrarily many inputs (*unbounded fan-in*) and NOT-gates can only be applied to inputs of the circuit [Vollmer 1999]. AC^0 is the class of functions computable by polynomial-size circuits of constant depth with AND- and OR-gates of unbounded fan-in and NOT-gates only at the inputs; Π_3 is the subclass of AC^0 that only allows circuits of depth 3 (not counting the NOT-gates) with an output AND-gate.

Finally, a Boolean function $f = \{f_n\}_{n<\omega}$ is in the class NP/poly if there is a polynomial p and a polynomial circuit $\boldsymbol{C} = \{\boldsymbol{C}_{n+p(n)}\}_{n<\omega}$ such that, for any n and $\boldsymbol{\alpha} \in \{0,1\}^n$,

$$f_n(\boldsymbol{\alpha}) = 1 \quad \text{iff} \quad \text{there is } \boldsymbol{\beta} \in \{0,1\}^{p(n)} \text{ such that } \boldsymbol{C}_{n+p(n)}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1 \tag{10}$$

(the $\boldsymbol{\beta}$-inputs are sometimes called *certificate inputs*).

By allowing only *monotone* circuits or formulas in the definitions of the complexity classes, we obtain their monotone variants: for example, the monotone variant of NP/poly is denoted by mNP/poly and defined by restricting the use of NOT-gates in the circuits to the certificate inputs only. We note in passing that the monotone variants of the classes in (9) also form a chain [Razborov 1985; Alon and Boppana 1987; Karchmer and Wigderson 1988]:

$$\mathsf{m\Pi_3} \subsetneqq \mathsf{mAC^0} \subsetneqq \mathsf{mNC^1} \subsetneqq \mathsf{mNL/poly} \subseteq \mathsf{mLOGCFL/poly} \subsetneqq \mathsf{mP/poly} \subsetneqq \mathsf{mNP/poly}. \tag{11}$$

Whether the inclusion mNL/poly $\subseteq$ mLOGCFL/poly is proper remains an open problem.

We use these facts in the next section to show lower bounds on the size of OMQ rewritings.

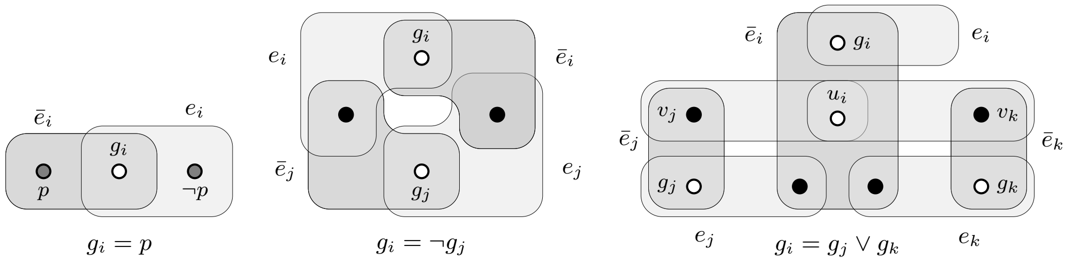


Fig. 15. HGP in the proof of Theorem 6.1: black vertices are labelled with 1 and white vertices with 0.

6.1. NP/poly and HGP^3

Our first result shows that NP/poly and mNP/poly coincide with the classes HGP^3 and mHGP^3 of Boolean functions computable by polynomial-size (sequences of) HGPs and monotone HGPs of degree at most 3, respectively.

THEOREM 6.1. NP/poly = HGP = HGP^3 *and* mNP/poly = mHGP = mHGP^3.

PROOF. Suppose P is a (monotone) HGP. We construct a non-deterministic circuit $\boldsymbol{C}$ of size polynomial in $|P|$, whose input variables are the same as the variables in P, certificate inputs correspond to the hyperedges of P, and such that $\boldsymbol{C}(\boldsymbol{\alpha},\boldsymbol{\beta}) = 1$ iff $\{e_i \mid \boldsymbol{\beta}(e_i) = 1\}$ is an independent set of hyperedges covering all zeros under $\boldsymbol{\alpha}$. It will then follow that

$$P(\boldsymbol{\alpha}) = 1 \quad \text{iff} \quad \text{there is } \boldsymbol{\beta} \text{ such that } \boldsymbol{C}(\boldsymbol{\alpha},\boldsymbol{\beta}) = 1. \tag{12}$$

First, for each pair of intersecting hyperedges e_i, e_j in P, we take the disjunction $\neg e_i \vee \neg e_j$, and, for each vertex in P labelled with a literal $\boldsymbol{l}$ (that is, p or $\neg p$) and the hyperedges $e_{i_1}, \dots, e_{i_k}$ incident to it, we take the disjunction $\boldsymbol{l} \vee e_{i_1} \vee \dots \vee e_{i_k}$. The circuit $\boldsymbol{C}$ is then a conjunction of all such disjunctions. Note that if P is monotone, then $\neg$ is only applied to the certificate inputs, $\boldsymbol{e}$, in $\boldsymbol{C}$.

Conversely, let $\boldsymbol{C}$ be a circuit with certificate inputs. We construct an HGP P of degree at most 3 satisfying (12) as follows. For each gate g_i in $\boldsymbol{C}$, the HGP contains a vertex g_i labelled with 0 and a pair of hyperedges $\bar{e}_i$ and e_i, both containing g_i. No other hyperedge contains g_i, and so either $\bar{e}_i$ or e_i should be present in any cover of zeros. To ensure this property, for each gate g_i, we add the following vertices and hyperedges to P (see Fig. 15):

– if g_i is an input p, then we add a vertex labelled with $\neg p$ to e_i and a vertex labelled with p to $\bar{e}_i$;
– if g_i is a certificate input, then no additional vertices and hyperedges are added;
– if $g_i = \neg g_j$, then we add a vertex labelled with 1 to hyperedges e_i and $\bar{e}_j$, and a vertex labelled with 1 to hyperedges $\bar{e}_i$ and e_j;
– if $g_i = g_j \vee g_k$, then we add a vertex labelled with 1 to hyperedges e_j and $\bar{e}_i$, add a vertex labelled with 1 to e_k and $\bar{e}_i$; then, we add vertices v_j and v_k labelled with 1 to $\bar{e}_j$ and $\bar{e}_k$, respectively, and a vertex u_i labelled with 0 to $\bar{e}_i$; finally, we add hyperedges $\{v_j, u_i\}$ and $\{v_k, u_i\}$ to P;
– if $g_i = g_j \wedge g_k$, then we add the pattern dual to the case of $g_i = g_j \vee g_k$: we add a vertex labelled with 1 to $\bar{e}_j$ and e_i, a vertex labelled with 1 to $\bar{e}_k$ and e_i; then, we add vertices v_j and v_k labelled with 1 to e_j and e_k, respectively, and a vertex u_i labelled with 0 to e_i; finally, we add hyperedges $\{v_j, u_i\}$ and $\{v_k, u_i\}$ to P.

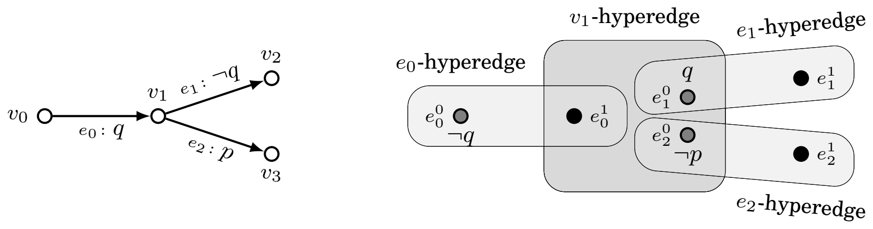


Fig. 16. HGP in the proof of Theorem 6.2: black vertices are labelled with 1.

Finally, we add one more vertex labelled with 0 to e_m for the output gate g_m of $\boldsymbol{C}$, which ensures that e_m must be included the cover. It is easily verified that the constructed HGP is of degree at most 3. One can establish (12) by induction on the structure of $\boldsymbol{C}$. We illustrate the proof of the inductive step for the case of $g_i = g_j \vee g_k$: we show that e_i is in the cover iff it contains either e_j or e_k. Suppose the cover contains e_j. Then it cannot contain $\bar{e}_i$, and so it contains e_i. The vertex u_i in this case can be covered by $\{v_j, u_i\}$ since $\bar{e}_j$ is not in the cover. Conversely, if neither e_j nor e_k is in the cover, then it must contain both $\bar{e}_j$ and $\bar{e}_k$, and so neither $\{v_j, u_i\}$ nor $\{v_k, u_i\}$ can belong to the cover, and thus we will have to include $\bar{e}_i$ to the cover.

If $\boldsymbol{C}$ is monotone, then we remove from P all vertices labelled with $\neg p$, for an input p, and denote the resulting program by P'. We claim that, for any $\boldsymbol{\alpha}$, we have $P'(\boldsymbol{\alpha}) = 1$ iff there is $\boldsymbol{\beta}$ such that $\boldsymbol{C}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1$. The implication ($\Leftarrow$) is trivial: if $\boldsymbol{C}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1$ then, by the argument above, $P(\boldsymbol{\alpha}) = 1$ and, clearly, $P'(\boldsymbol{\alpha}) = 1$. Conversely, suppose $P'(\boldsymbol{\alpha}) = 1$. Each of the vertices g_i in P' corresponding to the inputs is covered by one of the hyperdges e_i or $\bar{e}_i$. Let $\boldsymbol{\alpha}'$ be the vector corresponding to these hyperedges; clearly, $\boldsymbol{\alpha}' \leq \boldsymbol{\alpha}$. This cover of vertices of P' gives us $P(\boldsymbol{\alpha}') = 1$. Thus, by the argument above, there is $\boldsymbol{\beta}$ such that $\boldsymbol{C}(\boldsymbol{\alpha}', \boldsymbol{\beta}) = 1$. Since $\boldsymbol{C}$ is monotone, $\boldsymbol{C}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1$. □

6.2. NL/poly and HGP^2

A Boolean function belongs to the class NL/poly iff it can be computed by a polynomial-size *nondeterministic branching program* (NBP). We remind the reader (consult [Jukna 2012] for more details) that an NBP B is a directed graph $G = (V, E)$, whose arcs are labelled with the Boolean constants 0 and 1, propositional variables $p_1, \dots, p_n$ or their negations, and which distinguishes two vertices $s, t \in V$. Given an assignment $\boldsymbol{\alpha}$ to variables $p_1, \dots, p_n$, we write $s \to_{\boldsymbol{\alpha}} t$ if there is a path in G from s to t all of whose labels evaluate to 1 under $\boldsymbol{\alpha}$. We say that an NBP B *computes* a Boolean function f if $f(\boldsymbol{\alpha}) = 1$ iff $s \to_{\boldsymbol{\alpha}} t$, for any $\boldsymbol{\alpha} \in \{0,1\}^n$. The *size* $|B|$ of B is the size of the underlying graph, $|V| + |E|$. An NBP is *monotone* if there are no negated variables among its labels. The class of Boolean functions computable by polynomial-size monotone NBPs is denoted by mNL/poly; the class of functions f whose *duals* $f^*(p_1, \dots, p_n) = \neg f(\neg p_1, \dots, \neg p_n)$ are in mNL/poly is denoted by co-mNL/poly.

THEOREM 6.2. NL/poly = HGP^2 *and* co-mNL/poly = mHGP^2.

PROOF. As follows from [Szelepcsényi 1988; Immerman 1988], if a function f is computable by a polynomial-size NBP, then $\neg f$ is also computable by a polynomial-size NBP. So suppose $\neg f$ is computed by an NBP B. We construct an HGP P computing f of degree at most 2 and polynomial size in $|B|$ as follows (see Fig. 16). For each arc e in B, the HGP P has two vertices e^0 and e^1, which represent the beginning and the end of e, respectively. The vertex e^0 is labelled with the *negated* label of e in B and e^1 with 1. For each arc e in B, the HGP P has an e-hyperedge $\{e^0, e^1\}$. For each vertex v in B but s and t, the HGP P has a v-hyperedge comprising all vertices e^1 for the arcs

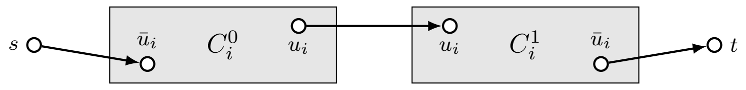


Fig. 17. The NBP in the proof of Theorem 6.2.

e leading to v, and all vertices e^0 for the arcs e leaving v. We also add to the HGP P a vertex w labelled with 0 and a hyperedge, $\bar{e}_w$, that consists of w and all vertices e^1 for the arcs e in B leading to t. We claim that the constructed HGP P computes f. Indeed, if $s \not\to_{\boldsymbol{\alpha}} t$ then the following subset of hyperedges is independent and covers all zeros: all e-hyperedges, for the arcs e reachable from s and labelled with 1 under $\boldsymbol{\alpha}$, and all v-hyperedges with $s \not\to_{\boldsymbol{\alpha}} v$ (including $\bar{e}_w$). Conversely, if $s \to_{\boldsymbol{\alpha}} t$ then one can show by induction that, for each arc e of the path, the e-hyperedge must be in the cover of all zeros. Thus, no independent set can cover w, which is labelled with 0.

Conversely, suppose f is computed by an HGP P of degree 2 with hyperedges $e_1, \dots, e_k$. We first provide a graph-theoretic characterisation of independent sets covering all zeros based on the implication graph [Aspvall et al. 1979]. With every hyperedge e_i we associate a propositional variable u_i and with every assignment $\boldsymbol{\alpha}$ we associate the following set $\Phi_{\boldsymbol{\alpha}}$ of propositional binary clauses:

$$\begin{array}{ll} \neg u_i \lor \neg u_j, & \text{if } e_i \cap e_j \neq \emptyset, \\ u_i \lor u_j, & \text{if there is } v \in e_i \cap e_j \text{ with } \boldsymbol{\alpha}(v) = 0. \end{array}$$

Informally, the former means that intersecting hyperedges cannot be chosen at the same time and the latter that all zeros must be covered; note that all vertices have at most two incident edges. By definition, X is an independent set covering all zeros iff $X = \{e_i \mid \boldsymbol{\gamma}(u_i) = 1\}$, for some assignment $\boldsymbol{\gamma}$ satisfying $\Phi_{\boldsymbol{\alpha}}$. Let $C_{\boldsymbol{\alpha}} = (V, E_{\boldsymbol{\alpha}})$ be the implication graph of $\Phi_{\boldsymbol{\alpha}}$, that is, a directed graph with

$$\begin{aligned} V &= \big\{u_i, \bar{u}_i \mid 1 \leq i \leq k\big\}, \\ E_{\boldsymbol{\alpha}} &= \big\{(u_i, \bar{u}_j) \mid e_i \cap e_j \neq \emptyset\big\} \cup \big\{(\bar{u}_i, u_j) \mid \text{there is } v \in e_i \cap e_j \text{ with } \boldsymbol{\alpha}(v) = 0\big\}. \end{aligned}$$

(V is the set of all 'literals' for the variables of $\Phi_{\boldsymbol{\alpha}}$ and $E_{\boldsymbol{\alpha}}$ is the arcs for the implicational form of the clauses of $\Phi_{\boldsymbol{\alpha}}$.) Note that $\neg u_i \lor \neg u_j$ gives rise to two implications, $u_i \to \neg u_j$ and $u_j \to \neg u_i$, and so to two arcs in the graph; similarly, for $u_i \lor u_j$. By [Aspvall et al. 1979, Theorem 1], $\Phi_{\boldsymbol{\alpha}}$ is satisfiable iff there is no u_i with a (directed) cycle going through u_i and $\bar{u}_i$. It will be convenient for us to regard the $C_{\boldsymbol{\alpha}}$, for assignments $\boldsymbol{\alpha}$, as a single labelled directed graph C with arcs of the form $(u_i, \bar{u}_j)$ labelled with 1 and arcs of the form $(\bar{u}_i, u_j)$ labelled with the negation of the literal labelling the uniquely defined $v \in e_i \cap e_j$ (recall that the hypergraph of P is of degree 2). It should be clear that $C_{\boldsymbol{\alpha}}$ has a cycle going through u_i and $\bar{u}_i$ iff we have both $\bar{u}_i \to_{\boldsymbol{\alpha}} u_i$ and $u_i \to_{\boldsymbol{\alpha}} \bar{u}_i$ in C. The required NBP B contains distinguished vertices s and t, and, for each hyperedge e_i in P, two copies, C_i^0 and C_i^1, of C with additional arcs from s to the $\bar{u}_i$ vertex of C_i^0, from the u_i vertex of C_i^0 to the u_i vertex of C_i^1, and from the $\bar{u}_i$ vertex of C_i^1 to t; see Fig. 17. By construction, $s \to_{\boldsymbol{\alpha}} t$ iff there is a hyperedge e_i in P such that $C_{\boldsymbol{\alpha}}$ contains a cycle going through u_i and $\bar{u}_i$. We have thus constructed a polynomial-size NBP B computing $\neg f$, and thus f must also be computable by a polynomial-size NBP.

As to $\mathsf{co\text{-}mNL/poly} = \mathsf{mHGP}^2$, observe that the first construction, if applied to a monotone NBP for f^*, produces a polynomial-size HGP of degree 2 computing $\neg f^*$, all of whose labels are negative. By removing negations from labels, we obtain a monotone HGP computing f. The second construction allows us to transform a monotone HGP of degree 2 for f into an NBP with only negative literals that computes $\neg f$. By changing the polarity of the literals in the labels, we obtain a monotone NBP computing f^*. □

6.3. NL/poly and THGP(ℓ)

For any natural $\ell \geq 2$, we denote by $\mathsf{THGP}(\ell)$ and $\mathsf{mTHGP}(\ell)$ the classes of Boolean functions computable by (sequences of) polynomial-size THGPs and, respectively, monotone THGPs whose underlying trees have at most ℓ leaves.

THEOREM 6.3. $\mathsf{NL/poly} = \mathsf{THGP}(\ell)$ *and* $\mathsf{mNL/poly} = \mathsf{mTHGP}(\ell)$*, for any* $\ell \geq 2$.

PROOF. Suppose a polynomial-size THGP P computes a Boolean function f. For simplicity, we consider only $\ell = 2$ here and prove the general case in Appendix E. Thus, we can assume that the vertices $v_1, \dots, v_n$ of P are consecutive edges of the path graph underlying P, and therefore, every hyperedge in P is of the form $[v_i, v_{i+m}] = \{v_i, \dots, v_{i+m}\}$, for some $m \geq 0$. We add to P two extra vertices, v_0 and v_{n+1} (thereby extending the underlying 2-leaf tree to $v_0, v_1, \dots, v_n, v_{n+1}$) and label them with 0; we also add two hyperedges $s = \{v_0\}$ and $t = \{v_{n+1}\}$ to P. Clearly, the resulting THGP P' computes the same f. To construct a polynomial-size NBP B computing f, we take a directed graph whose vertices are hyperedges of P' and which contains an arc from $e_i = [v_{i_1}, v_{i_2}]$ to $e_j = [v_{j_1}, v_{j_2}]$ iff $i_2 < j_1$; we label this arc with $\bigwedge_{i_2<k<j_1} \boldsymbol{l}_k$, where $\boldsymbol{l}_k$ is the label of v_k in HGP P. It is not hard to see that a path from s to t evaluated to 1 under given assignment $\boldsymbol{\alpha}$ corresponds to a cover of zeros in P' under $\boldsymbol{\alpha}$. Finally, to get rid of conjunctive labels on edges, we replace every arc with a label $\boldsymbol{l}_{i_1} \wedge \dots \wedge \boldsymbol{l}_{i_k}$ by a sequence of k arcs consequently labelled with $\boldsymbol{l}_{i_1}, \dots, \boldsymbol{l}_{i_k}$.

Conversely, suppose a Boolean function f is computed by an NBP B based on a directed graph with vertices $V = \{v_1, \dots, v_n\}$, edges $E = \{e_1, \dots, e_m\}$, $s = v_1$ and $t = v_n$. Without loss of generality, we assume that e_m is a loop from t to t labelled with 1. Thus, if there is a path from s to t whose labels evaluate to 1, then there is such a path of length $n-1$. We now construct a polynomial-size THGP computing f whose underlying tree T has two leaves. The vertices of the tree T are arranged into n vertex blocks and $n-1$ edge blocks, which alternate. The kth vertex (edge) block contains two copies $v_i^k, \bar{v}_i^k$ (respectively, $e_i^k, \bar{e}_i^k$) of every $v_i \in V$ (respectively, $e_i \in E$):

$$\underbrace{v_1^1, \bar{v}_1^1, v_2^1, \bar{v}_2^1, \dots, v_n^1, \bar{v}_n^1,}_{\text{1st vertex block}} \ \underbrace{e_1^1, \bar{e}_1^1, e_2^1, \bar{e}_2^1, \dots, e_m^1, \bar{e}_m^1,}_{\text{1st edge block}} \ \underbrace{v_1^2, \bar{v}_1^2, v_2^2, \bar{v}_2^2, \dots, v_n^2, \bar{v}_n^2,}_{\text{2nd vertex block}} \ \dots$$
$$\underbrace{e_1^{n-1}, \bar{e}_1^{n-1}, e_2^{n-1}, \bar{e}_2^{n-1}, \dots, e_m^{n-1}, \bar{e}_m^{n-1},}_{(n-1)\text{th edge block}} \ \underbrace{v_1^n, \bar{v}_1^n, v_2^n, \bar{v}_2^n, \dots, v_n^n, \bar{v}_n^n}_{n\text{th vertex block}}.$$

We remove the first, v_1^1, and last vertex, $\bar{v}_n^n$ (shown in grey in the formula above), and connect the adjacent vertices by edges to construct the undirected tree T. Consider now a hypergraph H whose vertices are the edges of T and hyperedges are of the form $h_i^k = [\bar{v}_j^k, e_i^k]$ and $g_i^k = [\bar{e}_i^k, v_{j'}^{k+1}]$, for $e_i = (v_j, v_{j'}) \in E$ and $1 \leq k < n$. The vertices of H of the form $\{e_i^k, \bar{e}_i^k\}$, which separate hyperedges h_i^k and g_i^k, are labelled with the label of e_i in the given NBP B, and all other vertices of H with 0. We show now that the constructed THGP P computes f. Indeed, if $f(\boldsymbol{\alpha}) = 1$, then there is a path $e_{i_1}, \dots, e_{i_{n-1}}$ from v_1 to v_n whose labels evaluate to 1 under $\boldsymbol{\alpha}$. It follows that $\{h_{i_k}^k, g_{i_k}^k \mid 1 \leq k < n\}$ is an independent set in H covering all zeros. Conversely, if E' is an independent set in H and covers all zeros under $\boldsymbol{\alpha}$, then it must contain exactly one pair of hyperedges $h_{i_k}^k$ and $g_{i_k}^k$ for every k with $1 \leq k < n$, and the corresponding sequence of edges $e_{i_1}, \dots, e_{i_{n-1}}$ defines a path from v_1 to v_n. Moreover, since E' does not cover vertices $\{e_{i_k}^k, \bar{e}_{i_k}^k\}$, for $1 \leq k < n$, their labels (that is, the labels of the e_{i_k} in B) evaluate to 1 under $\boldsymbol{\alpha}$. □

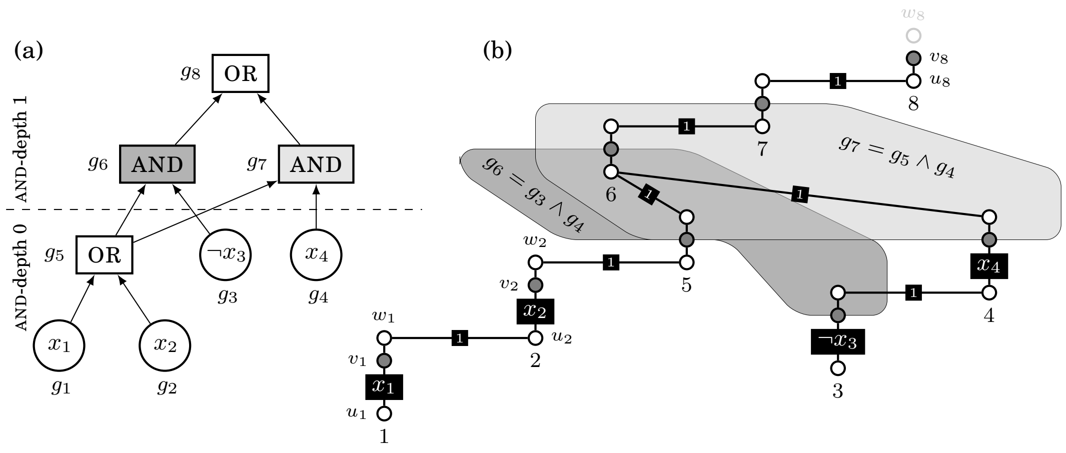


Fig. 18. (a) A circuit $\boldsymbol{C}$. (b) The labelled tree T for $\boldsymbol{C}$: the vertices in the ith triple are u_i, v_i, w_i and the omitted edge labels are 0s. The vertices of THGP are the edges of T (with the same labels) and the hyperedges are sets of edges of T (two of them are shown).

To prove Theorem 7.6 below, we shall require a somewhat different variant of Theorem 6.3. The proof of the following result is given in Appendix E:

THEOREM 6.4. *Fix $\ell \geq 2$. For any tree hypergraph H based on a tree with at most ℓ leaves, the function f_H can be computed by an NBP of size polynomial in $|H|$.*

Note that Theorem 6.4 does not immediately follow from Theorem 6.3 and Proposition 5.10 (*i*) because the transformation of H into a monotone HGP computing f_H given in the proof of Proposition 5.10 (*i*) does not preserve the number of leaves.

6.4. LOGCFL/poly and THGP

THGP and mTHGP are the classes of functions computable by polynomial-size THGPs and, respectively, monotone THGPs.

THEOREM 6.5. LOGCFL/poly = THGP *and* mLOGCFL/poly = mTHGP.

PROOF. To show LOGCFL/poly $\subseteq$ THGP, consider a SAC^1-circuit $\boldsymbol{C}$ of depth $d \leq \log |\boldsymbol{C}|$. It will be convenient to think of $\boldsymbol{C}$ as containing no NOT-gates but having *literals* as inputs. By the AND-*depth* of a gate g in $\boldsymbol{C}$ we mean the maximal number of AND-gates in a path from an input of $\boldsymbol{C}$ to g (it does not exceed d). Let S_n be the set of AND-gates in $\boldsymbol{C}$ of AND-depth n. We denote by left(g) and right(g) the sub-circuits of $\boldsymbol{C}$ computing the left and right inputs of an AND-gate g, respectively. Without loss of generality (see Lemma F.1 in Appendix F) we can assume that, for any $n \leq d$,

$$\bigcup_{g \in S_n} \mathsf{left}(g) \quad \cap \quad \bigcup_{g \in S_n} \mathsf{right}(g) \quad = \quad \emptyset.$$

Our aim is to transform $\boldsymbol{C}$ into a polynomial-size THGP P. We construct its underlying tree T by associating with each gate g_i three vertices u_i, v_i, w_i and arranging them into a tree as shown in Fig. 18. More precisely, we first arrange the vertices associated with the gates of maximal AND-depth, n, into a path following the order of the gates in $\boldsymbol{C}$ and the alphabetic order for u_i, v_i, w_i. Then we fork the path into two branches one of which is associated with the sub-circuit $\bigcup_{g \in S_n} \mathsf{left}(g)$ and the other with $\bigcup_{g \in S_n} \mathsf{right}(g)$, and so forth. We obtain the tree T by removing the vertex w_m from the result, where $m = |\boldsymbol{C}|$ and g_m is the output gate of $\boldsymbol{C}$; it has v_m as its root and contains $3|\boldsymbol{C}| - 1$

vertices. The THGP P is based on the hypergraph whose vertices are the edges of T and whose hyperedges comprise the following (see Fig. 18):

– $[w_i, u_i]$, for each $i < m$ (pairs of edges in each triple of vertices in Fig. 18);
– $[v_j, v_k, v_i]$, for each $g_i = g_j \wedge g_k$ (shown in Fig. 18 by shading);
– $[v_{j_1}, v_i], \dots, [v_{j_k}, v_i]$, for each $g_i = g_{j_1} \vee \dots \vee g_{j_k}$,

where $[L]$ is the minimal convex subtree of T containing the vertices in L. Finally, if an input gate g_i is a literal $\boldsymbol{l}$, we label the edge $\{u_i, v_i\}$ with $\boldsymbol{l}$; we label all other $\{u_j, v_j\}$- and $\{w_j, v_j\}$-edges with 0, and the remaining ones with 1. Clearly, the size of P is polynomial in $|\boldsymbol{C}|$. By Lemma F.2, for any input $\boldsymbol{\alpha}$, the output of g_i is 1 iff the subtree with root v_i can be covered, i.e., there is an independent set of hyperedges wholly inside and covering all zeros. Thus, P computes the same function as $\boldsymbol{C}$.

To show $\mathsf{THGP} \subseteq \mathsf{LOGCFL/poly}$, suppose a THGP P is based on a hypergraph H with an underlying tree T. By a *subtree* of T we understand a (possibly empty) connected subset of edges in T. Given an input $\boldsymbol{\alpha}$ for P and a nonempty subtree D of T, we set cover_D true iff there exists an independent subset of hyperedges in H that lie in D and cover all zeros in D. We also set $\mathsf{cover}_\emptyset$ true. Note that, for any edge e of T, $\mathsf{cover}_{\{e\}}$ is true if $\{e\}$ is a hyperedge of H; otherwise $\mathsf{cover}_{\{e\}}$ is the value of e's label in P under $\boldsymbol{\alpha}$.

Our aim is to construct recursively a polynomial-size SAC^1-circuit $\boldsymbol{C}$ computing the function cover_T. Observe that, if D is a subtree of T and a vertex v splits D into subtrees $D_1, \dots, D_k$, then

$$\mathsf{cover}_D \quad = \quad \bigwedge_{1 \le j \le k} \mathsf{cover}_{D_j} \quad \vee \quad \bigvee_{v \in h \subseteq D} \quad \bigwedge_{1 \le j \le k_h} \mathsf{cover}_{D^h_j}, \tag{13}$$

where h ranges over the hyperedges in H, and $D^h_1, \dots, D^h_{k_h}$ are the maximal convex subtrees of T that lie in $D \setminus h$. We call a vertex v of D *boundary* if T has an edge $\{v, u\}$ with u not in D, and define the *degree* $\mathsf{deg}(D)$ of D to be the number of its boundary vertices. Note that T itself is the only subtree of T of degree 0. The following lemma shows that to compute cover_T we only need subtrees of degree 1 and 2 and the depth of recursion $O(\log |P|)$.

LEMMA 6.6. *Let D be a subtree of T with m vertices and $\mathsf{deg}(D) \le 2$. If $\mathsf{deg}(D) \le 1$, then there is a vertex v splitting D into subtrees of size at most $m/2 + 1$ and degree at most 2. If $\mathsf{deg}(D) = 2$, then there is v splitting D 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. Let $\mathsf{deg}(D) \le 1$. Suppose some vertex v_1 splits D into subtrees one of which, say D_1, is larger than $m/2 + 1$. Let v_2 be the (unique) vertex in D_1 adjacent to v_1. The splitting of D by v_2 consists of the subtree $D_2 = (D \setminus D_1) \cup \{v_1, v_2\}$ of size at most $m/2$ and some other subtrees lying inside D_1; all of them are of degree at most 2. We repeat this process until the size of the largest subtree becomes at most $m/2 + 1$.

Let $\mathsf{deg}(D) = 2$, with b_1 and b_2 being the boundary vertices. We proceed as above starting from $v_1 = b_1$, but stop when either the largest subtree has $\le m/2 + 1$ vertices or v_{i+1} leaves the path between b_1 and b_2, in which case v_i splits D into subtrees of degree at most 2 and one subtree of degree 1 with more than $m/2 + 1$ vertices. □

By applying (13) to T recursively and choosing the splitting vertices v as prescribed by Lemma 6.6, we obtain a circuit $\boldsymbol{C}$ whose inputs are the labels of some vertices of H. Since any tree has polynomially many subtrees of degree 1 or 2, the size of $\boldsymbol{C}$ is polynomial in $|P|$. We now show how to make the depth of $\boldsymbol{C}$ logarithmic in $|P|$.

Suppose D is a subtree with m edges constructed on the recursion step i. To compute cover_D using (13), we need one OR-gate of unbounded fan-in and a number of AND-gates of fan-in 2. We show by induction that we can make the AND-depth of these AND-gates at most $\log m + i$. Suppose D_j in (13) has m_j edges, and so $m = m_1 + \cdots + m_k$. By the induction hypothesis, we can compute each cover_{D_j} within the AND-depth at most $\log m_j + i - 1$. Assign the probability m_j/m to D_j. As shown by Huffman [1952], there is a prefix binary code such that each D_j is encoded by a word of length $\lceil \log(m/m_j) \rceil$. This encoding can be represented as a binary tree whose leaves are labelled with the D_j so that the length of the branch ending at D_j is $\lceil \log(m/m_j) \rceil$. By replacing each non-leaf vertex of the tree with an AND-gate, we obtain a circuit for the first conjunction in (13) whose depth does not exceed

$$\max_j \{\log m_j + (i-1) + \log(m/m_j) + 1\} \quad = \quad \log m + i.$$

The second conjunction is considered analogously. □

6.5. NC^1, Π$_3$ and THGP^d

The proof of the following theorem, given in Appendix F, is a simplified version of the proof of Theorem 6.5:

THEOREM 6.7. $\mathsf{NC}^1 = \mathsf{THGP}^d$ *and* $\mathsf{mNC}^1 = \mathsf{mTHGP}^d$*, for any* $d \geq 3$.

THGPs of degree 2 turn out to be less expressive:

THEOREM 6.8. $\Pi_3 = \mathsf{THGP}^2 = \mathsf{THGP}^2(2)$ *and* $\mathsf{m}\Pi_3 = \mathsf{mTHGP}^2 = \mathsf{mTHGP}^2(2)$.

PROOF. To show $\mathsf{THGP}^2 \subseteq \Pi_3$, take a THGP P of degree 2. Without loss of generality we can assume that it contains no hyperedges e, e' with $e \subseteq e'$, for otherwise the vertices in e would not be covered by any other hyperedges, and so could be removed from P together with e.

Consider the graph D whose vertices are the hyperedges of P, with two vertices being connected if the corresponding hyperedges intersect. Clearly, D is a forest. We label an edge $\{e_1, e_2\}$ with the conjunction of the labels of the vertices in $e_1 \cap e_2$, and label a vertex e with the conjunction of the labels of the vertices in P contained exclusively in e. It is easy to see that, for any given input, an independent cover of zeros in P corresponds to an independent set in D covering all zeros in the vertices and such that each edge labelled with 0 has precisely one endpoint in that independent set.

We claim that there is no such an independent set I in D iff there is a path $e_0, e_1, \dots, e_k$ in D with odd k (in particular, $k = 1$) such that e_0 and e_k are labelled with 0 and 'even' edges $\{e_{i-1}, e_i\}$ with even i are labelled with 0. To see ($\Leftarrow$), observe that we have to include e_0 and e_k in I. Then, the edge $\{e_1, e_2\}$ labelled with 0 makes us to include e_2 to I (e_1 is adjacent to e and cannot be included in I). Next, the edge $\{e_3, e_4\}$ makes us to include e_4 in I and so on. In the end we will have to include e_{k-1} to I and, since e_k is also in I, this gives a contradiction with independence of I.

To show ($\Rightarrow$), suppose there is no such a pair of vertices. Then we can construct a desired independent set I. Add to I all vertices labelled with 0. If there is a triple of consecutive vertices e, e_1, e_2 in D such that e is already in I and an edge $\{e_1, e_2\}$ is labelled with 0, then we add e_2 to I. Note that, if we have add some vertex e' to e in this process, then there is a path $e = e_0, e_1, \dots, e_k = e'$ with even k such that vertex e is labelled with 0 and every edge $\{e_{i-1}, e_i\}$ for even i in this path is labelled with 0.

In this process we never add two connected vertices e and e' of D to I, for otherwise the union of the paths described above for these two vertices would result in a path of odd length with endpoints labelled with 0 and with every second edge labelled with 0. This directly contradicts our assumption.

If there are still edges labelled with 0 in D with no endpoints in I, then add any endpoint of such an edge to I and repeat the process above. This also will not lead to a

Table IV. The size of OMQ rewritings.

OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$	PE	NDL	FO
$\mathcal{T}$ of depth 2	exp (Th. 7.1)	exp (Th. 7.1)	> poly if NP $\not\subseteq$ P/poly (Th. 7.1) poly iff NP/poly $\subseteq$ NC^1 (Th. 7.2)
$\mathcal{T}$ of depth 1	> poly (Th. 7.4)	poly (Th. 7.3)	poly iff NL/poly $\subseteq$ NC^1 (Th. 7.5)
& $\boldsymbol{q}$ of treewidth t	poly (Th. 7.12)	poly (Th. 7.12)	poly (Th. 7.12)
& $\boldsymbol{q}$ tree	poly-Π_4 (Th. 7.13)	poly (Th. 7.13)	poly-Π_4 (Th. 7.13)
$\boldsymbol{q}$ tree with ℓ leaves	> poly (Th. 7.7) ($\mathcal{T}$ is of depth 2)	poly (Th. 7.6)	poly iff NL/poly $\subseteq$ NC^1 (Th. 7.8)
$\boldsymbol{Q}$ with PFSP $\boldsymbol{q}$ of treewidth t		poly (Th. 7.9)	poly iff LOGCFL/poly $\subseteq$ NC^1 (Th. 7.11)

pair of connected vertices in I. Indeed, if as a result we add to I a vertex e_1 connected to a vertex e which was added to I previously, then there is an edge $\{e_2, e_1\}$ labelled with 0 (that was the reason for adding e_1 to I), and so we should have added e_2 to I before. By repeating this process, we obtain an independent set I covering all vertices and edges labelled with 0.

The established claim means that an independent set I in D exists iff, for any simple path $e_0, e_1, \dots, e_k$ with an odd k, the label of e_0 or e_k evaluates to 1, or the label of at least one $\{e_{i-1}, e_i\}$, for even i, evaluates to 1. This property is computed by a Π_3-circuit where, for each simple path $e_0, e_1, \dots, e_k$ with an odd k, we take $(k+3)/2$-many AND-gates whose inputs are the literals in the labels of e_0, e_k and the $\{e_{i-1}, e_i\}$ for even i; then we send the outputs of those AND-gates to an OR-gate; and, finally, we collect the outputs of all the OR-gates as inputs to an AND-gate.

To show $\Pi_3 \subseteq \mathsf{THGP}^2(2)$, suppose we are given a Π_3-circuit $\boldsymbol{C}$. We can assume $\boldsymbol{C}$ to be a conjunction of DNFs. So, we first construct a generalised HGP P from $\mathsf{THGP}^2(2)$ computing the same function as $\boldsymbol{C}$. Denote the OR-gates of $\boldsymbol{C}$ by $g^1, \dots, g^k$ and the inputs of g^i by $h^i_1, \dots, h^i_{l_i}$, where each h^i_j is an AND-gate. Now, we define a tree hypergraph whose underlying path graph has the following edges (in the given order)

$$v^1_0, \dots, v^1_{2l_1-2}, \quad v^2_0, \dots, v^2_{2l_2-2}, \quad \dots, \quad v^k_0, \dots, v^k_{2l_k-2}$$

and whose hyperedges are of the form $\{v^i_j, v^i_{j+1}\}$. We label v^i_{2m} with a conjunction of the inputs of h^i_{m+1} and the remaining vertices with 0. By the previous analysis for a given i and an input for $\boldsymbol{C}$, we can cover all zeros among $v^i_0, \dots, v^i_{2l_i-2}$ with an independent set of hyperedges iff at least one of the gates $h^i_1, \dots, h^i_{l_i}$ outputs 1. For different i, the corresponding $v^i_0, \dots, v^i_{2l_i-2}$ are covered independently. Thus, P computes the same function as $\boldsymbol{C}$. We convert P to a THGP from $\mathsf{THGP}^2(2)$ using Proposition 5.10 (*ii*). □

7. THE SIZE OF OMQ REWRITINGS

In this section, by an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ we mean a sequence $\{\boldsymbol{Q}_n = (\mathcal{T}_n, \boldsymbol{q}_n)\}_{n<\omega}$ of OMQs whose size is polynomial in n; by a rewriting $\boldsymbol{q}'$ of $\boldsymbol{Q}$ we mean a sequence $\{\boldsymbol{q}'_n\}_{n<\omega}$, where each $\boldsymbol{q}'_n$ is a rewriting of $\boldsymbol{Q}_n$, for $n < \omega$.

By putting together the results of the previous three sections and some known facts from circuit complexity, we obtain the upper and lower bounds on the size of PE-, NDL- and FO-rewritings for various OMQ classes that are collected in Table IV, where exp means an exponential lower bound, > poly a superpolynomial lower bound, poly a polynomial upper bound, poly-Π_4 a polynomial-size Π_4-rewriting (that is, a PE-rewriting with the matrix of the form $\wedge\vee\wedge\vee$), and ℓ and t are any fixed constants. It is to be noted that, in case of polynomial upper bounds, we actually provide polynomial algorithms for constructing rewritings.

7.1. Rewritings for OMQs with ontologies of depth 2

By Theorem 6.1, OMQs with ontologies of depth 2 can compute any NP-complete monotone Boolean function, in particular, the function CLIQUE with $n(n-1)/2$ variables $e_{jj'}$, $1 \le j < j' \le n$, that returns 1 iff the graph with vertices $\{1,\dots,n\}$ and edges $\{\{j,j'\} \mid e_{jj'} = 1\}$ contains a k-clique, for some fixed k. A series of papers, started by Razborov [1985], gave an exponential lower bound for the size of monotone circuits computing CLIQUE, namely, $2^{\Omega(\sqrt{k})}$ for $k \le \frac{1}{4}(n/\log n)^{2/3}$ [Alon and Boppana 1987]. For monotone formulas, an even better lower bound is known: $2^{\Omega(k)}$ for $k = 2n/3$ [Raz and Wigderson 1992]. Thus, we obtain:

THEOREM 7.1. *There is an OMQ with ontologies of depth* 2*, any PE- and NDL-rewritings of which are of exponential size, while any FO-rewriting is of superpolynomial size unless* $\mathsf{NP} \subseteq \mathsf{P/poly}$.

PROOF. In view of CLIQUE $\in \mathsf{NP} \subseteq \mathsf{NP/poly}$ and Theorem 6.1, there is a polynomial-size monotone HGP P computing CLIQUE. Suppose P is based on a hypergraph H and $\boldsymbol{Q}_H$ is the OMQ for H constructed in Section 5.1. By Theorem 5.2 (*ii*), CLIQUE is a subfunction of the primitive evaluation function $f^{\vartriangle}_{\boldsymbol{Q}_H}$. By Theorem 4.6, if $\boldsymbol{q}'$ is a PE- or NDL-rewriting of $\boldsymbol{Q}_H$, then $f^{\vartriangle}_{\boldsymbol{Q}_H}$—and so CLIQUE—can be computed by a monotone formula or, respectively, circuit of size $O(|\boldsymbol{q}'|)$. Thus, $\boldsymbol{q}'$ must be of exponential size. If $\boldsymbol{q}'$ is an FO-rewriting of $\boldsymbol{Q}_H$ then, by Theorem 4.6, CLIQUE is computable by a Boolean formula of size $O(|\boldsymbol{q}'|)$. If $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ then CLIQUE cannot be computed by a polynomial circuit, and so $\boldsymbol{q}'$ must be of superpolynomial size. □

Our next theorem gives a complexity-theoretic characterisation of the existence of FO-rewritings for OMQs with ontologies of depth 2.

THEOREM 7.2. *The following conditions are equivalent*:

(*1*) *all OMQs with ontologies of depth* 2 *have polynomial-size FO-rewritings*;
(*2*) *all OMQs with ontologies of depth* 2 *and polynomially many tree witnesses have polynomial-size FO-rewritings*;
(*3*) $\mathsf{NP/poly} \subseteq \mathsf{NC}^1$.

PROOF. The implication (1) ⇒ (2) is obvious. To show that (2) ⇒ (3), suppose there is a polynomial-size FO-rewriting for the OMQ $\boldsymbol{Q}_H$ from the proof of Theorem 7.1, which has polynomially many tree witnesses. Then CLIQUE is computed by a polynomial-size Boolean formula. Since CLIQUE is $\mathsf{NP/poly}$-complete under NC^1-reductions, we have $\mathsf{NP/poly} \subseteq \mathsf{NC}^1$. Finally, to prove (3) ⇒ (1), assume $\mathsf{NP/poly} \subseteq \mathsf{NC}^1$. Let $\boldsymbol{Q}$ be an arbitrary OMQ with ontologies of depth 2. As observed in Section 4.1, the function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is in $\mathsf{NP} \subseteq \mathsf{NP/poly}$. Therefore, by our assumption, $f^{\blacktriangledown}_{\boldsymbol{Q}}$ can be computed by a polynomial-size formula, and so, by Theorem 4.5, $\boldsymbol{Q}$ has a polynomial-size FO-rewriting. □

7.2. Rewritings for OMQs with ontologies of depth 1

THEOREM 7.3. *Any OMQ* $\boldsymbol{Q}$ *with ontologies of depth* 1 *has a polynomial-size NDL-rewriting.*

PROOF. By Theorem 5.3, the hypergraph $\mathcal{H}(\boldsymbol{Q})$ is of degree at most 2, and so, by Proposition 4.7 (*ii*), there is a polynomial-size monotone HGP of degree at most 2 computing $f^{\triangledown}_{\boldsymbol{Q}}$. By Theorem 6.2, $\mathsf{co\text{-}mNL/poly} = \mathsf{mHGP}^2$, and so we have a polynomial-size monotone NBP computing the dual $f_{\boldsymbol{Q}}^{\triangledown *}$ of $f_{\boldsymbol{Q}}^{\triangledown}$. Since $\mathsf{mNL/poly} \subseteq \mathsf{mP/poly}$, we also have a polynomial-size monotone Boolean circuit that computes $f_{\boldsymbol{Q}}^{\triangledown *}$. By swapping AND- and OR-gates in that circuit, we obtain a polynomial-size monotone circuit computing $f_{\boldsymbol{Q}}^{\triangledown}$. It remains to apply Theorem 4.2 (*ii*). □

However, this upper bound cannot be extended to PE-rewritings:

THEOREM 7.4. *There is an OMQ* $\boldsymbol{Q}$ *with ontologies of depth* 1*, any PE-rewriting of which is of superpolynomial size* ($n^{\Omega(\log n)}$*, to be more precise*).

PROOF. Consider the monotone function REACHABILITY that takes the adjacency matrix of a directed graph G with two distinguished vertices s and t and returns 1 iff the graph G contains a directed path from s to t. It is known [Karchmer and Wigderson 1988; Jukna 2012] that REACHABILITY is computable by a polynomial-size monotone NBP (that is, belongs to $\mathsf{mNL/poly}$), but any monotone Boolean formula for REACHABILITY is of size $n^{\Omega(\log n)}$. Let f = REACHABILITY. By Theorem 6.2, there is a polynomial-size monotone HGP that is based on a hypergraph H of degree 2 and computes the dual f^* of f. Consider now the OMQ $\boldsymbol{S}_H$ for H defined in Section 5.2. By Theorem 5.5 (*ii*), f^* is a subfunction of $f_{\boldsymbol{S}_H}^{\vartriangle}$. By Theorem 4.6 (*i*), no PE-rewriting of the OMQ $\boldsymbol{S}_H$ can be shorter than $n^{\Omega(\log n)}$. □

THEOREM 7.5. *All OMQs with ontologies of depth* 1 *have polynomial-size FO-rewritings iff* $\mathsf{NL/poly} \subseteq \mathsf{NC}^1$.

PROOF. Suppose $\mathsf{NL/poly} \subseteq \mathsf{NC}^1$. Let $\boldsymbol{Q}$ be an OMQ with ontologies of depth 1. By Theorem 5.3, its hypergraph $\mathcal{H}(\boldsymbol{Q})$ is of degree 2 and polynomial size. By Proposition 4.7 (*ii*), there is a polynomial-size HGP of degree 2 that computes $f_{\boldsymbol{Q}}^{\triangledown}$. By Theorem 6.2, $f_{\boldsymbol{Q}}^{\triangledown} \in \mathsf{NL/poly}$. Therefore, by our assumption, $f_{\boldsymbol{Q}}^{\triangledown}$ can be computed by a polynomial-size Boolean formula. Finally, Theorem 4.2 (*i*) gives a polynomial-size FO-rewriting of $\boldsymbol{Q}$.

Conversely, suppose there is a polynomial-size FO-rewriting for any OMQ with ontologies of depth 1. Let f = REACHABILITY. Since $f \in \mathsf{NL} \subseteq \mathsf{NL/poly}$, by Theorem 6.2, we obtain a polynomial-size HGP computing f and based on a hypergraph H of degree 2. Consider the OMQ $\boldsymbol{S}_H$ with ontologies of depth 1 defined in Section 5.2. By Theorem 5.5 (*ii*), f a subfunction of $f_{\boldsymbol{S}_H}^{\vartriangle}$. By our assumption, $\boldsymbol{S}_H$ has a polynomial-size FO-rewriting; hence, by Theorem 4.6 (*i*), $f_{\boldsymbol{S}_H}^{\vartriangle}$ (and so f) are computed by polynomial-size Boolean formulas. Since f is $\mathsf{NL/poly}$-complete under NC^1-reductions [Razborov 1991], we obtain $\mathsf{NL/poly} \subseteq \mathsf{NC}^1$. □

7.3. Rewritings for tree-shaped OMQs with a bounded number of leaves

Since, by Theorem 6.4, the hypergraph function of a leaf-bounded OMQ can be computed by a polynomial-size NBP, we have:

THEOREM 7.6. *For any fixed* $\ell \geq 2$*, all tree-shaped OMQs with at most* ℓ *leaves have polynomial-size NDL-rewritings.*

The superpolynomial lower bound below is proved in exactly the same way as Theorem 7.4 using Theorems 6.3 and 5.9 instead of Theorems 6.2 and 5.5.

THEOREM 7.7. *There is an OMQ with ontologies of depth 2 and linear CQs any PE-rewriting of which is of superpolynomial size* ($n^{\Omega(\log n)}$*, to be more precise*).

Our next result is similar to Theorem 7.5:

THEOREM 7.8. *The following are equivalent*:

(*1*) *there exist polynomial-size FO-rewritings for all OMQs with linear CQs and ontologies of depth* 2;
(*2*) *for any fixed* ℓ*, there exist polynomial-size FO-rewritings for all tree-shaped OMQs with at most* ℓ *leaves*;
(*3*) $\mathsf{NL/poly} \subseteq \mathsf{NC}^1$.

PROOF. (1) $\Rightarrow$ (3) Suppose every OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ with linear $\boldsymbol{q}$ and $\mathcal{T}$ of depth 2 has an FO-rewriting of size $p(|\boldsymbol{Q}|)$, for some fixed polynomial p. Consider f = REACHABILITY. As f is monotone and $f \in \mathsf{NL}$, we have $f \in \mathsf{mNL/poly}$. Thus, Theorem 6.3 gives us an HGP P from $\mathsf{mTHGP}(2)$ that computes f. Let P be based on a hypergraph H, and let $\boldsymbol{T}_H$ be the OMQ with a linear CQ and an ontology of depth 2 constructed in Section 5.3. By Theorem 5.9 (*ii*), f is a subfunction of $f^{\vartriangle}_{\boldsymbol{T}_H}$. By our assumption, however, $\boldsymbol{T}_H$ has a polynomial-size FO-rewriting, and so, by Theorem 4.6 (*i*), it is computed by a polynomial-size Boolean formula. Since f is $\mathsf{NL/poly}$-complete under NC^1-reductions [Razborov 1991], we obtain $\mathsf{NL/poly} \subseteq \mathsf{NC}^1$. The implication (3) $\Rightarrow$ (2) follows from Theorems 6.4 and 4.2 (*i*), and (2) $\Rightarrow$ (1) is trivial. □

7.4. Rewritings for OMQs with PFSP and bounded treewidth

Since OMQs with the polynomial fundamental set property (PFSP, see Section 3) and CQs of bounded treewidth can be polynomially translated into monotone THGPs and $\mathsf{mTHGP} = \mathsf{mLOGCFL/poly} \subseteq \mathsf{mP/poly}$, we obtain:

THEOREM 7.9. *For any fixed* $t > 0$*, all OMQs with the PFSP and CQs of treewidth at most* t *have polynomial-size NDL-rewritings.*

Using Theorem 3.3 and the fact that OMQs with ontologies of bounded depth enjoy the PFSP, we obtain:

COROLLARY 7.10. *The following OMQs have polynomial-size NDL-rewritings*:
– *OMQs with ontologies of bounded depth and CQs of bounded treewidth*;
– *OMQs with ontologies not containing axioms of the form* $\varrho(x, y) \to \varrho'(x, y)$ (*and* (2)) *and CQs of bounded treewidth.*

Whether all OMQs without axioms of the form $\varrho(x, y) \to \varrho'(x, y)$ have polynomial-size rewritings remains open.[12]

THEOREM 7.11. *The following are equivalent*:

(*1*) *there exist polynomial-size FO-rewritings for all tree-shaped OMQs with ontologies of depth 2*;
(*2*) *there exist polynomial-size FO-rewritings for all OMQs with the PFSP and CQs of treewidth at most* t (*for any fixed* t);
(*3*) $\mathsf{LOGCFL/poly} \subseteq \mathsf{NC}^1$.

PROOF. The implication (2) $\Rightarrow$ (1) is trivial, and (1) $\Rightarrow$ (3) is proved similarly to the corresponding case of Theorem 7.8 using Theorems 6.5, 5.9 and 4.6. (3) $\Rightarrow$ (2) follows from Theorems 5.13, 6.5 and 4.5. □

7.5. Rewritings for OMQs with ontologies of depth 1 and CQs of bounded treewidth

We show finally that polynomial PE-rewritings are guaranteed to exist for OMQs with ontologies of depth 1 and CQs of bounded treewidth. By Theorem 6.7, it suffices to show that $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is computable by a THGP of bounded degree. However, since tree witnesses can

[12]A positive answer to this question given by Kikot et al. [2011] is based on a flawed proof.

be initiated by multiple roles, the THGPs constructed in Section 5.4 do not enjoy this property and require a minor modification.

Let $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ be an OMQ with $\mathcal{T}$ of depth 1. For every tree witness $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$, we take a fresh binary predicate $P_\mathfrak{t}$ (which cannot occur in any data instance) and extend $\mathcal{T}$ with the following axioms:

$$\begin{aligned} \tau(x) &\to \exists y\, P_\mathfrak{t}(x,y), && \text{if } \tau \text{ generates } \mathfrak{t},\\ P_\mathfrak{t}(x,y) &\to \varrho(x,y), && \text{if } \varrho(u,v) \in \boldsymbol{q}_\mathfrak{t}, u \in \mathfrak{t}_\mathsf{r} \text{ and } v \in \mathfrak{t}_\mathsf{i}. \end{aligned}$$

Denote the resulting ontology by $\mathcal{T}'$ and set $\boldsymbol{Q}' = (\mathcal{T}', \boldsymbol{q})$. By Theorem 5.3, the number of tree witnesses for $\boldsymbol{Q}$ does not exceed $|\boldsymbol{q}|$, and so the size of $\boldsymbol{Q}'$ is polynomial in $|\boldsymbol{Q}|$. It is easy to see that any rewriting of $\boldsymbol{Q}'$ (with $P_\mathfrak{t}$ replaced by $\bot$) is also a rewriting for $\boldsymbol{Q}$. Thus, it suffices to consider OMQs of the form $\boldsymbol{Q}'$, which will be called *explicit*.

Given an explicit OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$, we construct a THGP $P'_{\boldsymbol{Q}}$ in the same way as $P_{\boldsymbol{Q}}$ in Section 5.4 except that in the definition of E^k_i, instead of considering all types $\boldsymbol{w}_k$ of N_i, we only use $\boldsymbol{w}_k = (\boldsymbol{w}[1], \dots, \boldsymbol{w}[m])$ in which $\boldsymbol{w}[j]$ is either ε or $P_\mathfrak{t}$ for the unique tree witness $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$ with $\mathfrak{t}_\mathsf{i} = \{\lambda_j(N_i)\}$. (Since $\mathcal{T}$ is of depth 1, every tree witness $\mathfrak{t}$ has $\mathfrak{t}_\mathsf{i} = \{z\}$, for some variable z, and $\mathfrak{t}_\mathsf{i} \neq \mathfrak{t}'_\mathsf{i}$ whenever $\mathfrak{t} \neq \mathfrak{t}'$.) This modification guarantees that, for every i, the number of distinct E^k_i is bounded by 2^m. It follows that the hypergraph of $P'_{\boldsymbol{Q}}$ is of bounded degree, $2^m + 2^{2m}$ to be more precise. To establish the correctness of the modified construction, we can prove an analogue of Theorem 5.13, in which the original THGP $P_{\boldsymbol{Q}}$ is replaced by $P'_{\boldsymbol{Q}}$, and the function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is replaced by

$$f^{\blacktriangledown\prime}_{\boldsymbol{Q}} \;=\; \bigvee_{\substack{\Theta \subseteq \Theta_{\boldsymbol{Q}}\\ \text{independent}}} \Big(\bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta} p_{S(\boldsymbol{z})} \;\wedge\; \bigwedge_{\mathfrak{t} \in \Theta} \big(\bigwedge_{R(z,z') \in \boldsymbol{q}_\mathfrak{t}} p_{z=z'} \;\wedge\; \bigwedge_{z \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}} p_{\exists y P_\mathfrak{t}(z,y)} \big)\Big)$$

(which is obtained from $f^{\blacktriangledown}_{\boldsymbol{Q}}$ by always choosing $P_\mathfrak{t}$ as the predicate that initiates $\mathfrak{t}$). It is easy to see that Theorem 4.2 holds also for $f^{\blacktriangledown\prime}_{\boldsymbol{Q}}$ (with explicit $\boldsymbol{Q}$), which gives us:

THEOREM 7.12. *For any fixed $t > 0$, all OMQs with ontologies of depth 1 and CQs of treewidth at most t have polynomial-size PE-rewritings.*

For tree-shaped OMQs, we obtain an even better result. Indeed, by Theorem 5.7, $\mathcal{H}(\boldsymbol{Q})$ is a tree hypergraph; by Theorem 5.3, it is of degree at most 2, and so, by Theorem 6.8, $f^{\triangledown}_{\boldsymbol{Q}}$ is computed by a polynomial-size Π_3-circuit (which is monotone by definition). Thus, Theorem 4.2 (*i*) gives us the following (Π_3 turns into Π_4 because of the disjunction in the formula $\mathsf{tw}_\mathfrak{t}$):

THEOREM 7.13. *All tree-shaped OMQs with ontologies of depth 1 have polynomial-size Π_4-rewritings.*

8. COMBINED COMPLEXITY OF OMQ ANSWERING

The size of OMQ rewritings we investigated so far is crucial for classical OBDA, which relies upon a reduction to standard database query evaluation (under the assumption that it is efficient in real-world applications). However, this way of answering OMQs may not be optimal, and so understanding the size of OMQ rewritings does not shed much light on how hard OMQ answering actually is. For example, answering the OMQs from the proof of Theorem 7.4 via PE-rewriting requires superpolynomial time, while the graph reachability problem encoded by those OMQs is NL-complete. On the other hand, the existence of a short rewriting does not obviously imply tractability.

In this section, we analyse the *combined* complexity of answering OMQs classified according to the depth of ontologies and the shape of CQs. More precisely, our concern is the following decision problem: given an OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, a data instance $\mathcal{A}$

and a tuple $\boldsymbol{a}$ from $\mathsf{ind}(\mathcal{A})$ (of the same length as $\boldsymbol{x}$), decide whether $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$. Recall from Section 3 that $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(\boldsymbol{a})$ iff there exists a homomorphism from $\boldsymbol{q}(\boldsymbol{a})$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$.

The combined complexity of CQ evaluation has been thoroughly investigated in relational database theory. In general, evaluating CQs is NP-complete [Chandra and Merlin 1977], but becomes tractable for tree-shaped CQs [Yannakakis 1981] and bounded treewidth CQs [Chekuri and Rajaraman 2000; Grohe et al. 2001]—LOGCFL-complete, to be more precise [Gottlob et al. 2001].

The emerging combined complexity landscape for OMQ answering is summarised in Fig. 2 (b) in Section 1.3. The NP and LOGCFL lower bounds for arbitrary OMQs and tree-shaped OMQs with ontologies of bounded depth are inherited from the corresponding CQ evaluation problems. The NP upper bound for all OMQs was shown by Calvanese et al. [2007] and Artale et al. [2009], while the matching lower bound for tree-shaped OMQs by Kikot et al. [2011] and Gottlob et al. [2014]. By reduction of the reachability problem for directed graphs, one can easily show that evaluation of tree-shaped CQs with a bounded number of leaves (as well as answering OMQs with unary predicates only) is NL-hard. We now establish the remaining results.

8.1. OMQs with bounded-depth ontologies

We begin by showing that the LOGCFL upper bound for CQs of bounded treewidth [Gottlob et al. 2001] is preserved even in the presence of ontologies of bounded depth.

THEOREM 8.1. *For any fixed $d \geq 0$ and $t > 0$, answering OMQs with ontologies of depth at most d and CQs of treewidth at most t is* LOGCFL*-complete.*

PROOF. Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ with $\mathcal{T}$ of depth at most d and $\boldsymbol{q}$ of treewidth at most t. As $\mathcal{T}$ is of finite depth, $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ is finite for any $\mathcal{A}$. As LOGCFL is closed under $\mathsf{L}^{\mathsf{LOGCFL}}$ reductions [Gottlob et al. 1999] and evaluation of CQs of bounded treewidth is LOGCFL-complete, it suffices to show that $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ can be computed by an $\mathsf{L}^{\mathsf{LOGCFL}}$-transducer (a deterministic logspace Turing machine with a LOGCFL oracle). Clearly, we need only logarithmic space to represent any predicate name or individual constant from $\mathcal{T}$ and $\mathcal{A}$, as well as any word $aw \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ (since $|w| \leq d$ and d is fixed). Finally, as entailment in *OWL 2 QL* is in NL [Artale et al. 2009], each of the following problems can be decided by making a call to an NL (hence LOGCFL) oracle:

– decide whether $a\varrho_1 \dots \varrho_n \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for any $n \leq d$ and roles ϱ_i from $\mathcal{T}$;
– decide whether $u \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ belongs to $A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for a unary A from $\mathcal{T}$ and $\mathcal{A}$;
– decide whether $(u_1, u_2) \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} \times \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ is in $P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for a binary P from $\mathcal{T}$ and $\mathcal{A}$. ☐

If we restrict the number of leaves in tree-shaped OMQs, then the LOGCFL upper bound can be reduced to NL:

THEOREM 8.2. *For any fixed $d \geq 0$ and $\ell \geq 2$, answering OMQs with ontologies of depth at most d and tree-shaped CQs with at most ℓ leaves is* NL*-complete.*

PROOF. Algorithm 1 defines a non-deterministic procedure TreeQuery for deciding whether a tuple $\boldsymbol{a}$ is a certain answer to a tree-shaped OMQ $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ over $\mathcal{A}$. The procedure views $\boldsymbol{q}$ as a directed tree (we pick one of its variables z_0 as a root) and constructs a homomorphism from $\boldsymbol{q}(\boldsymbol{x})$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ on-the-fly by traversing the tree from root to leaves. The set frontier is initialised with a pair $z_0 \mapsto u_0$ representing the choice of where to map z_0. The possible choices for z_0 include $\mathsf{ind}(\mathcal{A})$ and $aw \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ such that $|w| \leq 2|\mathcal{T}| + |\boldsymbol{q}|$, which are enough to find a homomorphism if it exists [Artale et al. 2009]. This set of possible choices is denoted by U in Algorithm 1. Note that U occurs only in statements of the form '**guess** $u \in U$' and need not be materialised. Instead, we

ALGORITHM 1: Non-deterministic procedure TreeQuery for answering tree-shaped OMQs

```
Data: a tree-shaped OMQ $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, a data instance $\mathcal{A}$ and a tuple $\boldsymbol{a}$ from $\mathsf{ind}(\mathcal{A})$
Result: true if $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ and false otherwise
fix a directed tree $T$ compatible with the Gaifman graph of $\boldsymbol{q}$ and let $z_0$ be its root;
let $U = \{aw \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} \mid a \in \mathsf{ind}(\mathcal{A})$ and $|w| \leq 2|\mathcal{T}| + |\boldsymbol{q}|\}$;          /* not computed */
guess $u_0 \in U$;   /* use the definition of $U$ to check whether the guess is allowed */
check canMap($z_0$,$u_0$);
frontier ⟵ $\{z_0 \mapsto u_0\}$;
while frontier $\neq \emptyset$ do
    remove some $z \mapsto u$ from frontier;
    foreach child $z'$ of $z$ in $T$ do
        guess $u' \in U$;   /* use the def. of $U$ to check whether the guess is allowed */
        check $(u, u') \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $P(z, z') \in \boldsymbol{q}$, and canMap($z'$,$u'$);
        frontier ⟵ frontier $\cup \{z' \mapsto u'\}$
return true;

Function canMap($z$, $u$)
    if $z$ is the $i$th answer variable and $u \neq a_i$ then return false;
    if $u = aw\varrho$ then      /* the element $u$ is in the tree part of the canonical model */
        check $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$, for all $A(z) \in \boldsymbol{q}$, and $\mathcal{T} \models P(x, x)$, for all $P(z, z) \in \boldsymbol{q}$
    else                                                   /* otherwise, $u \in \mathsf{ind}(\mathcal{A})$ */
        check $u \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $A(z) \in \boldsymbol{q}$, and $(u, u) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $P(z, z) \in \boldsymbol{q}$
    return true;
```

assume that the sequence u is guessed element-by-element and the condition $u \in U$ is verified along the sequence of guesses. We use the subroutine call canMap(z_0, u_0) to check whether the guessed u_0 is compatible with z_0.[13] It first ensures that, if z_0 is an answer variable of $\boldsymbol{q}(\boldsymbol{x})$, then u_0 is the individual constant corresponding to z_0 in $\boldsymbol{a}$. Next, if $z_0 \in \mathsf{ind}(\mathcal{A})$, then it verifies that u_0 satisfies all atoms in $\boldsymbol{q}(\boldsymbol{x})$ that involve only z_0. If $u_0 \notin \mathsf{ind}(\mathcal{A})$, then u_0 must take the form $aw\varrho$ and the subroutine checks whether $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$ (equivalently, $aw\varrho \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$) for every $A(z_0) \in \boldsymbol{q}$ and whether $\mathcal{T} \models P(x, x)$ for every $P(z_0, z_0) \in \boldsymbol{q}$. The remainder of the algorithm consists of a while loop, in which we remove $z \mapsto u$ from frontier, and if z is not a leaf node, guess where to map its children. We must then check that the guessed element u' for child z' is compatible with (*i*) the binary atoms linking z to z' and (*ii*) the atoms that involve only z'; the latter is done by canMap(z', u'). If the check succeeds, we add $z' \mapsto u'$ to frontier, for each child z' of z; otherwise, false is returned. 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 each variable in $\boldsymbol{q}(\boldsymbol{x})$.

Correctness and termination of the algorithm are straightforward and hold for tree-shaped OMQs with arbitrary ontologies. Membership in NL for bounded-depth ontologies and bounded-leaf queries follows from the fact that the number of leaves of $\boldsymbol{q}$ does not exceed ℓ, in which case the cardinality of frontier is bounded by ℓ, and the fact that the depth of $\mathcal{T}$ does not exceed d, in which case every element of U requires only a fixed amount of space to store. So, since each variable z can be stored in logarithmic space, the set frontier can also be stored in logarithmic space. Finally, it should be clear that the subroutine canMap(z, u) can also be implemented in NL [Artale et al. 2009]. □

[13]The operator **check** immediately returns false if the condition is not satisfied.

8.2. OMQs with bounded-leaf CQs

It remains to settle the complexity of answering OMQs with arbitrary ontologies and bounded-leaf CQs, for which neither the upper bounds from the preceding subsection nor the NP lower bound by Kikot et al. [2011] are applicable.

THEOREM 8.3. *For any fixed $\ell \geq 2$, answering OMQs with tree-shaped CQs having at most ℓ leaves is* LOGCFL*-complete.*

PROOF. First, we establish the upper bound using a characterisation of the class LOGCFL in terms of non-deterministic auxiliary pushdown automata (NAuxPDAs). An NAuxPDA [Cook 1971] is a non-deterministic Turing machine with an additional work tape constrained to operate as a pushdown store. Sudborough [1978] showed that LOGCFL coincides with the class of problems that can be solved by NAuxPDAs running in logarithmic space and polynomial time (note that the space on the pushdown tape is not subject to the logarithmic space bound). Algorithms 2 and 3 give a procedure BLQuery for answering OMQs with bounded-leaf CQs that can be implemented by an NAuxPDA.

Similarly to TreeQuery, the idea is to view the input CQ $\boldsymbol{q}(\boldsymbol{x})$ as a tree and iteratively construct a homomorphism from $\boldsymbol{q}(\boldsymbol{x})$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, working from root to leaves. We begin by guessing an element $a_0 w$ to which the root variable z_0 is mapped and checking that $a_0 w$ is compatible with z_0. However, instead of storing directly $a_0 w$ in frontier, we guess it element-by-element and push the word w onto the stack, stack. We assume that we have access to the top of the stack, denoted by top(stack), and the call top(stack) on empty stack returns ε. During execution of BLQuery, the height of the stack will never exceed $2|\mathcal{T}| + |\boldsymbol{q}|$, and so we assume that the height of the stack, denoted by |stack|, is also available as, for example, a variable whose value is updated by the push and pop operations on stack.

After having guessed $a_0 w$, we check that z_0 can be mapped to it, which is done by calling `canMapTail(`z_0`, `a_0`, top(stack))`. If the check succeeds, we initialise frontier to the set of 4-tuples of the form $(z_0 \mapsto (a_0, |\mathsf{stack}|), z_i)$, for all children z_i of z_0 in T. Intuitively, a tuple $(z \mapsto (a, n), z')$ records that the variable z is mapped to the element $a\,\mathsf{stack}_{\leq n}$ and that the child z' of z remains to be mapped (in the explanations we use $\mathsf{stack}_{\leq n}$ to refer to the word comprising the first n symbols of stack; the algorithm, however, cannot make use of it).

In the main loop, we remove one or more tuples from frontier, choose where to map the variables and update frontier and stack accordingly. There are four options. Option 1 is used for tuples $(z \mapsto (a, 0), z')$ where both z and z' are mapped to individual constants, Option 2 (Option 3) for tuples $(z \mapsto (a, n), z')$ in which we map z' to a child (respectively, parent) of the image of z in $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, while Option 4 applies when z and z' are mapped to the same element (which is possible if $P(z, z') \in \boldsymbol{q}$, for some P that is reflexive according to $\mathcal{T}$). Crucially, however, the order in which tuples are treated matters due to the fact that several tuples 'share' the single stack. Indeed, when applying Option 3, we pop a symbol from stack, and may therefore lose some information that is needed for processing other tuples. To avoid this, Option 3 may only be applied to tuples $(z \mapsto (a, n), z')$ with maximal n, and it must be applied to *all* such tuples at the same time. For Option 2, we require that the selected tuple $(z \mapsto (a, n), z')$ is such that $n = |\mathsf{stack}|$: since z' is being mapped to an element $a\,\mathsf{stack}_{\leq n}\,\varrho$, we need to access the nth symbol in stack to determine the possible choices for ϱ and to record the symbol chosen by pushing it onto stack.

The procedure terminates and returns true when frontier is empty, meaning that we have successfully constructed a homomorphism witnessing that the input tuple is an answer. Conversely, given a homomorphism from $\boldsymbol{q}(\boldsymbol{a})$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, we can define a success-

ALGORITHM 2: Non-deterministic procedure BLQuery for answering bounded-leaf OMQs.

```
Data: a bounded-leaf OMQ $(\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, a data instance $\mathcal{A}$ and a tuple $\boldsymbol{a}$ from $\mathsf{ind}(\mathcal{A})$
Result: true if $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ and false otherwise
fix a directed tree $T$ compatible with the Gaifman graph of $\boldsymbol{q}$ and let $z_0$ be its root;
guess $a_0 \in \mathsf{ind}(\mathcal{A})$;                                   /* guess the ABox element */
guess $n_0 < 2|\mathcal{T}| + |\boldsymbol{q}|$;                 /* maximum distance from ABox of relevant elements */
foreach $n$ in $1, \dots, n_0$ do     /* guess the initial element in a step-by-step fashion */
    guess a role $\varrho$ in $\mathcal{T}$ such that isGenerated($\varrho$, $a_0$, top(stack));
    push $\varrho$ on stack
check canMapTail($z_0$, $a_0$, top(stack));
frontier $\longleftarrow \big\{(z_0 \mapsto (a_0, |\mathsf{stack}|), z_i) \mid z_i$ is a child of $z_0$ in $\mathcal{T}\big\}$;
while frontier $\neq \emptyset$ do
    guess one of the 4 options;
    if Option 1 then                                                    /* take a step in ind($\mathcal{A}$) */
        remove some $(z \mapsto (a, 0), z')$ from frontier;
        guess $a' \in \mathsf{ind}(\mathcal{A})$;
        check $(a, a') \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $P(z, z') \in \boldsymbol{q}$, and canMapTail($z'$, $a'$, $\varepsilon$);
        frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a', 0), z'_i) \mid z'_i$ is a child of $z'$ in $T\}$
    else if Option 2 and $|\mathsf{stack}| < 2|\mathcal{T}| + |\boldsymbol{q}|$ then  /* a step 'forward' in the tree part */
        remove some $(z \mapsto (a, |\mathsf{stack}|), z')$ from frontier;
        guess a role $\varrho$ in $\mathcal{T}$ such that isGenerated($\varrho$, $a$, top(stack));
        push $\varrho$ on stack;
        check $\mathcal{T} \models \varrho(x, y) \to P(x, y)$, for all $P(z, z') \in \boldsymbol{q}$, and canMapTail($z'$, $a$, top(stack));
        frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a, |\mathsf{stack}|), z'_i) \mid z'_i$ is a child of $z'$ in $T\}$
    else if Option 3 and $|\mathsf{stack}| > 0$ then    /* take a step 'backward' in the tree part */
        let deepest $= \{(z \mapsto (a, n), z') \in$ frontier $\mid n = |\mathsf{stack}|\}$;            /* may be empty */
        remove all deepest from frontier;
        pop $\varrho$ from stack;
        foreach $(z \mapsto (a, n), z') \in$ deepest do
            check $\mathcal{T} \models \varrho(x, y) \to P(x, y)$, for all $P(z', z) \in \boldsymbol{q}$, and canMapTail($z'$, $a$, top(stack));
            frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a, |\mathsf{stack}|), z'_i) \mid z'_i$ is a child of $z'$ in $T\}$
    else if Option 4 then                 /* take a 'loop'-step in the tree part of $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ */
        remove some $(z \mapsto (a, |\mathsf{stack}|), z')$ from frontier;
        check $\mathcal{T} \models P(x, x)$, for all $P(z, z') \in \boldsymbol{q}$, and canMapTail($z'$, $a$, top(stack));
        frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a, |\mathsf{stack}|), z'_i) \mid z'_i$ is a child of $z'$ in $T\}$
    else return false;
return true;
```

ful execution of BLQuery. We prove in Appendix H that BLQuery terminates (Proposition H.1), is correct (Proposition H.2) and can be implemented by an NAuxPDA (Proposition H.3). The following example illustrates the construction.

Example 8.4. Suppose $\mathcal{T}$ has the following axioms:

$$A(x) \to \exists y\, P(x, y), \qquad P(x, y) \to U(y, x),$$
$$\exists y\, P(y, x) \to \exists y\, S(x, y), \qquad \exists y\, S(y, x) \to \exists y\, T(y, x), \qquad \exists y\, P(y, x) \to B(x).$$

the query is as follows:

$$\boldsymbol{q}(x_1, x_2) \;=\; \exists y_1 y_2 y_3 y_4 y_5 \big(R(y_2, x_1) \wedge P(y_2, y_1) \wedge S(y_1, y_3) \wedge T(y_5, y_3) \wedge S(y_4, y_3) \wedge U(y_4, x_2)\big)$$

ALGORITHM 3: Subroutines for BLQuery.

```
Function canMapTail($z$, $a$, $\sigma$)
    if $z$ is the $i^{th}$ answer variable and either $a \neq a_i$ or $\sigma \neq \varepsilon$ then return false;
    if $\sigma \neq \varepsilon$ then            /* an element of the form $a \dots \sigma$ in the tree part */
        check $\mathcal{T} \models \exists y\, \sigma(y,x) \to A(x)$, for all $A(z) \in \boldsymbol{q}$, and $\mathcal{T} \models P(x,x)$, for all $P(z,z) \in \boldsymbol{q}$
    else                                          /* otherwise, in ind($\mathcal{A}$) */
        check $a \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $A(z) \in \boldsymbol{q}$, and $(a,a) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $P(z,z) \in \boldsymbol{q}$
    return true;

Function isGenerated($\varrho$, $a$, $\sigma$)
    if $\sigma \neq \varepsilon$ then            /* an element of the form $a \dots \sigma$ in the tree part */
        check $\mathcal{T} \models \exists y\, \sigma(y,x) \to \exists y\, \varrho(x,y)$
    else                                          /* otherwise, in ind($\mathcal{A}$) */
        check $(a,b) \in \varrho(x,y)^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for some $b \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} \setminus \mathsf{ind}(\mathcal{A})$
    return true;
```

and $\mathcal{A} = \{A(a), R(a,c)\}$. Observe that $\mathcal{C}_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(c,a)$. We show how to define an execution of BLQuery that returns true on $((\mathcal{T},\boldsymbol{q}),\mathcal{A},\boldsymbol{q},(c,a))$ and the homomorphism it induces. We fix some variable, say y_1, as the root of the query tree. We then guess the constant a and the word P, push P onto stack and check using canMapTail(y_1, a, P) that our choice is compatible with y_1. At the start of the while loop, we have

$$\mathsf{frontier} = \{(y_1 \mapsto (a,1), y_2), (y_1 \mapsto (a,1), y_3)\} \quad \text{and} \quad \mathsf{stack} = P, \tag{w-1}$$

where the first tuple, for example, records that y_1 has been mapped to $a\,\mathsf{stack}_{\leq 1} = aP$ and y_2 remains to be mapped. We are going to use Option 3 for $(y_1 \mapsto (a,1), y_2)$ and Option 2 for $(y_1 \mapsto (a,1), y_3)$. We (have to) start with Option 2 though: we remove $(y_1 \mapsto (a,1), y_3)$ from frontier, guess S, push it onto stack, and add $(y_3 \mapsto (a,2), y_4)$ and $(y_3 \mapsto (a,2), y_5)$ to frontier. Note that the tuples in frontier allow us to read off the elements $a\,\mathsf{stack}_{\leq 1}$ and $a\,\mathsf{stack}_{\leq 2}$ to which y_1 and y_3 are mapped. Thus,

$$\mathsf{frontier} = \{(y_1 \mapsto (a,1), y_2), (y_3 \mapsto (a,2), y_4), (y_3 \mapsto (a,2), y_5)\} \quad \text{and} \quad \mathsf{stack} = PS \tag{w-2}$$

at the start of the second iteration of the while loop. We are going to use Option 3 for $(y_3 \mapsto (a,2), y_4)$ and Option 2 for $(y_3 \mapsto (a,2), y_5)$. Again, we have to start with Option 2: we remove $(y_3 \mapsto (a,2), y_5)$ from frontier, and guess T^- and push it onto stack. As y_5 has no children, we leave frontier unchanged. At the start of the third iteration,

$$\mathsf{frontier} = \{(y_1 \mapsto (a,1), y_2), (y_3 \mapsto (a,2), y_4)\} \quad \text{and} \quad \mathsf{stack} = PST^-; \tag{w-3}$$

see Fig. 19 (a). We apply Option 3 and, since $\mathsf{deepest} = \emptyset$, we pop T^- from stack but make no other changes. In the fourth iteration, we again apply Option 3. Since $\mathsf{deepest} = \{(y_3 \mapsto (a,2), y_4)\}$, we remove this tuple from frontier and pop S from stack. As the checks succeed for S, we add $(y_4 \mapsto (a,1), x_2)$ to frontier. Before the fifth iteration,

$$\mathsf{frontier} = \{(y_1 \mapsto (a,1), y_2), (y_4 \mapsto (a,1), x_2)\} \quad \text{and} \quad \mathsf{stack} = P; \tag{w-5}$$

see Fig. 19 (b). We apply Option 3 with $\mathsf{deepest} = \{(y_1 \mapsto (a,1), y_2), (y_4 \mapsto (a,1), x_2)\}$. This leads to both tuples being removed from frontier and P popped from stack. We next perform the required checks and, in particular, verify that the choice of where to map the answer variable x_2 agrees with the input vector (c,a) (which is indeed the case). Then, we add $(y_2 \mapsto (a,0), x_1)$ to frontier. The final, sixth, iteration begins with

$$\mathsf{frontier} = \{(y_2 \mapsto (a,0), x_1)\} \quad \text{and} \quad \mathsf{stack} = \varepsilon; \tag{w-6}$$

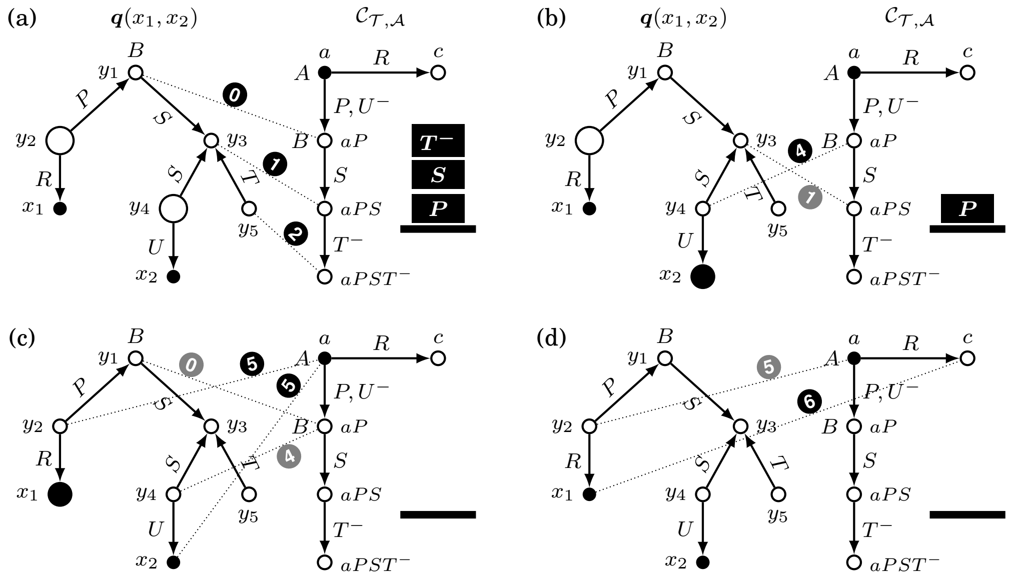


Fig. 19. Partial homomorphisms from a tree-shaped CQ $q(x_1, x_2)$ to the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}}$ and the contents of stack in Example 8.4: (a) before the third iteration, (b) before the fifth iteration, (c) before and (d) after the final (sixth) iteration. Large nodes indicate the last component of the tuples in frontier.

see Fig. 19 (c). We choose Option 1, remove $(y_2 \mapsto (a, 0), x_1)$ from frontier, guess c, and perform the required compatibility checks. As x_1 is a leaf, no new tuples are added to frontier; see Fig. 19 (d). We are thus left with frontier $= \emptyset$, and return true.

The proof of LOGCFL-hardness is by reduction of the following problem: decide whether an input of length n is accepted by the nth circuit of a *logspace-uniform* family of SAC^1 circuits, which is known to be LOGCFL-hard [Venkateswaran 1991]. This problem was used by Gottlob et al. [2001] to show LOGCFL-hardness of evaluating tree-shaped CQs. We follow a similar approach, but with one crucial difference: using an ontology, we 'unravel' the circuit into a tree, which allows us to replace tree-shaped CQs by linear ones. Following Gottlob et al. [2001], we assume without loss of generality 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 $i - 1$, the input gates on level 1 and the output gate on the greatest level;
– the number of levels is odd, all even-level gates are OR-gates, and all odd-level non-input gates are AND-gates.

It is well known [Gottlob et al. 2001; Venkateswaran 1991] that a circuit in normal form accepts an input α iff there is a labelled rooted tree (called a *proof tree*) such that

– the root node is labelled with the output AND-gate;
– if a node is labelled with an AND-gate g_i and $g_i = g_j \wedge g_k$, then it has two children labelled with g_j and g_k, respectively;
– if a node is labelled with an OR-gate g_i and $g_i = g_{j_1} \vee \ldots \vee g_{j_k}$, then it has a unique child that is labelled with one of $g_{j_1}, \ldots, g_{j_k}$;
– every leaf node is labelled with an input gate whose literal evaluates to 1 under α.

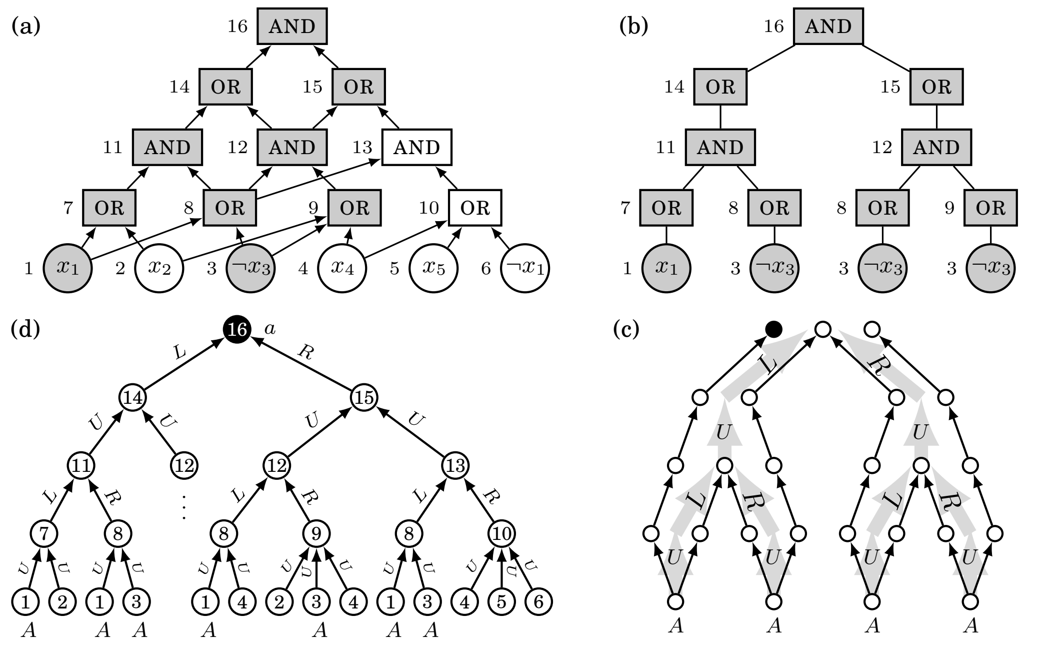


Fig. 20. (a) A circuit $\boldsymbol{C}$ of 5 levels with input $\boldsymbol{\alpha}$: $x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_3 \mapsto 0,\ x_4 \mapsto 0,\ x_5 \mapsto 0$ (the gate number is indicated on the left and gates with value 1 under $\boldsymbol{\alpha}$ are shaded); (b) a proof tree for $\boldsymbol{C}$ and $\boldsymbol{\alpha}$; (c) CQs $\boldsymbol{q}$ (thick gray arrows) and $\boldsymbol{q}'$ (black arrows); (d) canonical model of $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ with the subscript of G_i inside the nodes.

For example, the circuit in Fig. 20 (a) accepts $(1, 0, 0, 0, 1)$, as witnessed by the proof tree in Fig. 20 (b). 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 such that $\boldsymbol{C}$ accepts $\boldsymbol{\alpha}$ iff some labelling of T is a proof tree for $\boldsymbol{C}$ and $\boldsymbol{\alpha}$. Note that T depends only on the number of levels in $\boldsymbol{C}$. The reduction [Gottlob et al. 2001], which is for presentation purposes reproduced here with minor modifications, encodes $\boldsymbol{C}$ and $\boldsymbol{\alpha}$ in the database and uses a Boolean tree-shaped CQ based on the skeleton proof tree. Specifically, the database $D(\boldsymbol{\alpha})$ uses the gates of $\boldsymbol{C}$ as constants and consists of the following facts:

$L(g_j, g_i)$ and $R(g_k, g_i)$,	for every AND-gate g_i with $g_i = g_j \wedge g_k$;
$U(g_{j_1}, g_i), \dots, U(g_{j_k}, g_i)$,	for every OR-gate g_i with $g_i = g_{j_1} \vee \cdots \vee g_{j_k}$;
$A(g_i)$,	for every input gate g_i whose value is 1 under $\boldsymbol{\alpha}$.

The CQ $\boldsymbol{q}$ uses the nodes of T as variables, has an atom $U(z_j, z_i)$ ($L(z_j, z_i)$, $R(z_j, z_i)$) for every node z_i with unique (left, right) child z_j, and has an atom $A(z_i)$ for every leaf node z_i. These definitions guarantee that $D(\boldsymbol{\alpha}) \models \boldsymbol{q}$ iff $\boldsymbol{C}$ accepts $\boldsymbol{\alpha}$; moreover, both $\boldsymbol{q}$ and $D(\boldsymbol{\alpha})$ can be constructed by logspace transducers.

To adapt this reduction to our setting, we replace $\boldsymbol{q}$ by a linear CQ $\boldsymbol{q}'$, which is obtained by a depth-first traversal of $\boldsymbol{q}$. When evaluated on $D(\boldsymbol{\alpha})$, the CQs $\boldsymbol{q}'$ and $\boldsymbol{q}$ may give different answers, but the answers coincide if the CQs are evaluated on the *unravelling of* $D(\boldsymbol{\alpha})$ *into a tree*. Thus, we define $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ whose canonical model induces a tree isomorphic to the unravelling of $D(\boldsymbol{\alpha})$. To formally introduce $\boldsymbol{q}'$, consider the sequence of words defined inductively as follows:

$$w_0 = \varepsilon \quad \text{and} \quad w_{j+1} = L^- \, U^- \, w_j \, U \, L \, R^- \, U^- \, w_j \, U \, R, \text{ for } j > 0.$$

Suppose $\boldsymbol{C}$ has $2d+1$ levels, $d \geq 0$. Consider the dth word $w_d = \varrho_1\varrho_2\dots\varrho_k$ and take

$$\boldsymbol{q}'(y_0) \quad = \quad \exists y_1, \dots, y_k \Big[\bigwedge_{i=1}^{k} \varrho_i(y_{i-1}, y_i) \quad \wedge \bigwedge_{\varrho_i\varrho_{i+1} = U^-\,U} A(y_i) \Big];$$

see Fig. 20 (c). We now define $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$. Suppose $\boldsymbol{C}$ has gates $g_1, \dots, g_m$, with g_m the output gate. In addition to predicates U, L, R, A, we introduce a unary predicate G_i for each gate g_i. We set $\mathcal{A} = \{G_m(a)\}$ and include the following axioms in $\mathcal{T}_{\boldsymbol{\alpha}}$:

$$G_i(x) \to \exists y\,\big(S(x,y) \wedge G_j(y)\big), \qquad \text{for every } S(g_j, g_i) \in D(\boldsymbol{\alpha}),\ S \in \{U, L, R\},$$
$$G_i(x) \to A(x), \qquad \text{for every } A(g_i) \in D(\boldsymbol{\alpha});$$

see Fig. 20 (d) for an illustration. When restricted to predicates U, L, R, A, the canonical model of $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ is isomorphic to the unravelling of $D(\boldsymbol{\alpha})$ starting from g_m.

We show in Appendix I that $\boldsymbol{q}'$ and $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ can be constructed by logspace transducers (Proposition I.1), and that $\boldsymbol{C}$ accepts $\boldsymbol{\alpha}$ iff $\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A} \models \boldsymbol{q}'(a)$ (Proposition I.2). □

9. CONCLUSIONS AND OPEN PROBLEMS

Our aim in this work was to understand how the size of OMQ rewritings and the combined complexity of OMQ answering depend on (*i*) the existential depth of *OWL 2 QL* ontologies, (*ii*) the treewidth of CQs or the number of leaves in tree-shaped CQs, and (*iii*) the type of rewriting: PE, NDL or arbitrary FO.

We tackled the succinctness problem by representing OMQ rewritings as (Boolean) hypergraph functions and establishing an unexpectedly tight correspondence between the size of OMQ rewritings and the size of various computational models for computing these functions. It turned out that polynomial-size *PE-rewritings* can only be constructed for OMQs with ontologies of depth 1 and CQs of bounded treewidth. Ontologies of larger depth require, in general, PE-rewritings of super-polynomial size. The good and surprising news, however, is that, for classes of OMQs with ontologies of bounded depth and CQs of bounded treewidth, we can always (efficiently) construct polynomial-size *NDL-rewritings*. The same holds if we consider OMQs obtained by pairing ontologies of depth 1 with arbitrary CQs or coupling arbitrary ontologies with bounded-leaf queries; see Fig. 2 for details. The existence of polynomial-size *FO-rewritings* for different classes of OMQs was shown to be equivalent to major open problems in computational and circuit complexity such as '$\mathsf{NL}/\mathsf{poly} \subseteq \mathsf{NC}^1$?', '$\mathsf{LOGCFL}/\mathsf{poly} \subseteq \mathsf{NC}^1$?' and '$\mathsf{NP}/\mathsf{poly} \subseteq \mathsf{NC}^1$?'

We also determined the combined complexity of answering OMQs from the considered classes. In particular, we showed that OMQ answering is tractable—either NL- or LOGCFL-complete—for bounded-depth ontologies coupled with bounded treewidth CQs, as well as for arbitrary ontologies paired with tree-shaped queries with a bounded number of leaves. We point out that membership in LOGCFL implies that answering OMQs from the identified tractable classes can be 'profitably parallelised' (for details, consult [Gottlob et al. 2001]).

Comparing the two sides of Fig. 2, we remark that the class of tractable OMQs nearly coincides with the OMQs admitting polynomial-size NDL-rewritings (the only exception being OMQs with ontologies of depth 1 and arbitrary CQs). However, the LOGCFL and NL membership results cannot be immediately inferred from the existence of polynomial-size NDL-rewritings, since evaluating polynomial-size NDL-queries is a PSPACE-complete problem in general. In fact, much more work is required to construct NDL-rewritings that can be evaluated in LOGCFL and NL, which will be done in a follow-up publication; see technical report [Bienvenu et al. 2016].

Although the present work gives comprehensive solutions to the succinctness and combined complexity problems formulated in Section 1, it also raises some interesting and challenging questions:

(1) What is the size of rewritings of OMQs with a *fixed ontology*?
(2) What is the size of rewritings of OMQs with ontologies in a *fixed signature*?
(3) Is answering OMQs with CQs of bounded treewidth and ontologies of finite depth fixed-parameter tractable if the *ontology depth* is the parameter?
(4) What is the size of rewritings for OMQs whose ontologies do not contain role inclusions, that is, axioms of the form $\varrho(x, y) \to \varrho'(x, y)$?

Answering these questions would provide further insight into the difficulty of OBDA and could lead to the identification of new classes of well-behaved OMQs.

As far as practical OBDA is concerned, our experience with the query answering engine Ontop [Rodriguez-Muro et al. 2013; Kontchakov et al. 2014], which employs the tree-witness rewriting, shows that mappings and database constraints together with semantic query optimisation techniques can drastically reduce the size of rewritings and produce efficient SQL queries over the data. The role of mappings and data constraints in OBDA is yet to be fully investigated [Rodriguez-Muro and Calvanese 2012; Rosati 2012; Lembo et al. 2015; Bienvenu and Rosati 2015] and constitutes another promising avenue for future work.

Finally, the focus of this paper was on the ontology language *OWL 2 QL* that has been designed specifically for OBDA via query rewriting. However, in practice ontology designers often require constructs that are not available in *OWL 2 QL*. Typical examples are axioms such as $A(x) \to B(x) \lor C(x)$ and $P(x, y) \land A(y) \to B(x)$. The former is a standard covering constraint in conceptual modelling, while the latter occurs in ontologies such as SNOMED CT. There are at least two ways of extending the applicability of rewriting techniques to a wider class of ontology languages. A first approach relies upon the observation that although many ontology languages do not guarantee the existence of rewritings for all ontology-query pairs, it may still be the case that the queries and ontologies typically encountered in practice do admit rewritings. This has motivated the development of diverse methods for identifying particular ontologies and OMQs for which (first-order or Datalog) rewritings exist [Lutz et al. 2011; Bienvenu et al. 2013; Bienvenu et al. 2014; Kaminski et al. 2014; Hansen et al. 2015]. A second approach consists in replacing an ontology formulated in a complex ontology language (which lacks efficient query answering algorithms) by an ontology written in a simpler language, for which query rewriting methods can be employed. Ideally, one would show that the simpler ontology is equivalent to the original with regards to query answering [Botoeva et al. 2016], and thus provides the exact set of answers. Alternatively, one can use a simpler ontology to approximate the answers for the full one [Console et al. 2014; Botoeva et al. 2016] (possibly employing a more costly complete algorithm to decide the status of the remaining candidate answers [Zhou et al. 2015]).

REFERENCES

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

N. Alon and R. Boppana. 1987. The monotone circuit complexity of Boolean functions. *Combinatorica* 7, 1 (1987), 1–22.

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

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

B. Aspvall, M. Plass, and R. Tarjan. 1979. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Process. Lett.* 8, 3 (1979), 121–123.

J. Avigad. 2003. Eliminating definitions and Skolem functions in first-order logic. *ACM Transactions on Computational Logic* 4, 3 (2003), 402–415.

J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175, 9–10 (2011), 1620–1654.

M. Bienvenu, S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev. 2016. Theoretically Optimal Datalog Rewritings for OWL 2 QL Ontology-Mediated Queries. *CoRR* abs/1604.05258 (2016). http://arxiv.org/abs/1604.05258

M. Bienvenu, C. Lutz, and F. Wolter. 2013. First-Order Rewritability of Atomic Queries in Horn Description Logics. In *Proc. of the 23nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*. IJCAI/AAAI, 754–760.

M. Bienvenu, M. Ortiz, and M. Simkus. 2015. Regular Path Queries in Lightweight Description Logics: Complexity and Algorithms. *Journal of Artificial Intelligence Research (JAIR)* 53 (2015), 315–374.

M. Bienvenu, M. Ortiz, M. Simkus, and G. Xiao. 2013. Tractable Queries for Lightweight Description Logics. In *Proc. of the 23nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*. IJCAI/AAAI, 768–774.

M. Bienvenu and R. Rosati. 2015. Query-based comparison of OBDA specifications. In *Proc. of the 28th International Workshop on Description Logics (DL 2015) (CEUR Workshop Proceedings)*, Vol. 1350. CEUR-WS, 55–66.

M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. 2014. Ontology-Based Data Access: A Study through Disjunctive Datalog, CSP, and MMSNP. *ACM Transasctions on Database Systems* 39, 4 (2014), 33:1–44.

E. Botoeva, D. Calvanese, V. Santarelli, D. F. Savo, A. Solimando, and G. Xiao. 2016. Beyond OWL 2 QL in OBDA: Rewritings and Approximations. In *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI 2016)*. AAAI.

A. Brandstädt, V. B. Le, and J. P. Spinrad. 1999. *Graph Classes: A Survey*. SIAM, Philadelphia, PA, USA.

A. Bretto. 2013. *Hypergraph Theory: An Introduction*. Springer.

A. Calì, G. Gottlob, and T. Lukasiewicz. 2012a. A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics* 14 (2012), 57–83.

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

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

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

A. Chandra and P. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *Conference Record of the 9th Annual ACM Symposium on Theory of Computing (STOC'77)*. ACM, 77–90.

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

A. Chortaras, D. Trivela, and G. Stamou. 2011. Optimized Query Rewriting for OWL 2 QL. In *Proc. of CADE-23 (LNCS)*, Vol. 6803. Springer, 192–206.

C. Civili and R. Rosati. 2012. A Broad Class of First-Order Rewritable Tuple-Generating Dependencies. In *Proc. of the 2nd Int. Datalog 2.0 Workshop (Lecture Notes in Computer Science)*, Vol. 7494. Springer, 68–80.

M. Console, J. Mora, R. Rosati, V. Santarelli, and D. F. Savo. 2014. Effective Computation of Maximal Sound Approximations of Description Logic Ontologies. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part II (Lecture Notes in Computer Science)*, Vol. 8797. Springer, 164–179.

S. A. Cook. 1971. Characterizations of Pushdown Machines in Terms of Time-Bounded Computers. *Journal of the ACM* 18, 1 (1971), 4–18.

T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao. 2012. Query Rewriting for Horn-SHIQ Plus Rules. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI 2012)*. AAAI, 726–733.

C. Flament. 1978. Hypergraphes arborés. *Discrete Mathematics* 21, 3 (1978), 223–227.

J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory*. Springer.

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

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

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

G. Gottlob, N. Leone, and F. Scarcello. 2001. The complexity of acyclic conjunctive queries. *Journal of the ACM* 48, 3 (2001), 431–498.

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

G. Gottlob and T. Schwentick. 2012. Rewriting Ontological Queries into Small Nonrecursive Datalog Programs. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012)*. AAAI, 254–263.

M. Grohe, T. Schwentick, and L. Segoufin. 2001. When is the evaluation of conjunctive queries tractable?. In *Proc. of the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001)*. ACM, 657–666.

V. Gutiérrez-Basulto, Y. Ibáñez-García, R. Kontchakov, and E. V. Kostylev. 2015. Queries with negation and inequalities over lightweight ontologies. *J. Web Sem.* 35 (2015), 184–202.

P. Hansen, C. Lutz, I. Seylan, and F. Wolter. 2015. Efficient Query Rewriting in the Description Logic EL and Beyond. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015)*. AAAI, 3034–3040.

D. A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the Institute of Radio Engineers* 40, 9 (1952), 1098–1101.

N. Immerman. 1988. Nondeterministic Space is Closed Under Complementation. *SIAM J. Comput.* 17, 5 (1988), 935–938.

D. S. Johnson and A. C. Klug. 1982. Testing Containment of Conjunctive Queries Under Functional and Inclusion Dependencies. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS)*. ACM, 164–169.

S. Jukna. 2012. *Boolean Function Complexity — Advances and Frontiers*. Algorithms and combinatorics, Vol. 27. Springer.

M. Kaminski, Y. Nenov, and B. Cuenca Grau. 2014. Datalog Rewritability of Disjunctive Datalog Programs and its Applications to Ontology Reasoning. In *Proc. of the 28th AAAI Conference on Artificial Intelligence (AAAI 2014)*. AAAI, 1077–1083.

M. Karchmer and A. Wigderson. 1988. Monotone Circuits for Connectivity Require Super-logarithmic Depth. In *Proc. of the 20th Annual ACM Symposium on Theory of Computing (STOC '88)*. ACM, 539–550.

E. Kharlamov, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M. G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, and I. Horrocks. 2015. Ontology Based Access to Exploration Data at Statoil. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC 2015), Part II (Lecture Notes in Computer Science)*, Vol. 9367. Springer, 93–112.

E. Kharlamov, N. Solomakhina, Ö. L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, and S. Watson. 2014. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part I (Lecture Notes in Computer Science)*, Vol. 8796. Springer, 601–619.

S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev. 2012b. Exponential Lower Bounds and Separation for Query Rewriting. In *Proc. of the 39th Int. Colloquium on Automata, Languages and Programming (ICALP 2012) (Lecture Notes in Computer Science)*, Vol. 7392. Springer, 263–274.

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

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

M. König, M. Leclère, and M.-L. Mugnier. 2015a. Query Rewriting for Existential Rules with Compiled Preorder. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 3106–3112.

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

R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. 2010. The Combined Approach to Query Answering in DL-Lite. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 12th Int. Conf. KR 2010*. AAAI Press, 247–257.

R. Kontchakov, M. Rezk, M. Rodriguez-Muro, G. Xiao, and M. Zakharyaschev. 2014. Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part I (Lecture Notes in Computer Science)*, Vol. 8796. Springer, 552–567.

E. V. Kostylev, J. L. Reutter, and D. Vrgoc. 2015. XPath for DL Ontologies. In *Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI, 1525–1531.

D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. 2015. Mapping Analysis in Ontology-Based Data Access: Algorithms and Complexity. In *Proc. of the 14th Int. Semantic Web Conf. (ISWC 2015) (Lecture Notes in Computer Science)*, Vol. 9366. Springer, 217–234.

L. Libkin. 2004. *Elements Of Finite Model Theory*. Springer.

C. Lutz. 2008. The Complexity of Conjunctive Query Answering in Expressive Description Logics. In *Proc. of the 4th Int. Joint Conf. on Automated Reasoning (IJCAR 2008) (LNAI)*. Springer, 179–193.

C. Lutz, R. Piro, and F. Wolter. 2011. Description Logic TBoxes: Model-Theoretic Characterizations and Rewritability. In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*. IJCAI/AAAI, 983–988.

J. Mora, R. Rosati, and Ó. Corcho. 2014. Kyrie2: Query Rewriting under Extensional Constraints in ELHIO. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014) (Lecture Notes in Computer Science)*, Vol. 8796. Springer, 568–583.

H. Pérez-Urbina, B. Motik, and I. Horrocks. 2009. A Comparison of Query Rewriting Techniques for DL-lite. In *Proc. of the 22nd Inte. Workshop on Description Logics (DL 2009) (CEUR Workshop Proceedings)*, Vol. 477. CEUR-WS.

H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. 2012. Evaluation of Query Rewriting Approaches for OWL 2. In *Proc. of SSWS+HPCSW 2012 (CEUR Workshop Proceedings)*, Vol. 943. CEUR-WS.

A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. 2008. Linking Data to Ontologies. *Journal on Data Semantics* X (2008), 133–173.

R. Raz and A. Wigderson. 1992. Monotone Circuits for Matching Require Linear Depth. *Journal of the ACM* 39, 3 (1992), 736–744.

A. Razborov. 1985. Lower bounds for the monotone complexity of some Boolean functions. *Dokl. Akad. Nauk SSSR* 281, 4 (1985), 798–801.

A. A. Razborov. 1991. Lower Bounds for Deterministic and Nondeterministic Branching Programs. In *Proc. of the 8th Int. Symposium on Fundamentals of Computation Theory (FCT'91) (Lecture Notes in Computer Science)*, Vol. 529. Springer, 47–60.

M. Rodriguez-Muro and D. Calvanese. 2012. High Performance Query Answering over DL-Lite Ontologies. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012)*. AAAI, 308–318.

M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. 2013. Ontology-Based Data Access: Ontop of Databases. In *Proc. of the 12th Int. Semantic Web Conf. (ISWC 2013) (Lecture Notes in Computer Science)*, Vol. 8218. Springer, 558–573.

R. Rosati. 2007. The Limits of Querying Ontologies. In *Proc. of the 11th Int. Conf. on Database Theory (ICDT 2007) (Lecture Notes in Computer Science)*, Vol. 4353. Springer, 164–178.

R. Rosati. 2012. Prexto: Query Rewriting under Extensional Constraints in DL-Lite. In *Proc. of the 9th Extended Semantic Web Conf. (EWSC 2012) (Lecture Notes in Computer Science)*, Vol. 7295. Springer, 360–374.

R. Rosati and A. Almatelli. 2010. Improving Query Answering over DL-Lite Ontologies. In *Proc. of the 12th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2010)*. AAAI Press, 290–300.

J. F. Sequeda, M. Arenas, and D. P. Miranker. 2014. OBDA: Query Rewriting or Materialization? In Practice, Both!. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part I (Lecture Notes in Computer Science)*, Vol. 8796. Springer, 535–551.

I. H. Sudborough. 1978. On the Tape Complexity of Deterministic Context-Free Languages. *Journal of the ACM* 25, 3 (1978), 405–414.

R. Szelepcsényi. 1988. The method of forced enumeration for nondeterministic automata. *Acta Informatica* 26, 3 (1988), 279–284.

M. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*. ACM, 137–146.

H. Venkateswaran. 1991. Properties that Characterize LOGCFL. *J. Comput. System Sci.* 43, 2 (1991), 380–404.

H. Vollmer. 1999. *Introduction to Circuit Complexity: A Uniform Approach*. Springer.

M. Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *Proc. of the 7th Int. Conf. on Very Large Data Bases (VLDB)*. IEEE Computer Society, 82–94.

Y. Zhou, B. Cuenca Grau, Y. Nenov, M. Kaminski, and I. Horrocks. 2015. PAGOdA: Pay-As-You-Go Ontology Query Answering Using a Datalog Reasoner. *Journal of Artificial Intelligence Research (JAIR)* 54 (2015), 309–367.

Received February 2007; revised March 2009; accepted June 2009

Online Appendix to: Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity

MEGHYN BIENVENU, CNRS & University of Montpellier
STANISLAV KIKOT, Birkbeck, University of London
ROMAN KONTCHAKOV, Birkbeck, University of London
VLADIMIR PODOLSKII, Steklov Mathematical Institute, Moscow
MICHAEL ZAKHARYASCHEV, Birkbeck, University of London

A. PROOF OF THEOREM 4.5

THEOREM 4.5 (*i*) *For any OMQ $\boldsymbol{Q}(\boldsymbol{x})$, the formulas $\boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{x})$ and $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ are equivalent, and so $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ is a PE-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.*
(*ii*) *Theorem 4.2 continues to hold for $f^{\triangledown}_{\boldsymbol{Q}}$ replaced by $f^{\blacktriangledown}_{\boldsymbol{Q}}$.*

PROOF. Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ and $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$. We begin by showing that for every tree witness $\mathfrak{t}$ for $\boldsymbol{Q}(\boldsymbol{x})$, we have the following chain of equivalences:

$$\bigwedge_{R(z,z')\in \boldsymbol{q}_{\mathfrak{t}}} (z = z') \ \wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \ \bigwedge_{z\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} \varrho^*(z) \quad \equiv \bigwedge_{z,z'\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} (z = z') \ \wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \ \bigwedge_{z\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} \varrho^*(z)$$
$$\equiv \ \exists z_0 \Big(\bigwedge_{z\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} (z = z_0) \ \wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \varrho^*(z_0)\Big) \quad \equiv \quad \exists z_0 \Big(\bigwedge_{z\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} (z = z_0) \ \wedge \bigvee_{\mathfrak{t} \text{ generated by } \tau} \tau(z_0)\Big),$$

where z_0 is a fresh variable. The first equivalence follows from the transitivity of equality and the fact that every pair of variables z, z' in a tree witness must be linked by a sequence of binary atoms. The following equivalence can be readily verified using first-order semantics. For the final equivalence, we use the fact that if $\mathfrak{t}$ is ϱ-initiated and $\mathcal{T} \models \tau(x) \to \exists y\, \varrho(x, y)$, then $\mathfrak{t}$ is generated by τ, and conversely, if $\mathfrak{t}$ is generated by τ, then there is some ϱ that initiates $\mathfrak{t}$ and is such that $\mathcal{T} \models \tau(x) \to \exists y\, \varrho(x, y)$.

By the above equivalences, the query $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ can be equivalently expressed as follows:

$$\exists \boldsymbol{y} \bigvee_{\substack{\Theta\subseteq\Theta_{\boldsymbol{Q}}\\ \text{independent}}} \Big(\bigwedge_{S(\boldsymbol{z})\in \boldsymbol{q}\setminus\boldsymbol{q}_{\Theta}} S(\boldsymbol{z}) \ \wedge \bigwedge_{\mathfrak{t}\in\Theta} \big(\exists z_0 \,(\bigwedge_{z\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} (z = z_0) \ \wedge \bigvee_{\mathfrak{t} \text{ is generated by } \tau} \tau(z_0))\big)\Big).$$

Finally, we observe that, for every independent $\Theta \subseteq \Theta_{\boldsymbol{Q}}$, the variables that occur in some $\mathfrak{t}_{\mathsf{i}}$, for $\mathfrak{t} \in \Theta$, do not occur in $\mathfrak{t}'_{\mathsf{i}}$ for any other $\mathfrak{t}' \in \Theta$. It follows that if $z \in \mathfrak{t}_{\mathsf{i}}$ and $\mathfrak{t} \in \Theta$, then the only occurrence of z in the disjunct for Θ is in the equality atom $z = z_0$. We can thus drop all such atoms, while preserving equivalence, which gives us precisely the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{x})$. In particular, this means that $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.

To establish the second statement, let Φ be a Boolean formula that computes

$$f^{\blacktriangledown}_{\boldsymbol{Q}} \ = \bigvee_{\substack{\Theta\subseteq\Theta_{\boldsymbol{Q}}\\ \text{independent}}} \Big(\bigwedge_{S(\boldsymbol{z})\in \boldsymbol{q}\setminus\boldsymbol{q}_{\Theta}} p_{S(\boldsymbol{z})} \ \wedge \bigwedge_{\mathfrak{t}\in\Theta} \big(\bigwedge_{R(z,z')\in \boldsymbol{q}_{\mathfrak{t}}} p_{z=z'} \ \wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \ \bigwedge_{z\in \mathfrak{t}_{\mathsf{r}}\cup\mathfrak{t}_{\mathsf{i}}} p_{\varrho^*(z)}\big)\Big),$$

and let $\boldsymbol{q}'(\boldsymbol{x})$ be the FO-formula obtained by replacing each $p_{z=z'}$ in Φ with $z = z'$, each $p_{S(\boldsymbol{z})}$ with $S(\boldsymbol{z})$, each $p_{\varrho^*(z)}$ with $\bigvee_{\mathcal{T}\models\tau(x)\to\exists y\, \varrho(x,y)} \tau(z)$, and prefixing the result

© 2010 ACM. 0004-5411/2010/03-ART39 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

with $\exists \boldsymbol{y}$. Recall that the modified rewriting $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ was obtained by applying this same transformation to the original monotone Boolean formula for $f^{\blacktriangledown}_{\boldsymbol{Q}}$. Since Φ computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$, $\boldsymbol{q}'(\boldsymbol{x})$ and $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ are equivalent FO-formulas. As we have already established that $\boldsymbol{q}'_{\mathsf{tw}}(\boldsymbol{x})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$, the same must be true of $\boldsymbol{q}'(\boldsymbol{x})$. The statement regarding NDL-rewritings can be proved similarly to the proof of Theorem 4.2 (*ii*). □

B. PROOF OF THEOREM 5.9

THEOREM 5.9 (*i*) *Any tree hypergraph H is isomorphic to a subgraph of $\mathcal{H}(\boldsymbol{T}_H)$.*
(*ii*) *Any monotone THGP based on a tree hypergraph H computes a subfunction of the primitive evaluation function $f^{\vartriangle}_{\boldsymbol{T}_H}$.*

PROOF. (i) Fix a tree hypergraph $H = (V, E)$ whose underlying tree $T = (V_T, E_T)$ has vertices $V_T = \{1, \dots, n\}$, for $n > 1$, and 1 is a leaf of T. The *directed* tree obtained from T by fixing 1 as the root and orienting the edges away from 1 is denoted by $T^1 = (V_T, E^1_T)$. By definition, each $e \in E$ induces a convex subtree $T^e = (V_e, E_e)$ of T^1. Since, for each subtree T^e, the OMQ $\boldsymbol{T}_H$ has a tree-witness $\mathfrak{t}^e$ with

$$\mathfrak{t}^e_{\mathsf{r}} = \{\, z_i \mid i \text{ in on the boundary of } e \,\},$$
$$\mathfrak{t}^e_{\mathsf{i}} = \{\, z_i \mid i \text{ is in the interior of } e \,\} \cup \{\, y_{ij} \mid (i,j) \in e \,\},$$

it follows that H is isomorphic to the subgraph of $\mathcal{H}(\boldsymbol{T}_H)$ obtained by removing all superfluous hyperedges and all vertices corresponding to atoms with S_{ij}.

(ii) Suppose that P is bssed on a tree hypergraph H. Given an input $\boldsymbol{\alpha}$ for P, we define an assignment γ for the predicates in $\boldsymbol{T}_H = (\mathcal{T}, \boldsymbol{q})$ by taking each $\gamma(R_{ij})$ and $\gamma(S_{ij})$ to be the value of the label of $(i,j) \in E^1_T$ under $\boldsymbol{\alpha}$ and $\gamma(A_e) = 1$ for all $e \in E$ ($\gamma(R_\zeta) = 0$, for all normalisation predicates R_ζ). We show that for all $\boldsymbol{\alpha}$ we have

$$P(\boldsymbol{\alpha}) = 1 \quad \text{iff} \quad f^{\vartriangle}_{\boldsymbol{T}_H}(\gamma) = 1.$$

Observe that the canonical model $\mathcal{C}_{\mathcal{T},\mathcal{A}(\gamma)}$ contains two labelled nulls, w_e and w'_e, for each $e \in E$, satisfying

$$\mathcal{C}_{\mathcal{T},\mathcal{A}(\gamma)} \models \bigwedge_{(i,j)\in E_e,\ i=r^e} R_{r^e j}(a, w_e) \quad\wedge \bigwedge_{(i,j)\in E_e,\ j\in L_e} S_{ij}(w_e, a) \quad\wedge$$
$$\bigwedge_{(i,j)\in E_e,\ i\neq r^e} R_{ij}(w'_e, w_e) \quad\wedge \bigwedge_{(i,j)\in E_e,\ j\notin L_e} S_{ij}(w_e, w'_e).$$

($\Rightarrow$) Suppose that $P(\boldsymbol{\alpha}) = 1$. Then there exists an independent $E' \subseteq E$ that covers all zeros of $\boldsymbol{\alpha}$. We show $\mathcal{T}, \mathcal{A}(\gamma) \models \boldsymbol{q}$ (that is, $f^{\vartriangle}_{\boldsymbol{T}_H}(\gamma) = 1$). Define a mapping h as follows:

$$h(z_i) = \begin{cases} w'_e, & \text{if } i \text{ is in the interior of } e \in E', \\ a, & \text{otherwise,} \end{cases} \qquad h(y_{ij}) = \begin{cases} w_e, & \text{if } (i,j) \in e \in E', \\ a & \text{otherwise.} \end{cases}$$

Note that h is well-defined: since E' is independent, its hyperedges share no interior, and there can be at most one hyperedge $e \in E'$ containing any given vertex (i,j).

It remains to show that h is a homomorphism from $\boldsymbol{q}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}(\gamma)}$. Consider a pair of atoms $R_{ij}(z_i, y_{ij})$ and $S_{ij}(y_{ij}, z_j)$ in $\boldsymbol{q}$. Then $(i,j) \in E^1_T$. If there is $e \in E'$ with$(i,j) \in e$ then there are four possibilities to consider:

– if neither i nor j is in the interior then, since T_e is a tree and (i,j) is its edge, the only possibility is $e = \{(i,j)\}$, whence $h(z_i) = h(z_j) = a$ and $h(y_{ij}) = w_e$;
– i is on the boundary and j is internal, then $h(z_i) = a$, $h(y_{ij}) = w_e$, and $h(z_j) = w'_e$;
– if j is on the boundary and i is internal, then this case is the mirror image;

– if both i and j are in the interior, then $h(z_i) = h(z_j) = w'_e$ and $h(y_{ij}) = w_e$.

Otherwise, the label of (i, j) must evaluate to 1 under $\boldsymbol{\alpha}$, whence $\mathcal{A}(\boldsymbol{\gamma})$ contains $R_{ij}(a, a)$ and $S_{ij}(a, a)$ and we set $h(z_i) = h(y_{ij}) = h(z_j) = a$. In all cases, h preserves the atoms $R_{ij}(z_i, y_{ij})$ and $S_{ij}(y_{ij}, z_j)$, and so h is indeed a homomorphism.

($\Rightarrow$) Suppose that $f^{\vartriangle}_{\boldsymbol{T}_H}(\boldsymbol{\gamma}) = 1$. Then $\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma}) \models \boldsymbol{q}$, and so there is a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$. We show that there is an independent $E' \subseteq E$ that covers all zeros of $\boldsymbol{\alpha}$. Let E' be the set of all $e \in E$ such that $h^{-1}(w_e) \neq \emptyset$ (that is, w_e is in the image of h). To show that E' is independent, we need the following claim:

Claim. If $h^{-1}(w_e) \neq \emptyset$, then $h(y_{ij}) = w_e$ for all $(i, j) \in e$.

Proof of claim. Let r^e be the root of T^e and L_e its leaves. Pick some variable $z \in h^{-1}(w_e)$ such that there is no $z' \in h^{-1}(w_e)$ higher than z in $\boldsymbol{q}$ (we use the ordering of variables induced by the tree T^1). Observe that z cannot be of the form z_j, because then $\boldsymbol{q}$ would contain some atom $R_{j\ell}(z_j, y_{j\ell})$ or $S_{\ell j}(y_{\ell j}, z_j)$, but w_e has no outgoing $R_{j\ell}$ or $S^-_{\ell j}$ arcs in $\mathcal{C}_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$. It follows that z is of the form $y_{j\ell}$, for some j, ℓ. By considering the available arcs leaving w_e again, we conclude that $(j, \ell) \in e$. We next show that $j = r^e$. Suppose that this is not the case. Then, there must be $(p, j) \in e$ with $(p, j) \in T^1$. A simple examination of the axioms in $\mathcal{T}$ shows that the only way for h to satisfy the atom $R_{j\ell}(z_j, y_{j\ell})$ is to map z_j to w'_e. It follows that to satisfy the atom $S_{pj}(y_{pj}, z_j)$, we must put $h(y_{pj}) = w_e$ contrary to the assumption that $z = y_{j\ell}$ was a highest vertex in $h^{-1}(w_e)$. Thus, $j = r^e$. Now, using a simple inductive argument on the distance from z_{r^e}, and considering the possible ways of mapping the atoms of $\boldsymbol{q}$, we can show that $h(y_{ij}) = w_e$ for every $(i, j) \in e$. (*end proof of claim*)

Suppose that there are two distinct hyperedges $e, e' \in E'$ that have a non-empty intersection: $(i, j) \in e \cap e'$. We know that either y_{ij} or y_{ji} occurs in $\boldsymbol{q}$, and we can assume the former without loss of generality. By the claim, we obtain $h(y_{ij}) = w_e = w_{e'}$, a contradiction. Therefore, E' is independent. We now show that it covers all zeros. Let (i, j) be such that its label evaluates to 0 under $\boldsymbol{\alpha}$, and assume again without loss of generality that y_{ij} occurs in $\boldsymbol{q}$. Then $\mathcal{A}(\boldsymbol{\gamma})$ does not contain $R_{ij}(a, a)$, so the only way h can satisfy the atom $R_{ij}(z_i, y_{ij})$ is by mapping y_{ij} to some w_e with $(i, j) \in e$. It follows that there is an $e \in E'$ such that $(i, j) \in e$, so all zeros of $\boldsymbol{\alpha}$ are covered by E'. We have thus shown that E' is an independent subset of E that covers all zeros of $\boldsymbol{\alpha}$, and hence, $P(\boldsymbol{\alpha}) = 1$. $\square$

C. PROOF OF PROPOSITION 5.10

PROPOSITION 5.10. (*i*) *For any tree hypergraph H of degree $\leq d$, there is a monotone THGP of size $O(|H|)$ that computes f_H and such that its hypergraph is of degree $\leq \max(2, d)$.*

(*ii*) *For every generalised THGP P over n variables, there is a THGP P' computing the same function and such that $|P'| \leq n \cdot |P|$.*

PROOF. (*i*) Consider a hypergraph $H = (V, E)$ based on a tree $T = (V_T, E_T)$ with $V = E_T$. We label each $v \in V$ with a variable p_v and, for each $e \in E$, we choose some $v_e \in \bigcup e$, add fresh vertices a_e and b_e with edges $\{v_e, a_e\}$ and $\{a_e, b_e\}$ to T as well as a new hyperedge $e' = [v_e, b_e]$ to E. We label the segment $[v_e, a_e]$ with 1 and the segment $[a_e, b_e]$ with p_e. We also extend e to include the segment $[v_e, a_e]$. We claim that the resulting THGP P computes f_H. Indeed, for any input $\boldsymbol{\alpha}$ with $\boldsymbol{\alpha}(p_e) = 0$, we have to include the edge e' into the cover, and so cannot include the edge e itself. Thus, $P(\boldsymbol{\alpha}) = 1$ iff there is an independent set E of hyperedges with $\boldsymbol{\alpha}(p_e) = 1$, for all $e \in E$, covering all zeros of the variables p_v. It follows that P computes f_H.

(*ii*) Let P be a generalised THGP based on a hypergraph $H = (V, E)$ with the underlying tree $T = (V_T, E_T)$ such that $V = E_T$. To construct P', we split every vertex $v \in V$ (which is an edge of T) labelled with $\bigwedge_{i=1}^{k} \boldsymbol{l}_i$ into k new edges $v_1, \dots, v_k$ and label v_i with $\boldsymbol{l}_i$, for $1 \leq i \leq k$; each hyperedge containing v will now contain all the v_i. It is easy to see that $P(\boldsymbol{\alpha}) = P'(\boldsymbol{\alpha})$, for any valuation $\boldsymbol{\alpha}$. Since $k \leq n$, we have $|P'| \leq n \cdot |P|$. It should be clear that the degree of P' and the number of leaves in it are the same as in P. $\square$

D. PROOF OF THEOREM 5.13

THEOREM 5.13. *For every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with a fundamental set $\Omega_{\boldsymbol{Q}}$ and with $\boldsymbol{q}$ of treewidth t, the generalised monotone THGP $P_{\boldsymbol{Q}}$ computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$ and is of size polynomial in $|\boldsymbol{q}|$ and $|\Omega_{\boldsymbol{Q}}|^t$.*

PROOF. By [Flum and Grohe 2006, Lemma 11.9], we can assume that the tree T in the tree decomposition of $\boldsymbol{q}$ has at most N, $N \leq |\boldsymbol{q}|$, nodes. Recall that $M = |\Omega_{\boldsymbol{Q}}|^t$ is the number of bag types. We claim that $P_{\boldsymbol{Q}}$

– contains at most $(2M + 1)N$ vertices and at most $N(M + M^2)$ hyperedges;
– and has labels with at most $3|\boldsymbol{q}|$ conjuncts.

The vertices of the hypergraph of $P_{\boldsymbol{Q}}$ correspond to the edges of T', and there can be at most $N \cdot (2M + 1)$ of them, because there can be no more than N edges in T, and each is replaced by a sequence of $2M + 1$ new edges. The hyperedges are of two types: E_i^k (where $1 \leq i \leq N$ and $1 \leq k \leq M$) and $E_{ij}^{k\ell}$ (where (i, j) correspond to an edge in T and $1 \leq k, \ell \leq M$). It follows that the total number of hyperedges cannot exceed $N(M + M^2)$. Finally, a simple examination of the labelling function shows that there can be at most $3|\boldsymbol{q}|$ conjuncts in each label. Indeed, given i, j and k, each atom $S(\boldsymbol{z})$ with $\boldsymbol{z} \subseteq \lambda(N_i)$ generates either 1 or 3 propositional variables in the label of $\{u_{ij}^k, v_{ij}^k\}$, and $|\boldsymbol{q}|$ is the upper bound for the number of such atoms.

To complete the proof, we show that $P_{\boldsymbol{Q}}$ computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$: for any valuation $\boldsymbol{\alpha}$,

$$f^{\blacktriangledown}_{\boldsymbol{Q}}(\boldsymbol{\alpha}) = 1 \qquad \text{iff} \qquad P_{\boldsymbol{Q}}(\boldsymbol{\alpha}) = 1.$$

($\Rightarrow$) Let $\boldsymbol{\alpha}$ be such that $f^{\blacktriangledown}_{\boldsymbol{Q}}(\boldsymbol{\alpha}) = 1$. Then we can find an independent $\Theta \subseteq \Theta_{\boldsymbol{Q}}$ such that $\boldsymbol{\alpha}$ satisfies the corresponding disjunct of $f^{\blacktriangledown}_{\boldsymbol{Q}}$:

$$\bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta} p_{S(\boldsymbol{z})} \quad \wedge \bigwedge_{\mathfrak{t} \in \Theta} \Big(\bigwedge_{R(z,z') \in \boldsymbol{q}_{\mathfrak{t}}} p_{z=z'} \;\; \wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \;\; \bigwedge_{z \in \mathfrak{t}_{\mathsf{r}} \cup \mathfrak{t}_{\mathsf{i}}} p_{\varrho^*(z)} \Big). \tag{14}$$

For every $\mathfrak{t} \in \Theta$, let $\varrho_{\mathfrak{t}}$ be a role that makes the disjunction hold. Since $\mathfrak{t}$ is $\varrho_{\mathfrak{t}}$-initiated, we can choose a homomorphism $h_{\mathfrak{t}} \colon \boldsymbol{q}_{\mathfrak{t}} \to \mathcal{C}_{\mathcal{T}}^{\varrho_{\mathfrak{t}}(a)}$ such that, for every $z \in \mathfrak{t}_{\mathsf{i}}$, $h_{\mathfrak{t}}(z)$ is of the form $a\varrho_{\mathfrak{t}}w$, for some w .

With each node N in the tree decomposition (T, λ) we associate the *type $\boldsymbol{w}$ of N* by taking, for all $z \in \lambda(N)$:

$$\boldsymbol{w}[\nu_N(z)] = \begin{cases} w, & \text{if } z \in \mathfrak{t}_{\mathsf{i}} \text{ and } h_{\mathfrak{t}}(z) = aw, \text{ for some } \mathfrak{t} \in \Theta, \\ \varepsilon, & \text{otherwise.} \end{cases}$$

Observe that $\boldsymbol{w}$ is well-defined since the independence of Θ guarantees that every variable in $\boldsymbol{q}$ can appear in $\mathfrak{t}_{\mathsf{i}}$ for at most one $\mathfrak{t} \in \Theta$. We show that $\boldsymbol{w}$ is compatible with N. Consider a unary atom $A(z) \in \boldsymbol{q}$ such that $z \in \lambda(N)$ and $\boldsymbol{w}[\nu_N(z)] \neq \varepsilon$. Then there must be $\mathfrak{t} \in \Theta$ such that $z \in \mathfrak{t}_{\mathsf{i}}$, in which case $h_{\mathfrak{t}}(z) = a\boldsymbol{w}[\nu_N(z)]$. Let ϱ be the final symbol in $h_{\mathfrak{t}}(z)$. Since $h_{\mathfrak{t}} \colon \boldsymbol{q}_{\mathfrak{t}} \to \mathcal{C}_{\mathcal{T}}^{\exists y \varrho_{\mathfrak{t}}(a,y)}$ is a homomorphism, we have $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$.

Consider now a binary atom $P(z,z') \in \boldsymbol{q}$ such that $z, z' \in \lambda(N)$ and either $\boldsymbol{w}[\nu_N(z)] \neq \varepsilon$ or $\boldsymbol{w}[\nu_N(z')] \neq \varepsilon$. We assume w.l.o.g. that the former is true (the other case is handled analogously). By definition, there is $\mathfrak{t} \in \Theta$ such that $z \in \mathfrak{t}_\mathsf{i}$ and $h_\mathfrak{t}(z) = a\boldsymbol{w}[\nu_N(z)]$. Since $z \in \mathfrak{t}_\mathsf{i}$ and $P(z,z') \in \boldsymbol{q}$, by the definition of tree witnesses, $z' \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}$. Since $h_\mathfrak{t} \colon \boldsymbol{q}_\mathfrak{t} \to \mathcal{C}_\mathcal{T}^{\exists y \varrho_\mathfrak{t}(a,y)}$ is a homomorphism, one of the following holds:

– $\boldsymbol{w}[\nu_N(z')] = \boldsymbol{w}[\nu_N(z)]$ and $\mathcal{T} \models P(x,x)$;
– $\boldsymbol{w}[\nu_N(z)] = \boldsymbol{w}[\nu_N(z')] \cdot \varrho$ for some ϱ with $\mathcal{T} \models \varrho(y,x) \to P(x,y)$;
– $\boldsymbol{w}[\nu_N(z')] = \boldsymbol{w}[\nu_N(z)] \cdot \varrho$ for some ϱ with $\mathcal{T} \models \varrho(x,y) \to P(x,y)$.

This establishes the second part of the compatibility condition. Next, we show that the pairs associated with different nodes in T are compatible. Consider a pair of nodes N and N' and their types $\boldsymbol{w}$ and $\boldsymbol{w}'$. It is clear that, by construction, $\boldsymbol{w}[\nu_N(z)] = \boldsymbol{w}'[\nu_{N'}(z)]$, for all $z \in \lambda(N) \cap \lambda(N')$.

Let $\boldsymbol{w}_1, \dots, \boldsymbol{w}_M$ be all the bag types. Consider now the tree hypergraph $P_{\boldsymbol{Q}}$, and let E' be the set consisting of the following hyperedges:

– for every N_i in T, the hyperedge $E_i^k = [N_i, u^k_{ij_1}, \dots, u^k_{ij_n}]$, where k is such that $\boldsymbol{w}_k$ is the type of N_i, and $N_{j_1}, \dots, N_{j_n}$ are the neighbours of N_i;
– for every pair of adjacent nodes N_i, N_j in T, the hyperedge $E_{ij}^{k\ell} = [v^k_{ij}, v^\ell_{ji}]$, where k and ℓ are such that $\boldsymbol{w}_k$ and $\boldsymbol{w}_\ell$ are the types of N_i and N_j, respectively.

Note that all these hyperedges are present in the hypergraph of $P_{\boldsymbol{Q}}$ because we have shown that the type of each node N_i is compatible with it and that the pairs of types of N_i and N_j are compatible with the pair (N_i, N_j). It is easy to see that E' is independent, since whenever we include E_i^k or $E_{ij}^{k\ell}$, we do not include any $E_i^{k'}$ or $E_{ij}^{k'\ell}$ for $k' \neq k$. It remains to show that every vertex of the hypergraph of $P_{\boldsymbol{Q}}$ that is not covered by E' evaluates to 1 under $\boldsymbol{\alpha}$. Observe first that most of the vertices are covered by E'. Specifically:

– $\{N_i, u^1_{ij}\}$ is covered by E_i^k;
– $\{v^k_{ij}, u^{k+1}_{ij}\}$ is covered either by E_i^n (if $n \leq k+1$) or by $E_{ij}^{n\ell}$ (if $n > k+1$);
– $\{v^M_{ij}, v^M_{ji}\}$ is covered by $E_{ij}^{k\ell}$;
– $\{u^k_{ij}, v^k_{ij}\}$ is covered by E_i^n if $k < n$, and by $E_{ij}^{n\ell}$ if $n > k$.

Thus, the only type of vertex not covered by E' is of the form $\{u^k_{ij}, v^k_{ij}\}$, where $\boldsymbol{w}_k$ is the type of N_i. In this case, by definition, $\{u^k_{ij}, v^k_{ij}\}$ is labelled by the following variables:

– $p_{S(\boldsymbol{z})}$, if $S(\boldsymbol{z}) \in \boldsymbol{q}$, $\boldsymbol{z} \subseteq \lambda(N_i)$ and $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varepsilon$, for all $z \in \boldsymbol{z}$;
– $p_{\varrho^*(z)}$, if $A(z) \in \boldsymbol{q}$, $z \in \lambda(N_i)$ and $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varrho w$;
– $p_{\varrho^*(z)}$, $p_{\varrho^*(z')}$ and $p_{z=z'}$, if $S(z,z') \in \boldsymbol{q}$ (possibly with $z = z'$), $z, z' \in \lambda(N_i)$ and either $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varrho w$ or $\boldsymbol{w}_k[\nu_{N_i}(z')] = \varrho w$.

First suppose that $p_{S(\boldsymbol{z})}$ appears in the label of $\{u^k_{ij}, v^k_{ij}\}$. Then $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varepsilon$, for all $z \in \boldsymbol{z}$, and hence there is no variable in $S(\boldsymbol{z})$ that belongs to any $\mathfrak{t}_\mathsf{i}$ for $\mathfrak{t} \in \Theta$. It follows that $S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta$, and since (14) is satisfied, the variable $p_{S(\boldsymbol{z})}$ evaluates to 1 under $\boldsymbol{\alpha}$. Next suppose that one of $p_{\varrho^*(z)}$, $p_{\varrho^*(z')}$ and $p_{z=z'}$ is part of the label. We focus on the case where these variables came from a binary atom (third item above), but the proof is similar for the case of a unary atom (second item above). We know that there is some atom $S(z,z') \in \boldsymbol{q}$ with $z, z' \in \lambda(N_i)$ and either $\boldsymbol{w}_k[\nu_{N_i}(z)] = \varrho w$ or $\boldsymbol{w}_k[\nu_{N_i}] = \varrho w$. It follows that there is a tree witness $\mathfrak{t} \in \Theta$ such that $z, z' \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}$. This means that the atom $p_{z=z'}$ is a conjunct of (14), and so it is satisfied under $\boldsymbol{\alpha}$. Also, either $\boldsymbol{w}_k[\nu_{N_i}(z)] = h_\mathfrak{t}(z)$ or $\boldsymbol{w}_k[\nu_{N_i}(z')] = h_\mathfrak{t}(z')$ is of the form ϱw, and, since all non-empty words in the image of $h_\mathfrak{t}$ begin by $\varrho_\mathfrak{t}$, we obtain $\varrho = \varrho_\mathfrak{t}$. Since $\varrho_\mathfrak{t}$ was chosen so that $\bigwedge_{z \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}} p_{\varrho^*(z)}$ is satisfied under $\boldsymbol{\alpha}$, both $p_{\varrho^*(z)}$ and $p_{\varrho^*(z')}$ evaluate to 1 under $\boldsymbol{\alpha}$. Therefore, E' is independent and covers all zeros under $\boldsymbol{\alpha}$, which means that $P_{\boldsymbol{Q}}(\boldsymbol{\alpha}) = 1$.

($\Leftarrow$) Suppose $P_{\boldsymbol{Q}}(\boldsymbol{\alpha}) = 1$, i.e., there is an independent subset E' of the hyperedges in $P_{\boldsymbol{Q}}$ that covers all vertices evaluated to 0 under $\boldsymbol{\alpha}$. It is clear from the construction of $P_{\boldsymbol{Q}}$ that E' contains exactly one hyperedge of the form E_i^k for every node N_i in T, and so we can associate with every node N_i the unique index $\mu(N_i) = k$. We also know that E' contains exactly one hyperedge of the form $E_{ij}^{k\ell}$ for every edge $\{N_i, N_j\}$ in T. Moreover, if we have hyperedges E_i^k and $E_{ij}^{k'\ell}$ (respectively, E_j^ℓ and $E_{ij}^{k\ell'}$), then $k = k'$ (respectively, $\ell = \ell'$). It also follows from the definition of $P_{\boldsymbol{Q}}$ that every $\boldsymbol{w}_{\mu(N_i)}$ is compatible with N_i, and pairs $(\boldsymbol{w}_{\mu(N_j)}, \boldsymbol{w}_{\mu(N_j)})$ are compatible for adjacent nodes N_i, N_j. Using the compatibility properties and the connectedness condition of tree decompositions, we can conclude that the pairs assigned to *any two nodes* N_i and 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 with every variable z in $\boldsymbol{q}$.

Since all zeros are covered by E', we know that for every node N_i, the following variables are assigned to 1 by $\boldsymbol{\alpha}$:

– $p_{S(\boldsymbol{z})}$, if $S(\boldsymbol{z}) \in \boldsymbol{q}$, $\boldsymbol{z} \subseteq \lambda(N_i)$ and $w_z = \varepsilon$, for $z \in \boldsymbol{z}$;
– $p_{\varrho^*(z)}$, if $A(z) \in \boldsymbol{q}$, $z \in \lambda(N_i)$, and $w_z = \varrho w$; $(\star)$
– $p_{\varrho^*(z)}$, $p_{\varrho(z')}$ and $p_{z=z'}$, if $S(z, z') \in \boldsymbol{q}$ (possibly with $z = z'$), $z, z' \in \lambda(N_i)$ and either $w_z = \varrho w$ or $w_{z'} = \varrho w$.

Now let $\equiv$ be the smallest equivalence relation on the atoms of $\boldsymbol{q}$ that satisfies the following condition, for every variable z in $\boldsymbol{q}$,

$$\text{if } w_z \neq \varepsilon \text{ and } z \text{ occurs in both } S_1(\boldsymbol{z}_1) \text{ and } S_2(\boldsymbol{z}_2), \text{ then } S_1(\boldsymbol{z}_1) \equiv S_2(\boldsymbol{z}_2).$$

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

Claim. For every $\boldsymbol{q}_i$ that contains a variable z with $w_z \neq \varepsilon$:

(1) there is a role ϱ_i such that every $w_z \neq \varepsilon$ (with z a variable in $\boldsymbol{q}_i$) begins by ϱ_i;
(2) there is a homomorphism $h_i \colon \boldsymbol{q}_i \to \mathcal{C}_\mathcal{T}^{\exists y \varrho_i(a,y)}$ such that $h_i(z) = a w_z$ for every variable z in $\boldsymbol{q}_i$;
(3) there is a tree witness $\mathfrak{t}^i$ for $\boldsymbol{Q}$ that is ϱ_i-initiated and such that $\boldsymbol{q}_i = \boldsymbol{q}_{\mathfrak{t}^i}$

Proof of claim. By the definition of $\boldsymbol{q}_i$, there exists a sequence $Q_0, \dots, Q_n$ of subsets of $\boldsymbol{q}$ such that $Q_0 = \{S_0(\boldsymbol{z}_0)\} \subseteq \boldsymbol{q}_i$ contains a variable z_0 with $w_{z_0} \neq \varepsilon$, $Q_n = \boldsymbol{q}_i$, and for every $0 \leq \ell < n$, $Q_{\ell+1}$ is obtained from Q_ℓ by adding an atom that contains a variable z that appears in Q_ℓ and is such that $w_z \neq \varepsilon$. By construction, every atom in $\boldsymbol{q}_i$ contains a variable z with $w_z \neq \varepsilon$. Let ϱ_i be the first letter of the word w_{z_0}, and for every $0 \leq \ell \leq n$, let h_ℓ be the function that maps every variable z in Q_ℓ to $a w_z$.

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 z in Q_ℓ with $w_z \neq \varepsilon$ begins by ϱ_i, and that h_ℓ is a homomorphism of Q_ℓ into the canonical model $\mathcal{C}_\mathcal{T}^{\exists y \varrho_i(a,y)}$ that satisfies $h_\ell(y) = a w_z$. We let $S(\boldsymbol{z})$ be the unique atom in $Q_{\ell+1} \setminus Q_\ell$. Then $S(\boldsymbol{z})$ contains a variable z that appears in Q_ℓ and is such that $w_z \neq \varepsilon$. If $S(\boldsymbol{z}) = B(z)$ or $S(\boldsymbol{z}) = R(z, z)$, then Statement 1 for w_z is immediate. For Statement 2, we let N be a node in T such that $z \in \lambda(N)$. Since $\boldsymbol{w}_N$ is compatible with N, it follows that, if $S(\boldsymbol{z}) = B(z)$, then w_z ends by a role ϱ with $\mathcal{T} \models \exists y\, \varrho(y,x) \to B(x)$, and, if $S(\boldsymbol{z}) = R(z,z)$, then $\mathcal{T} \models R(x,x)$, which proves Statement 2. Next, consider the case when $S(\boldsymbol{z})$ contains two variables, that is, it is of the form $R(z,z')$ or $R(z,z')$. We give the argument for the former (the the latter is analogous). Let N be a node in T such that $\{z,z'\} \subseteq \lambda(N)$. Since $\boldsymbol{w}_N$ is compatible with N, either

– $w_{z'} = w_z\varrho$ with $\mathcal{T} \models \varrho(x,y) \to R(x,y)$, or
– $w_z = w_{z'}\varrho$ with $\mathcal{T} \models \varrho(y,x) \to R(x,y)$.

Since w_z begins with ϱ_i, the same holds for $w_{z'}$ unless $w_{z'} = \varepsilon$, which proves Statement 1. It is also clear from the way we defined $h_{\ell+1}$ that it is homomorphism from $Q_{\ell+1}$ to $\mathcal{C}_{\mathcal{T}}^{\exists y \varrho_i(a,y)}$, so Statement 2 holds.

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

Let Θ consist of all the tree witnesses $\mathfrak{t}^i$ obtained in the claim. As the $\boldsymbol{q}_i$ are disjoint, the set $\{\boldsymbol{q}_{\mathfrak{t}^i} \mid \mathfrak{t}^i \in \Theta\}$ is independent. We show that $\boldsymbol{\alpha}$ satisfies the disjunct of $f^{\blacktriangledown}_{\boldsymbol{Q}}$ that corresponds to Θ; cf. (14). First, consider some $S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta$. Then, for every variable z in $S(\boldsymbol{z})$, we have $w_z = \varepsilon$. Let N be a node in T such that $\boldsymbol{z} \subseteq \lambda(N)$. Then $w_z = \varepsilon$, for all $z \in \boldsymbol{z}$. It follows from $(\star)$ that $\boldsymbol{\alpha}(p_{S(\boldsymbol{z})}) = 1$. Next, consider a variable $p_{z=z'}$ such that there is an atom $S(\boldsymbol{z}) \in \boldsymbol{q}_{\mathfrak{t}^i}$ with $\boldsymbol{z} = \{z,z'\}$. Since $S(\boldsymbol{z}) \in \boldsymbol{q}_i$, either $w_z \neq \varepsilon$ or $w_{z'} \neq \varepsilon$. It follows from $(\star)$ that $\boldsymbol{\alpha}(p_{z=z'}) = 1$. Finally, let us consider a tree witness $\mathfrak{t}^i \in \Theta$, and let ϱ_i be the role from the claim. We show that $p_{\varrho_i^*(z)} = 1$ for every variable z in $\mathfrak{t}^i$, which will imply that the final disjunction is satisfied by $\boldsymbol{\alpha}$. Consider a variable z in $\mathfrak{t}^i$. By the construction of the query $\boldsymbol{q}_i = \boldsymbol{q}_{\mathfrak{t}^i}$, it contains a binary atom $S(\boldsymbol{z})$ with $z, z' \in \boldsymbol{z}$ and either $w_z \neq \varepsilon$ or $w_{z'} \neq \varepsilon$. By the definition of tree decompositions, there is a node N in T with $z, z' \in \lambda(N)$. Then, by Statement 1 of the claim, either $w_z = \varrho_i w$ or $w_{z'} = \varrho_i w$. Now we can apply $(\star)$ to obtain $p_{\varrho_i^*(z)} = 1$, as required. □

E. PROOFS OF THEOREMS 6.3 AND 6.4

THEOREM 6.3 (GENERAL CASE). $\mathsf{NL/poly} = \mathsf{THGP}(\ell)$ and $\mathsf{mNL/poly} = \mathsf{mTHGP}(\ell)$, for any $\ell \geq 2$.

PROOF. Suppose a polynomial-size THGP P based on a tree hypergraph H with at most ℓ leaves computes a Boolean function f. We show how to construct a polynomial-size NBP that computes the same f. By Theorem 6.4 (to be proved below), f_H can be computed by a polynomial-size NBP B. We replace the vertex variables p_v in labels of B by the corresponding vertex labels in P and fix all the edge variables p_e to 1; see the proof of Proposition 4.7 (*i*). Clearly, the resulting NBP B' is as required.

The converse direction is given in Section 6.3. □

THEOREM 6.4. *Fix $\ell \geq 2$. For any tree hypergraph H based on a tree with at most ℓ leaves, the function f_H can be computed by an NBP of size polynomial in $|H|$.*

PROOF. Let $H = (V,E)$ be a tree hypergraph and $T = (V_T, E_T)$ its underlying tree ($V = E_T$ and each $e \in E$ induces a convex subtree T_e of T). Pick some vertex $r \in V_T$ and fix it as a root of T. We call an independent subset $F \subseteq E$ of hyperedges *flat* if every simple path in T with endpoint r intersects at most one of T_e, for $e \in F$. Note that every flat subset can contain at most ℓ hyperedges, so the number of flat subsets is bounded by a polynomial in $|H|$. We denote by $[F]$ the union of subtrees T_e, for all $e \in F$ ($[F]$ is a possibly disconnected subgraph of T). Flat subsets can be partially ordered by taking

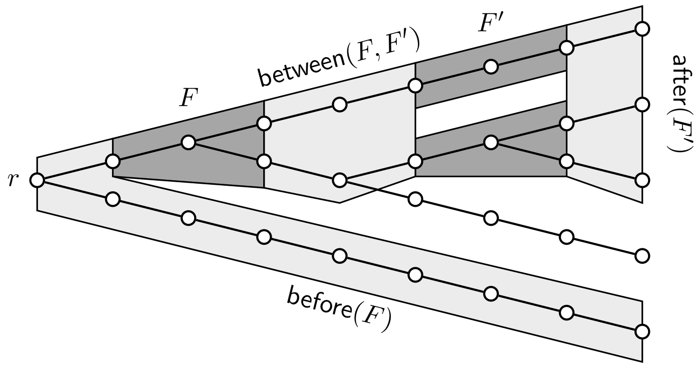


Fig. 21. Parts of the underlying tree in the proof of Theorem 6.4: $F \prec F'$ for F with one hyperedge and F' with two hyperedges.

$F \preceq F'$ if every simple path between the root r and a vertex of $[F']$ intersects $[F]$. As usual, $F \prec F'$ if $F \preceq F'$ but $F \neq F'$.

The required NBP P is based on the graph $G = (V_P, E_P)$ with

$$V_P = \big\{u_F, \bar{u}_F \mid F \text{ is flat}\big\} \cup \big\{s, t\big\},$$
$$E_P = \big\{(s, u_F), (\bar{u}_F, t), (u_F, \bar{u}_F) \mid F \text{ is flat}\big\} \cup \big\{(\bar{u}_F, u_{F'}) \mid F, F' \text{ are flat and } F \prec F'\big\}.$$

To define labels, we introduce some notation first for sets of edges of T (which are sets of vertices of H). For a flat F, let before(F) be the edges of T that lie outside $[F]$ and are accessible from the root r via paths not passing through $[F]$; we denote by after(F) the edges of T outside $[F]$ that are accessible from r only via paths passing through $[F]$. Finally, for flat F and F' with $F \prec F'$, we denote by between(F, F') the set of edges in T 'between' $[F]$ and $[F']$, that is those edges of T outside $[F]$ and $[F']$ that are accessible from $[F]$ via paths not passing through $[F']$ but are not accessible from the root r via a path not passing through $[F]$; see Fig. 21. Now we are ready to define the labelling for edges of G:

– each $(u_F, \bar{u}_F)$ is labelled with the conjunction of p_e for $e \in F$;
– each (s, u_F) is labelled with the conjunction of p_v for $v \in$ before(F);
– each $(\bar{u}_F, u_{F'})$ is labelled with the conjunction of p_v for $v \in$ between(F, F');
– each $(\bar{u}_F, t)$ is labelled with the conjunction of p_v for $v \in$ after(F).

We claim that under any valuation $\boldsymbol{\alpha}$ of p_e and p_v, there is a path from s to t in G all of whose labels evaluate to 1 under $\boldsymbol{\alpha}$ iff $f_H(\boldsymbol{\alpha}) = 1$, that is, iff there is an independent (not necessarily flat) subset $E' \subseteq E$ such that $\boldsymbol{\alpha}(p_e) = 1$ for all $e \in E'$ and $\boldsymbol{\alpha}(p_v) = 1$ for all $v \in V \setminus V_{E'}$. Indeed, any such E' splits into flat 'layers' $F^1, F^2, \dots F^m$ that form a path

$$s \to u_{F^1} \to \bar{u}_{F^1} \to u_{F^2} \to \cdots \to \bar{u}_{F^m} \to t$$

in G and whose edge labels evaluate to 1: take F^1 to be the set of all hyperedges from E' that are accessible from r via paths which do not cross (that is come in and go out) any hyperedge of E'; take F^2 to be the set of all edges from $E' \setminus F^1$ that are accessible from r via paths that do not cross any hyperedge of $E' \setminus F^1$, and so on. Conversely, any path leading from s to t gives us a covering E', which is the union of all flat sets that occur in the subscripts of vertices on this path. □

F. PROOF OF THEOREMS 6.5 AND 6.7

LEMMA F.1. *Any semi-unbounded fan-in circuit $\boldsymbol{C}$ of* AND*-depth d is equivalent to a semi-unbounded fan-in circuit $\boldsymbol{C}'$ of size $2^d|\boldsymbol{C}|$ and* AND*-depth d such that, for each $n \leq d$, $\boldsymbol{C}'$ satisfies*

$$\bigcup_{g \in S_n} \mathsf{left}(g) \quad \cap \quad \bigcup_{g \in S_n} \mathsf{right}(g) \quad = \quad \emptyset.$$

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

Consider a subcircuit $\bigcup_{g \in S_n} \mathsf{left}(g)$ of $\boldsymbol{C}$, take its copy $\boldsymbol{C}''$ and feed the outputs of $\boldsymbol{C}''$ as left inputs to AND-gates in S_n. This at most doubles the size of the circuit and ensures the property for S_n. Now apply the induction hypothesis to both $\boldsymbol{C}''$ and $\bigcup_{g \in S_n} \mathsf{right}(g)$ (which do not intersect). The size of the resulting circuit will increase at most by the factor of 2^{n-1} and the property for S_i for $i < n$ will be ensured. □

Let g_i be a gate in $\boldsymbol{C}$. We denote by T_i the subtree of T with the root v_i and, given an input $\boldsymbol{\alpha}$, we say that T_i can be *covered* under $\boldsymbol{\alpha}$ if the hypergraph with the underlying tree T_i has an independent subset of hyperedges that are wholly in T_i and cover all zeros under $\boldsymbol{\alpha}$.

LEMMA F.2. *For a given input $\boldsymbol{\alpha}$ and any i, the gate g_i outputs 1 iff T_i can be covered.*

PROOF. We prove the claim by induction on i. If g_i is an input gate and outputs 1, then the label of the edge $\{v_i, u_i\}$ is evaluated into 1 under $\boldsymbol{\alpha}$, and the remainder of T_i can be covered by a set of $[w_j, u_j]$-hyperedges. Conversely, if an input gate g_i outputs 0, then no hyperedge can cover $\{v_i, u_i\}$.

If $g_i = g_j \land g_k$ is an AND-gate and outputs 1, then both its inputs output 1. We cover both subtrees corresponding to the inputs (by induction hypothesis) and add to the covering the hyperedge $[v_i, v_j, v_k]$, which covers T_i. Conversely, any covering of zeros in T_i must include the hyperedge $[v_i, v_j, v_k]$, and so the subtrees T_j and T_k must be covered. Thus, by the induction hypothesis, g_j and g_k should output 1, and so does g_i.

If $g_i = g_{j_1} \lor \dots \lor g_{j_k}$ is an OR-gate and outputs 1, then one of its inputs, say, g_j, is 1. By the induction hypothesis, we cover its subtree and add the hyperedge $[v_i, v_j]$, which forms a covering of T_i. Conversely, since $\{v_i, u_i\}$ is labelled by 0, any covering of T_i must include a hyperedge of the form $[v_i, v_j]$ for some $j \in \{j_1, \dots, j_k\}$. Thus T_j must also be covered. By the induction hypothesis, g_j outputs 1 and so does g_i. □

THEOREM 6.7.$\mathsf{NC}^1 = \mathsf{THGP}^d$ *and* $\mathsf{mNC}^1 = \mathsf{mTHGP}^d$*, for any $d \geq 3$.*

PROOF. To prove $\mathsf{NC}^1 \subseteq \mathsf{THGP}^3$, consider a polynomial-size formula $\boldsymbol{C}$, which we represent as a tree of gates $g_1, \dots, g_m$ enumerated so that $j < i$ whenever g_j is an input of g_i. We assume that $\boldsymbol{C}$ has negated variables in place of NOT-gates. We now construct the tree, T, underlying the THGP P we are after: T contains triples of vertices u_i, v_i, w_i partially ordered in the same way as the g_i in $\boldsymbol{C}$. We then remove vertex w_m and make v_m the root of T. The THGP P is based on the hypergraph whose vertices are the edges of T and whose hyperedges comprise the following:

– $[u_i, u_j]$, for each $i < m$, where $j < i$ and g_j is the input of g_i;
– $[v_i, v_j, v_k]$, for each $g_i = g_j \land g_k$;
– $[v_i, v_j, w_k], [v_i, w_j, v_j]$, for each $g_i = g_j \lor g_k$.

Finally, if an input gate g_i is a literal $\boldsymbol{l}$, we label the edge $\{u_i, v_i\}$ with $\boldsymbol{l}$; we label all other edges with 0. It is not hard to check that P is of degree 3, has size polynomial in $|\boldsymbol{C}|$, and computes the same function as $\boldsymbol{C}$.

The inclusion $\mathsf{NC}^1 \supseteq \mathsf{THGP}^d$ follows from the proof of $\mathsf{LOGCFL/poly} \subseteq \mathsf{THGP}$ in Theorem 6.5. Indeed, if the degree of the THGP is at most d, then the disjunction in (13) has at most $d+1$ disjuncts, and so the constructed circuit has depth $O(\log s)$. $\square$

G. PROOFS FOR SECTION 7

THEOREM 7.6. *For any fixed $\ell \geq 2$, all tree-shaped OMQs with at most ℓ leaves have polynomial-size NDL-rewritings.*

PROOF. Fix $\ell \geq 2$ and let $\boldsymbol{Q}$ be a tree-shaped OMQ with at most ℓ leaves. By Theorem 5.7, $\mathcal{H}(\boldsymbol{Q})$ is a tree hypergraph whose underlying tree has at most ℓ leaves. By Theorem 6.4, $f^{\triangledown}_{\boldsymbol{Q}}$ is computable by a polynomial-size monotone NBP, and so, since $\mathsf{mNL/poly} \subseteq \mathsf{mP/poly}$, $f^{\triangledown}_{\boldsymbol{Q}}$ can be computed by a polynomial-size monotone Boolean circuit. It remains to apply Theorem 4.2 (*ii*). $\square$

THEOREM 7.7. *There is an OMQ with ontologies of depth 2 and linear CQs any PE-rewriting of which is of superpolynomial size $n^{\Omega(\log n)}$.*

PROOF. We consider the function $f = \textsc{Reachability}$. Since $f \in \mathsf{mNL/poly}$, by Theorem 6.3, there is a polynomial-size monotone HGP that is based on a hypergraph H with underlying tree with 2 leaves and computes f. Consider now the OMQ $\boldsymbol{T}_H$ for H defined in Section 5.3, which has an ontology of depth 2. By Theorem 5.9 (*ii*), f is a subfunction of $f^{\vartriangle}_{\boldsymbol{T}_H}$. By Theorem 4.6 (*i*), no PE-rewriting of the OMQ $\boldsymbol{T}_H$ can be shorter than $n^{\Omega(\log n)}$. $\square$

THEOREM 7.9. *For any fixed $t > 0$, all OMQs with the PFSP and CQs of treewidth at most t have polynomial-size NDL-rewritings.*

PROOF. Fix a $t > 0$ and a class of OMQs with PFSP. Take an OMQ $\boldsymbol{Q}$ of treewidth at most t from the class. By Theorem 5.13, there is a polynomial-size monotone THGP that computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$. Since $\mathsf{mTHGP} \subseteq \mathsf{mLOGCFL/poly} \subseteq \mathsf{mP/poly}$ (Theorem 6.5), $f^{\blacktriangledown}_{\boldsymbol{Q}}$ can be computed by a polynomial-size monotone Boolean circuit. It remains to apply Theorem 4.5 (*ii*). $\square$

THEOREM 7.12. *For any fixed $t > 0$, all OMQs with ontologies of depth 1 and CQs of treewidth at most t have polynomial-size PE-rewritings.*

The main argument underlying Theorem 7.12 was given in the body of the paper. To complete the proof, we give the following two lemmas, which are the modified versions of Theorems 4.2 and 5.13 mentioned in the body.

LEMMA G.1. *Theorem 4.2 continues to hold if $f^{\triangledown}_{\boldsymbol{Q}}$ is replaced by $f^{\blacktriangledown\prime}_{\boldsymbol{Q}}$.*

PROOF. The proof proceeds similarly to the proof of Theorem 4.5. The key step in the proof is showing that the FO-formula

$$\exists \boldsymbol{y} \bigvee_{\substack{\Theta \subseteq \Theta_{\boldsymbol{Q}} \\ \text{independent}}} \Big(\bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_{\Theta}} S(\boldsymbol{z}) \ \wedge \bigwedge_{\mathfrak{t} \in \Theta} \big(\bigwedge_{R(z,z') \in \boldsymbol{q}_{\mathfrak{t}}} z = z' \ \wedge \bigwedge_{z \in \mathfrak{t}_{\mathsf{r}} \cup \mathfrak{t}_{\mathsf{i}}} \ \bigvee_{\mathfrak{t} \text{ generated by } \tau} \tau(z)\big)\Big)$$

obtained from $f^{\blacktriangledown\prime}_{\boldsymbol{Q}}$ by replacing variables $p_{S(\boldsymbol{z})}$, $p_{z=z'}$, and $p_{\exists y P_{\mathfrak{t}}(z,y)}$ by $S(\boldsymbol{z})$, $z = z'$, and $\bigvee_{\mathfrak{t} \text{ generated by } \tau} \tau(z)$ respectively is equivalent the tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$. $\square$

LEMMA G.2. *In the setting of Section 7.5, for the modified hypergraph program $P'_{\boldsymbol{Q}}$ we still have $f_{P'_{\boldsymbol{Q}}}(\boldsymbol{v}) = f^{\blacktriangledown\prime}_{\boldsymbol{Q}}(\boldsymbol{v})$.*

PROOF. The proof closely follows that of Theorem 5.13. For the first direction of the proof, the only notable difference is that instead of selecting a role $\varrho_{\mathfrak{t}}$ that satisfies the disjunct corresponding to the tree witness $\mathfrak{t}$, we must take the special role $P_\mathfrak{t}$. For the second direction, we use the assumption that $\mathcal{T}$ is of depth 1 to show that every query $\boldsymbol{q}_j$ (constructed according to the equivalence relation) has a single variable v_j such that $w_{v_j} \neq \varepsilon$. This allows us to prove a stronger version of the claim in which $\boldsymbol{q}_j = \boldsymbol{q}_{\mathfrak{t}^j}$, with $\mathfrak{t}^j$ the unique tree witness with $\mathfrak{t}^j_\mathsf{i} = \{v_j\}$, and the selected role ϱ_j is equal to the special predicate $P_{\mathfrak{t}^j}$ associated with $\mathfrak{t}^j$. □

THEOREM 7.13. *All tree-shaped OMQs with ontologies of depth 1 have polynomial-size Π_4-rewritings.*

PROOF. Take an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ with $\mathcal{T}$ of depth 1 and a tree-shaped $\boldsymbol{q}$. By Theorems 5.3 and 5.7, $\mathcal{H}(\boldsymbol{Q})$ is a polynomial-size tree hypergraph of degree at most 2. By Proposition 5.10 (i), $f^{\triangledown}_{\boldsymbol{Q}}$ can be computed by a polynomial-size THGP P of degree at most 2. By Theorem 6.8, there is a polynomial-size monotone Π_3-circuit computing $f^{\triangledown}_{\boldsymbol{Q}}$. By a simple unravelling argument, it follows that there is polynomial-size monotone Boolean formula computing $f^{\triangledown}_{\boldsymbol{Q}}$. It remains to apply Theorem 4.2 (i) and conclude that there is a polynomial-size positive existential Π_4-rewriting for $\boldsymbol{Q}$. □

H. PROOF OF LOGCFL MEMBERSHIP IN THEOREM 8.3

We say that an iteration of the **while** loop is *successful* if the procedure BLQuery does not return false; in particular, if none of the **check** operations returns false. The following properties can be easily seen to hold by examination of BLQuery and straightforward induction:

For every tuple $(z \mapsto (a, n), z') \in \mathsf{frontier}$, z' is a child of z in T. (15)

For every tuple $(z \mapsto (a, n), z') \in \mathsf{frontier}$, we have $n \leq |\mathsf{stack}|$. (16)

All tuples $(z \mapsto (a, n), z') \in \mathsf{frontier}$ with $n > 0$ share the same a. (17)

Once $(z \mapsto (a, n), z')$ is added to frontier, no tuple of the form $(z \mapsto (a', n'), z')$ can ever be added to frontier. (18)

In every successful iteration, either at least one tuple is removed from frontier or frontier is unchanged but one ϱ is popped from the stack. (19)

If $(z \mapsto (a, n), z')$ is removed from frontier in a successful iteration, then a tuple of the form $(z' \mapsto (a', n'), z'')$ is added to frontier, for every child z'' of z' in T. (20)

PROPOSITION H.1. *Every execution of* BLQuery *terminates.*

PROOF. A simple examination of BLQuery shows that the only possible source of non-termination is the **while** loop, which continues as long as frontier is non-empty. By (15) and (18), the total number of tuples that may appear in frontier at any point cannot exceed the number of edges in T, which is itself bounded by $|\boldsymbol{q}|$. By (18) and (19), every tuple is added at most once and is eventually removed from frontier. Thus, either the algorithm will exit the **while** loop by returning false (if one of the **check** operations fails), or it will eventually exit the loop after reaching an empty frontier. □

PROPOSITION H.2. *There exists an execution of* BLQuery *that returns* true *on input* $((\mathcal{T}, \boldsymbol{q}), \mathcal{A}, \boldsymbol{a})$ *if and only if* $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$.

PROOF. ($\Leftarrow$) Suppose that $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$. Then there exists a homomorphism $h \colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ such that $h(\boldsymbol{x}) = \boldsymbol{a}$. Without loss of generality we may choose h so that the image of h consists of elements aw with $|w| \leq 2|\mathcal{T}| + |\boldsymbol{q}|$ [Artale et al. 2009]. We use h to specify an execution of BLQuery$((\mathcal{T}, \boldsymbol{q}), \mathcal{A}, \boldsymbol{a})$ that returns true. First, we fix an arbitrary variable z_0 as root, and then, we choose the element $h(z_0) = a_0w_0$. Since h defines a homomorphism of $\boldsymbol{q}(\boldsymbol{a})$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, the call canMapTail($z_0$, a_0, top(stack)) returns true. We initialise stack to w_0 and frontier to $\{(z_0 \mapsto (a_0, |\mathsf{stack}|), v_i \mid v_i \text{ is a child of } v_0\}$. Next, we enter the **while** loop. Our aim is to make the non-deterministic choices to satisfy the following invariant:

$$\text{If} \quad (z \mapsto (a,m), z') \in \mathsf{frontier}, \qquad \text{then} \quad h(z) = a\,\mathsf{stack}_{\leq m}. \tag{21}$$

Recall that $\mathsf{stack}_{\leq m}$ denotes the word obtained by concatenating the first m symbols of stack. Observe that before the **while** loop, property (21) is satisfied. At the start of each iteration of the while loop, we proceed as follows.

[CASE 1.] If frontier contains $(z \mapsto (a,0), z')$ such that $h(z') \in \mathsf{ind}(\mathcal{A})$, then we choose Option 1. We remove the tuple from frontier and choose the individual $a' = h(z')$ for the guess. As $a = h(z)$ (by (21)) and h is a homomorphism, we have $(a, a') \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for all $P(z,z') \in \boldsymbol{q}$, and the call canMapTail(z', a', ε) returns true. We thus add $(z' \mapsto (a',0), z'')$ to frontier for every child z'' of z' in T. These additions to frontier clearly preserve the invariant.

[CASE 2.] If Case 1 does not apply and frontier contains $(z \mapsto (a, |\mathsf{stack}|), z')$ such that $h(z') = h(z)$, then we choose Option 4 and remove the tuple from frontier. Since h is homomorphism, we have $\mathcal{T} \models P(x,x)$, for all $P(z,z') \in \boldsymbol{q}$, and canMapTail($z'$, a, top(stack)) returns true. Then, for every child z'' of z' in T, we add $(z' \mapsto (a, |\mathsf{stack}|), z'')$ to frontier. Observe that since $h(z) = h(z')$ and (21) holds for z, property (21) also holds for the newly added tuples.

[CASE 3.] If neither Case 1 nor Case 2 applies and frontier contains $(z \mapsto (a, |\mathsf{stack}|), z')$ such that $h(z') = h(z)\varrho$, then we choose Option 2 and remove the tuple from frontier. Note that in this case, $|\mathsf{stack}| < 2|\mathcal{T}| + |\boldsymbol{q}|$ since (i) by (21), $h(z) = aw$, for $w = \mathsf{stack}_{\leq|\mathsf{stack}|}$, and (ii) by the choice of homomorphism h, we have $|w\varrho| \leq 2|\mathcal{T}| + |\boldsymbol{q}|$. So, we continue and choose ϱ for the guess. By (21), since h is a homomorphism and $h(z') = h(z)\varrho$, the call isGenerated(ϱ, a, top(stack)) returns true, $\mathcal{T} \models \varrho(x,y) \to P(x,y)$, for all $P(z,z') \in \boldsymbol{q}$ and the call canMapTail(z', a, top(stack)) returns true. So, we push ϱ onto stack and add $(z' \mapsto (a, |\mathsf{stack}|), z'')$ to frontier for every child z'' of z' in T. As stack contains the word component of $h(z')$, invariant (21) holds for the newly added tuples.

[CASE 4.] If none of Case 1, Case 2 or Case 3 is applicable, then we choose Option 3 and remove all elements in $\mathsf{deepest} = \{(z \mapsto (a,n), z') \in \mathsf{frontier} \mid n = |\mathsf{stack}|\}$ from frontier. Since neither Case 1 nor Case 3 applies, $|\mathsf{stack}| > 0$. So, we pop the top symbol ϱ from stack. Suppose first that $\mathsf{deepest} \neq \emptyset$. By (17), all tuples in deepest share the same individual a. By (21), every tuple $(z \mapsto (a,n), z') \in \mathsf{deepest}$ is such that $h(z) = aw\varrho$, where $w = \mathsf{stack}_{\leq|\mathsf{stack}|}$. Moreover, since Case 3 is not applicable, for every such tuple $(z \mapsto (a,n), z')$, we have $h(z') = aw$. Using the fact that h is a homomorphism, one can show that $\mathcal{T} \models \varrho(x,y) \to P(x,y)$, for all $P(z',z) \in \boldsymbol{q}$, and canMapTail($z'$, a, top(stack)) returns true. So, we add to frontier all tuples $(z' \mapsto (a, |\mathsf{stack}|), z'')$, a child z'' of z' in T. Note that invariant (21) is satisfied by all the new tuples. Moreover, since we only removed the last symbol in stack, all the remaining tuples in frontier continue to satisfy (21). Finally, if deepest was empty, then we do nothing but the tuples in frontier continue to satisfy (21).

It is easily verified that so long as frontier is non-empty, one of these four cases applies. Since we have shown how to make the non-deterministic choices in the **while** loop without returning false, by Proposition H.1, the procedure eventually leaves the **while** loop and returns true.

($\Rightarrow$) Consider an execution of $\mathsf{BLQuery}((\mathcal{T}, \boldsymbol{q}), \mathcal{A}, \boldsymbol{a})$ that returns true. It follows that the **while** loop is successfully exited after reaching an empty $\mathsf{frontier}$. Let L be the total number of iterations of the **while** loop. We inductively define a sequence $h_0, h_1, \dots, h_L$ of partial functions from the variables of $\boldsymbol{q}$ to $\Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ by considering the guesses made during the different iterations of the **while** loop. The domain of h_i will be denoted by $\mathsf{dom}(h_i)$. We will ensure that the following properties hold for every $0 \leq i < L$:

$$\text{If } i > 0, \text{ then } \mathsf{dom}(h_{i-1}) \subseteq \mathsf{dom}(h_i), \text{ and } h_i(z) = h_{i-1}(z), \text{ for } z \in \mathsf{dom}(h_{i-1}). \quad (22)$$

$$\text{If } (z \mapsto (a, n), z') \in \mathsf{frontier} \text{ at the end of iteration } i, \text{ then} \quad (23)$$

$$h_i(z) = aw, \text{ where } w = \mathsf{stack}_{\leq n}, \quad (23a)$$

$$\text{and neither } z' \text{ nor any of its descendants belongs to } \mathsf{dom}(h_i). \quad (23b)$$

$$h_i \text{ is a homomorphism } \boldsymbol{q}_i \to \mathcal{C}_{\mathcal{T},\mathcal{A}}, \text{ where } \boldsymbol{q}_i \text{ is the restriction of } \boldsymbol{q} \text{ to } \mathsf{dom}(h_i). \quad (24)$$

We begin by setting $h_0(z_0) = a_0 w_0$, where w_0 is the word in stack (and leaving h_0 undefined for all other variables). Property (22) is vacuously satisfied. Property (23) holds because of the initial values of $\mathsf{frontier}$ and stack because only $z_0 \in \mathsf{dom}(h_0)$, and z_0 cannot be its own child (hence, it cannot appear in the last component of a tuple in $\mathsf{frontier}$). To see why (24) is satisfied, first suppose that $w_0 = \varepsilon$ and so $a_0 w_0 \in \mathsf{ind}(\mathcal{A})$. Then, the call `canMapTail(`$z_0$, a_0, $\mathsf{top}(\mathsf{stack})$`)` returns true. It follows that

if z_0 is the jth answer variable then $a_0 = a_j$;

$a_0 \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for each $A(z_0) \in \boldsymbol{q}$, and $(a_0, a_0) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, for each $P(z_0, z_0) \in \boldsymbol{q}$;

and hence, h_0 defines a homomorphism of $\boldsymbol{q}_0$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Otherwise, w_0 is non-empty and $w_0 = w_0' \varrho$. It follows that

z_0 is not an answer variable of $\boldsymbol{q}$;

$\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$, for each $A(z_0) \in \boldsymbol{q}$, and $\mathcal{T} \models P(x, x)$, for each $P(z_0, z_0) \in \boldsymbol{q}$;

and hence h_0 homomorphically maps all atoms of $\boldsymbol{q}_0$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Thus, the initial partial function h_0 satisfies (22)–(24).

Next we show how to inductively define h_i from h_{i-1} while preserving (22)–(24). The variables that belong to $\mathsf{dom}(h_i) \setminus \mathsf{dom}(h_{i-1})$ are precisely those variables that appear in the last position of tuples removed from $\mathsf{frontier}$ 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 the four options was selected. In what follows, we will use stack^i to denote the contents of stack at the end of iteration i.

OPTION 1: we remove a tuple $(z \mapsto (a, 0), z')$ and guess $a' \in \mathsf{ind}(\mathcal{A})$. So, we set $h_i(z') = a'$ and $h_i(v) = h_{i-1}(v)$ for all $v \in \mathsf{dom}(h_{i-1})$ (all other variables remain undefined). Property (22) is by definition. For property (23), consider a tuple $\tau = (v \mapsto (c, m), v')$ that belongs to $\mathsf{frontier}$ at the end of iteration i. Suppose first τ was added to $\mathsf{frontier}$ during iteration i, in which case $\tau = (z' \mapsto (a', 0), z'')$ for some child z'' of z'. Property (23a) is satisfied because $\mathsf{stack}^i_{\leq 0} = \varepsilon$. Since h_{i-1} satisfies (23), z'' (a descendant of z') is not in $\mathsf{dom}(h_{i-1})$, which satisfies (23b). The remaining possibility is that τ was already in $\mathsf{frontier}$ at the beginning of iteration i. Since h_{i-1} satisfies (23), we have $h_{i-1}(v) = cw$ for $w = \mathsf{stack}^{i-1}_{\leq n}$ and neither v' nor any of its descendants belongs to $\mathsf{dom}(h_{i-1})$. Since $\mathsf{stack}^i = \mathsf{stack}^{i-1}$ and $h_i(v) = h_{i-1}(v)$, property (23a) holds for τ. Moreover, as τ was not removed from $\mathsf{frontier}$ during iteration i, we have $\tau \neq (z \mapsto (a, 0), z')$, and so, by (18), $v' \neq z'$. Thus, neither v' nor any of its descendants is in $\mathsf{dom}(h_i)$.

For property (24), we first note that since h_i agrees with h_{i-1} on $\mathsf{dom}(h_i)$ and h_{i-1} satisfies (24), it is only necessary to consider the atoms in $\boldsymbol{q}_i$ that do not belong to $\boldsymbol{q}_{i-1}$. There are three kinds of such atoms:

– if $A(z') \in \boldsymbol{q}_i$, then, since `canMapTail(`$z'$, a', ε`)` returns true, $h_i(z') = a' \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
– if $P(z',z') \in \boldsymbol{q}_i$, then, again, since `canMapTail(`$z'$, a', ε`)` returns true, we have $(h_i(z'), h_i(z')) = (a',a') \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
– if $P(z',v) \in \boldsymbol{q}_i$ with $v \neq z'$, then $v \in \mathsf{dom}(h_i)$, so v must coincide with z, the parent of z' (rather than being one of the children of z'); the **check** operation in the algorithm then guarantees $(h_i(z'), h_i(v)) = (a',a) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

Thus, (24) holds for h_i.

OPTION 2: a tuple $(z \mapsto (a,n), z')$ was removed from frontier, $n = |\mathsf{stack}|$ and a role ϱ was guessed. We set $h_i(z') = h_{i-1}(z)\varrho$. By (23), $h_{i-1}(z)$ is defined. Moreover, the call `isGenerated(`ϱ, a, top(stack)`)` ensures that $h_{i-1}(z)\varrho \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$. We also set $h_i(v) = h_{i-1}(v)$ for all $v \in \mathsf{dom}(h_{i-1})$ and leave the remaining variables undefined. Property (22) is immediate from the definition of h_i, and (23b) can be shown exactly as for Option 1. To show (23a), consider a tuple $\tau = (v \mapsto (c,m), v')$ that belongs to frontier at the end of iteration i. Suppose first that τ was added to frontier during iteration i, in which case $\tau = (z' \mapsto (a, n+1), z'')$ for some child z'' of z'. Since h_{i-1} satisfies (23), $h_{i-1}(z) = a\,\mathsf{stack}^{i-1}_{\leq n}$. Property (23a) follows then from $h_i(z') = h_{i-1}(z)\varrho$ and $\mathsf{stack}^i = \mathsf{stack}^{i-1}\,\varrho$. The other possibility is that τ was present in frontier at the beginning of iteration i. Since h_{i-1} satisfies (23), we have $h_{i-1}(v) = a\,\mathsf{stack}^{i-1}_{\leq m}$. Property (23a) continues to hold for τ because $\mathsf{stack}^i = \mathsf{stack}^{i-1}\,\varrho$ and $m \leq |\mathsf{stack}^{i-1}|$ and $h_i(v) = h_{i-1}(v)$.

We now turn to property (24). As explained in the proof for Option 1, it is sufficient to consider the atoms in $\boldsymbol{q}_i \setminus \boldsymbol{q}_{i-1}$, which can be of three types:

– if $A(z') \in \boldsymbol{q}_i$, then, since `canMapTail(`$z'$, a, ϱ`)` returns true, we have $\mathcal{T} \models \exists y\ \varrho(y,x) \to A(x)$, hence $h_i(z') = h_{i-1}(z)\varrho \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
– if $P(z',z') \in \boldsymbol{q}_i$, then, again, since `canMapTail(`$z'$, a, ϱ`)` returns true, we have $\mathcal{T} \models P(x,x)$, hence $(h_i(z'), h_i(z')) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.
– if $P(z',v) \in \boldsymbol{q}_i$ with $v \neq z'$ then $v = z$ (see Option 1); so, $\mathcal{T} \models \varrho(x,y) \to P(y,x)$, whence $(h_i(z'), h_i(v)) = (h_{i-1}(z)\varrho, h_{i-1}(z)) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

Therefore, h_i is a homomorphism from $\boldsymbol{q}_i$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, which is required by (24).

OPTION 3: tuples in $\mathsf{deepest} = \{(z \mapsto (a,n), z') \in \mathsf{frontier} \mid n = |\mathsf{stack}|\}$ are removed from frontier, and role ϱ is popped from stack. By (17), all tuples in deepest share the same individual a. Let $V = \{z' \mid (z \mapsto (a,n), z') \in \mathsf{deepest}\}$. For every $v \in V$, we set $h_i(v) = a\,\mathsf{stack}^i$; we also set $h_i(v) = h_{i-1}(v)$ for all $v \in \mathsf{dom}(h_{i-1})$ and leave the remaining variables undefined. Property (22) is again immediate, and the argument for (23b) is the same as in Option 1. For property (23a), take any tuple $\tau = (v \mapsto (c,m), v')$ in frontier at the end of iteration i. If the tuple was added to frontier during this iteration, then $v \in V$, $a = c$, $m = |\mathsf{stack}^i|$, and $h_i(v) = a\,\mathsf{stack}^i$, whence (23a). The other possibility is that τ was present in frontier at the beginning of iteration i. Then $h_{i-1}(v) = c\,\mathsf{stack}^{i-1}_{\leq m}$ and $m < |\mathsf{stack}^{i-1}|$. Since stack^i is obtained from stack^{i-1} by popping one role, we have $m \leq |\mathsf{stack}^i|$, and so (23a) holds for τ.

For property (24), the argument is similar to Options 1 and 2 and involves considering the different types of atoms that may appear in $\boldsymbol{q}_i \setminus \boldsymbol{q}_{i-1}$:

– if $A(z') \in \boldsymbol{q}_i$ with $z' \in V$ then, since `canMapTail(`z', a, top(stack)`)` returns true, we have $h_i(z') \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$ (see Options 1 and 2);
– if $P(z',z') \in \boldsymbol{q}_i$ with $z' \in V$ then, since `canMapTail(`z', a, top(stack)`)` returns true, we have $(h_i(z'), h_i(z')) = (a,a) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
– if $P(z',v) \in \boldsymbol{q}_i$ with $v \neq z'$ and $z' \in V$, then v is the parent of z (see Option 1) and, since $\mathcal{T} \models \varrho(y,x) \to P(x,y)$, we obtain $(h_i(z'), h_i(v)) = (a\,\mathsf{stack}^i, a\,\mathsf{stack}^i\,\varrho) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

Thus, (24) holds for h_i.

OPTION 4: a tuple $(z \mapsto (a,n), z')$ was removed from $\mathsf{frontier}$ with $n = |\mathsf{stack}|$. We set $h_i(z') = h_i(z)$, $h_i(v) = h_{i-1}(v)$ for every $v \in \mathsf{dom}(h_{i-1})$, and leave all other variables unmapped. Again, it is easy to see that properties (22) and (23b) are satisfied by h_i. For property (23a), let $\tau = (v \mapsto (c,m), v')$ be a tuple in $\mathsf{frontier}$ at the end of iteration i. If the tuple was added during iteration i, then $v = z'$, $a = c$, and $m = n$. Since $(z \mapsto (a,n), z')$ was present at the end of iteration $i{-}1$ and $\mathsf{stack}^i = \mathsf{stack}^{i-1}$, we have $h_i(z') = a\,\mathsf{stack}^{i-1}_{\leq n}$, hence $h_i(z) = c\,\mathsf{stack}^{i}_{\leq m}$. As $h_i(z') = h_i(z)$, we have $h_i(z') = a\,\mathsf{stack}^{i}_{\leq m}$, so τ satisfies (23a). If τ was already present at the beginning of iteration i, then we can use the fact that $\mathsf{stack}^i = \mathsf{stack}^{i-1}$ and all tuples in $\mathsf{frontier}$ satisfy (23a).

To show (24), we consider the three types of atoms that may appear in $\boldsymbol{q}_i \setminus \boldsymbol{q}_{i-1}$:

– if $A(z') \in \boldsymbol{q}_i$ then, since `canMapTail(`z', a, $\mathsf{top}(\mathsf{stack})$`)` returns true, then $\mathcal{T} \models \exists y\, \varrho(y,x) \to A(x)$, where $\varrho = \mathsf{top}(\mathsf{stack})$, and so $h_i(z') \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
– if $P(z',z') \in \boldsymbol{q}_i$ then, since `canMapTail(`z', a, $\mathsf{top}(\mathsf{stack})$`)` returns true, then $\mathcal{T} \models P(x,x)$, and so $(h_i(z'), h(z')) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
– if $P(z',v) \in \boldsymbol{q}_i$ with $v \neq z'$, then $v = z$ (see Option 1), and so, since $\mathcal{T} \models P(x,x)$, we have $(h_i(z'), h_i(z)) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

We claim that the final partial function h_L is a homomorphism of $\boldsymbol{q}$ to $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Since h_L is a homomorphism of $\boldsymbol{q}_L$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$, it suffices to show that $\boldsymbol{q} = \boldsymbol{q}_L$, or equivalently, that all variables of $\boldsymbol{q}$ are in $\mathsf{dom}(h_L)$. This follows from the tree-shapedness of $\boldsymbol{q}$ (which in particular means that $\boldsymbol{q}$ is connected), invariants (15), and (20), and the fact that $\mathsf{dom}(h_{i+1}) = \mathsf{dom}(h_i) \cup \{z' \mid (z \mapsto (a,n), z')$ is removed from $\mathsf{frontier}$ during iteration $i\}$. $\square$

PROPOSITION H.3. $\mathsf{BLQuery}$ *can be implemented by an NAuxPDA.*

PROOF. It suffices to show that $\mathsf{BLQuery}$ runs in non-deterministic logarithmic space and polynomial time (the size of stack does not have to be bounded).

First, we non-deterministically fix a root variable z_0, but do not actually need to store the induced directed tree T in memory. Instead, it suffices to decide, given two variables z and z', whether z' is a child of z in T, which clearly belongs to NL.

Next, we need only logarithmic space to store the individual a_0. The initial word $w_0 = \varrho_1 \dots \varrho_{n_0}$ is guessed symbol by symbol and pushed onto stack. We note that both subroutines, `isGenerated` and `canMapTail`, can be made to run in non-deterministic logarithmic space. Then, since the children of a node in T can be identified in NL, we can decide in non-deterministic logarithmic space whether a tuple $(z_0 \mapsto (a_0, |\mathsf{stack}|, z_i)$ should be included in $\mathsf{frontier}$. Moreover, since the input query $\boldsymbol{q}$ is a tree-shaped query with a bounded number of leaves, we know that only constantly many tuples can be added to $\mathsf{frontier}$ by each such operation. Moreover, it is clear that every tuple can be stored using in logarithmic space. More generally, by (15) and (18), 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 in logarithmically space.

Next observe that every iteration of the **while** loop involves a polynomial number of the following elementary operations such as

– remove a tuple from $\mathsf{frontier}$, or add a tuple to $\mathsf{frontier}$;
– pop a role from stack, or push a role onto stack;
– guess a single individual constant or symbol;
– identify the children of a given variable;
– test whether $\mathcal{T} \models \alpha$, for some inclusion α involving symbols from $\mathcal{T}$;

– make a call to one of the subroutines `isGenerated` or `canMapTail`.

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 proof, observe that, by (19), each iteration of the while loop involves removing a tuple from frontier or popping a role from stack. By (15) every tuple in frontier corresponds to an edge in T, and, by (18), we create at most one tuple per edge. Thus, there can be at most $|\boldsymbol{q}|$ iterations involving the removal of a tuple. The total number of roles added to stack is bounded by at most $\leq 2|\mathcal{T}| + |\boldsymbol{q}|$ roles in the initial stack, plus the at most $|q|$ roles added in later iterations, yielding at most $2|\mathcal{T}| + 2|\boldsymbol{q}|$ iterations involving only the popping of a role. Thus, the total number of iterations of the while loop cannot exceed can $2|\mathcal{T}| + 3|\boldsymbol{q}|$. □

I. PROOF OF LOGCFL-HARDNESS IN THEOREM 8.3

PROPOSITION I.1. *The query $\boldsymbol{q}'$ and KB $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ can be computed from $\boldsymbol{C}$ by logspace transducers.*

PROOF. Consider a circuit $\boldsymbol{C}$ in normal form with $2d + 1$ layers of gates, where d is logarithmic in number of its inputs n. We show that $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ and $\boldsymbol{q}'$ can be constructed using $O(\log(n))$ worktape memory.

To produce the query $\boldsymbol{q}'$, 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 n, and so we need only logarithmic space to store the index of the current variable.

The ontology $\mathcal{T}_{\boldsymbol{\alpha}}$ 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 them. 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 $\boldsymbol{\alpha}$. Finally, $\mathcal{A}$ consists of a single constant atom. □

PROPOSITION I.2. *$\boldsymbol{C}$ accepts $\boldsymbol{\alpha}$ iff $\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A} \models \boldsymbol{q}'(a)$.*

PROOF. Denote by e the natural homomorphism from $\boldsymbol{q}'$ to $\boldsymbol{q}$, and by e' the natural homomorphism from $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ to $D(\boldsymbol{\alpha})$. Since $\boldsymbol{C}$ accepts input $\boldsymbol{\alpha}$ iff there is a homomorphism h from $\boldsymbol{q}$ to $D(\boldsymbol{\alpha})$ [Gottlob et al. 2001], it suffices to show that there exists a homomorphism f from $\boldsymbol{q}'$ to $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ iff there is a homomorphism h from $\boldsymbol{q}$ to $D(\boldsymbol{\alpha})$:

(a)
$$\begin{tikzcd}
\boldsymbol{q}' \arrow[r, dashed, "f"] \arrow[d, "e"'] & \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \arrow[d, "e'"] \\
\boldsymbol{q} \arrow[r, "h"'] \arrow[ur, dashed, "h'"] & D(\boldsymbol{\alpha})
\end{tikzcd}$$

(b)
$$\begin{tikzcd}
\boldsymbol{q}' \arrow[r, "f"] \arrow[d, "e"'] & \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \arrow[d, "e'"] \\
\boldsymbol{q} \arrow[r, dashed, "h"'] \arrow[ur, dashed, "f'"] & D(\boldsymbol{\alpha})
\end{tikzcd}$$

($\Rightarrow$) Suppose that h is a homomorphism from $\boldsymbol{q}$ to $D(\boldsymbol{\alpha})$. We define a homomorphism $h' \colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ inductively moving from the root n_1 of $\boldsymbol{q}$ to its leaves. For the basis of induction, we set $h'(n_1) = a$; note that $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \models G_m(a)$. For the inductive step, suppose that n_j is a child of n_i, $h'(n_i)$ is defined, $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \models G_{i'}(h(n_i))$ 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 $\mathcal{T}_{\boldsymbol{\alpha}}$ that $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \models G_{j'}(h'(n_j))$, which enables us to continue the induction. It should be clear that h' is indeed a homomorphism from $\boldsymbol{q}$ into $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$. The desired homomorphism $f\colon \boldsymbol{q}' \to \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ can be obtained as the composition of e and h', as illustrated in diagram (a).

($\Leftarrow$) Suppose that f is a homomorphism from $\boldsymbol{q}'$ to $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$. We prove, by induction on $|j-i|$, that for all its variables y_i, y_j,

$$e(y_i) = e(y_j) \quad \text{implies} \quad f(y_i) = f(y_j). \tag{25}$$

The base case ($|j-i| = 0$) is trivial. For the inductive step, we may assume without loss of generality that $i < j$ and between y_i and y_j there is no intermediate variable y_k with $e(y_i) = e(y_k) = e(y_j)$ (otherwise, we can simply use the induction hypothesis together with the transitivity of equality). It follows that $e(y_{i+1}) = e(y_{j-1})$, and the atom between y_{j-1} and y_j is oriented from y_{j-1} towards y_j, while the atom between y_i and y_{i+1} goes from y_{i+1} to y_i. Indeed, this holds if the node $n = e(y_i) = e(y_j)$ is an OR-node since there are exactly two variables in $\boldsymbol{q}'$ which are mapped to n, and they bound the subtree in $\boldsymbol{q}$ 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 $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ 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, we define $f'\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ by setting $f'(x) = f(y)$, where y is such that $e(y) = x$. By (25), f' is well-defined, and because f is a homomorphism, the same holds for f'. To obtain the desired homomorphism $h\colon \boldsymbol{q} \to D(\boldsymbol{\alpha})$, it suffices to consider the composition of f' and e'; see diagram (b). $\square$