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

MEGHYN BIENVENU, CNRS & University of Montpellier, France
STANISLAV KIKOT, Birkbeck, University of London, UK
ROMAN KONTCHAKOV, Birkbeck, University of London, UK
VLADIMIR V. PODOLSKII, Steklov Mathematical Institute of the Russian Academy of Sciences, Moscow, Russia and National Research University Higher School of Economics, Russia
MICHAEL ZAKHARYASCHEV, Birkbeck, University of London, UK

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.

CCS Concepts: • **Computing methodologies** → **Description logics**; • **Information systems** → **Query languages**; • **Theory of computation** → **Description logics**; **Circuit complexity**;

Additional Key Words and Phrases: ontology-based data access, query rewriting, ontology-mediated query, succinctness, computational complexity.

ACM Reference Format:
Meghyn Bienvenu, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. 2018. Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity. *J. ACM* 1, 1, Article 1 (January 2018), 70 pages. https://doi.org/10.1145/3191832

1 INTRODUCTION

1.1 Ontology-Based Data Access

Ontology-based data access (OBDA) via query rewriting was proposed by Poggi et al. [72] 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 ontology that serves as a high-level conceptual view of the data, gives a

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, meghyn@lirmm.fr; S. Kikot, R. Kontchakov, M. Zakharyaschev, Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London WC1E 7HX, UK, {kikot, roman, michael}@dcs.bbk.ac.uk; V. V. Podolskii, Steklov Mathematical Institute of the Russian Academy of Sciences, 8 Gubkina str., 119991, Moscow, Russia and National Research University Higher School of Economics, 20 Myasnitskaya str., 101000, Moscow, Russia, podolskii@mi.ras.ru.

0004-5411/2018/1-ART1 $15.00
https://doi.org/10.1145/3191832

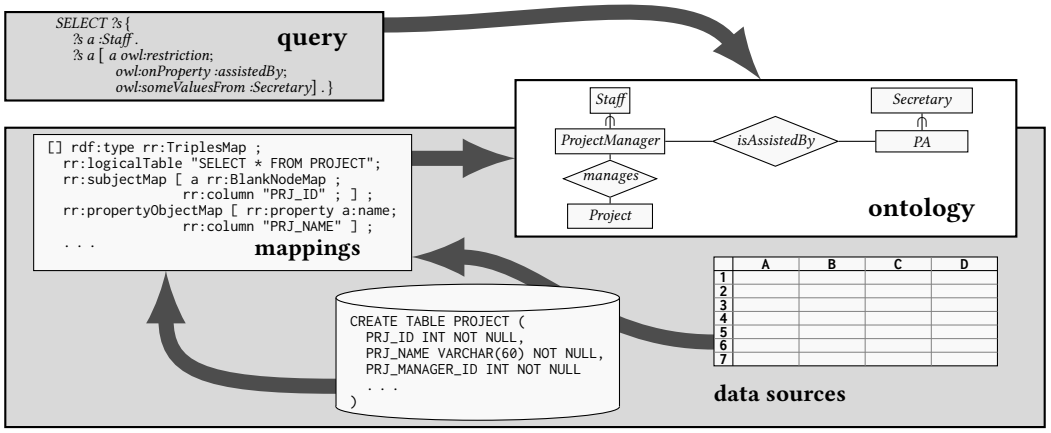


Fig. 1. Ontology-based data access.

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:

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

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

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

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 ground atoms), $\mathcal{A}$, in the signature of the ontology. For example, the global-as-view (GAV) mappings

$$\forall x, y, z \left(\mathtt{PROJECT}(x, y, z) \to \mathit{managesProject}(z, x)\right),$$
$$\forall x, y \left(\mathtt{STAFF}(x, y) \land (y = 2) \to \mathit{ProjectManager}(x)\right)$$

populate the ontology predicates *managesProject* and *ProjectManager* with values from the database relations `PROJECT` and `STAFF`, respectively. In the query rewriting approach of Poggi et al. [72], 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

$$\Phi(x) \;=\; \exists y \left[\mathit{Staff}(x) \land \mathit{isAssistedBy}(x, y) \land (\mathit{Secretary}(y) \lor \mathit{PA}(y))\right] \lor \mathit{ProjectManager}(x) \lor \exists z \, \mathit{managesProject}(x, z)$$

is an *FO-rewriting* of the *ontology-mediated query* (OMQ) $\boldsymbol{Q}(x) = (\mathcal{T}, \boldsymbol{q}(x))$ over any RDF graph $\mathcal{A}$ in the sense that a is an answer to $\Phi(x)$ over $\mathcal{A}$ if and only if $\boldsymbol{q}(a)$ 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 Φ 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. [23] and later extended by Artale et al. [6], 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 [9, 21, 27]. We note in passing that while most work on OBDA (including the present article) 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 [13, 43, 62, 78]. 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 to the setting, which are, however, necessarily limited to enable 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 [72], Presto/Prexto [79, 80], tree witness rewriting [57]), sets of tuple-generating dependencies (Nyaya [38], PURE [59]), and more expressive ontology languages that require recursive datalog rewritings (Requiem [70], Rapid [26], Clipper [30] and Kyrie [69]). A few mature OBDA systems have also recently emerged: pioneering MASTRO [22], commercial Stardog [71] and Ultrawrap [81], and the Optique platform [33] based on the query answering engine Ontop [61, 77]. 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 [53] and Statoil [52] experimenting with OBDA technologies to streamline the process of data access for their engineers.[3]

1.2 Problems: Succinctness and Complexity

In this article, 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* is to understand how difficult it is to construct rewritings for OMQs in a given class and, in particular, to determine whether OMQs in the class have polynomial-size rewritings or not. In other words, the succinctness problem clarifies the computational cost of the *reduction* of OMQ answering to database query evaluation. The original FO-rewriting of any given OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ suggested by Calvanese et al. [23] and called the 'perfect reformulation' is a union of CQs (UCQ) of size $|\mathcal{T}|^{|\boldsymbol{q}|} \cdot 2^{O(|\boldsymbol{q}|^2)}$. Having observed that UCQ-rewritings are prohibitively large in practice, Rosati and Almatelli [80] designed an algorithm for a shorter rewriting of $\boldsymbol{Q}$ into a nonrecursive datalog (NDL) program of size $|\mathcal{T}|^{O(1)} \cdot 2^{O(|\boldsymbol{q}|)}$. Kikot et al. [57] and Thomazo [84] identified common structures in UCQ-rewritings and compactified them into unions of semiconjunctive queries (USCQs)—positive-existential (PE) formulas with matrices of the form $\vee\wedge\vee$—of size $|\mathcal{T}| \cdot 2^{O(|\boldsymbol{q}|^2)}$. The first lower bounds on the size of FO-, PE- and NDL-rewritings[4] were obtained

[1] https://www.w3.org/TR/owl2-profiles/#OWL_2_QL
[2] http://www.w3.org/TR/sparql11-entailment
[3] See, e.g., http://optique-project.eu.
[4] Note that domain-independent FO-rewritings correspond to plain SQL queries, PE-rewritings to Select-Project-Join-Union (SPJU) queries, and NDL-rewritings to SPJU queries with views.

by Gottlob et al. [34] who constructed a sequence of OMQs (with tree-shaped CQs) whose PE- and NDL-rewritings can only be of exponential size, while FO-rewritings are superpolynomial unless NP $\subseteq$ P/poly.

To understand how optimal OBDA via OMQ rewriting can be, we also have to 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 $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ from a certain class, a data instance $\mathcal{A}$ and a tuple $\boldsymbol{a}$ of constants from $\mathcal{A}$, decide whether $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$. It is not hard to see that this problem is NP-complete [6, 23], with the lower bound inherited from the complexity of CQ evaluation. The combined complexity of CQ evaluation has been thoroughly investigated in database theory. In particular, it is known that tree-shaped CQs and, more generally, CQs of bounded treewidth are tractable, LogCFL-complete to be more precise [25, 36, 42, 88]. The presence of ontologies in OMQs makes a transfer of these positive results to the OBDA setting impossible: indeed, answering OMQs with tree-shaped CQs is NP-hard [56]. The tractability of CQs can be transferred to OMQs only at the expense of sacrificing the expressivity of the ontology language *OWL 2 QL*, for example, by disallowing 'role inclusions' [14].

In this article, we obtain solutions to the following major research problems:

- give an interesting—both theoretically and practically—classification of all OMQs according to the structure of their ontologies and CQs;
- determine whether OMQs in each of the identified classes have polynomial-size PE-, NDL-, and FO-rewritings;
- determine the combined complexity of answering OMQs in each of the classes.

Extended abstracts with initial results that ultimately led to the current article appeared in the Proceedings of the ACM/IEEE Symposium on Logic in Computer Science [11, 55].

1.3 Our Contribution

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. Tree-shapedness is especially relevant in the context of OBDA: 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. For instance, the NPD FactPages ontology,[5] which was designed to facilitate querying the datasets of the Norwegian Petroleum Directorate,[6] is of depth 5. A typical example of an ontology axiom causing infinite depth is $\forall x \left(\mathit{Person}(x) \rightarrow \exists y \left(\mathit{ancestor}(y, x) \land \mathit{Person}(y)\right)\right)$.

Figure 2a gives a summary of the succinctness results obtained in this article. 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, where tree-shaped OMQs (with CQs of treewidth 1) have polynomial-size Π_4-PE-rewritings (with matrices of the form $\land\lor\land\lor$). 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 OMQs with ontologies

[5] http://sws.ifi.uio.no/project/npd-v2
[6] http://factpages.npd.no/factpages

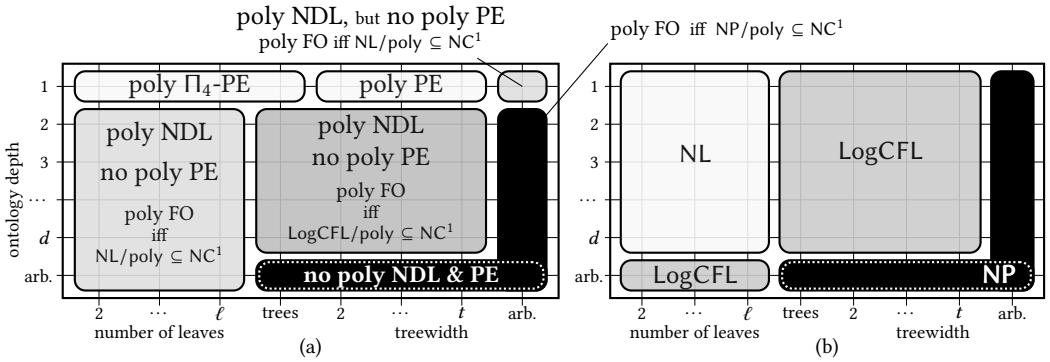


Fig. 2. (a) Succinctness of OMQ rewritings, and (b) combined complexity of OMQ answering (tight bounds).

of depth 1 and CQs of bounded treewidth—turns out to be equivalent to one of the major open problems in computational complexity such as[7] NP/poly $\subseteq^?$ NC^1. The only previously known result in Fig. 2a is indicated by the white dotted line; for details, see [34, 54].

To obtain the new results in Fig. 2a, we develop a novel framework that connects succinctness of rewritings and circuit complexity, a branch of computational complexity theory that classifies Boolean functions according to the size of circuits (and formulas) computing them. Our starting point is the observation that the tree-witness PE-rewriting of an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ constructed by Kikot et al. [57] 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 with (possibly negated) Boolean variables or constants 0 and 1, as a program computing a Boolean function f_H that returns 1 on an assignment to the variables iff there is an independent subset of hyperedges covering all vertices labelled with 0 (under the assignment). 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. 2a, 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 at most 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 NL/poly $\subseteq$ P/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 a superpolynomial lower bound for PE-rewritings. It also follows that all such OMQs have polynomial-size FO-rewritings just in case NC^1 = NL/poly.

The succinctness results in Fig. 2a, characterising the complexity of the reduction to plain database query evaluation, are complemented by the combined complexity results in Fig. 2b, where the only previously known result [56] is encircled by a white dotted line. Here, we prove that, surprisingly, answering OMQs with ontologies of bounded depth and CQs of bounded treewidth

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

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 in spite of a possibly infinite chase, LogCFL-complete to be more precise. Thus, in our classification, only the OMQs with arbitrary ontologies and bounded treewidth CQs turn out to be more complex than their underlying CQs (unless LogCFL = NP).

The plan of the article is as follows. Section 2 introduces *OWL 2 QL*, OMQs and rewritings. Section 3 defines tree-witness rewritings. Section 4 reduces the succinctness problem for OMQ rewritings to the succinctness problem for hypergraph Boolean functions associated with the tree-witness rewritings. Sections 5 and 6 introduce hypergraph programs for computing these functions and establish a correspondence between classes of OMQs in Fig. 2 and classes of hypergraph programs. Section 7 characterises the computational power of hypergraph programs in these classes by relating them to standard models of computation for Boolean functions. Section 8 uses the results of the previous four sections and known facts from circuit complexity to obtain the upper and lower bounds on the size of PE-, NDL- and FO-rewritings in Fig. 2a; a roadmap for the succinctness results is given in Fig. 19 (Section 8). Section 9 establishes the combined complexity results in Fig. 2b. We conclude in Section 10 by discussing the obtained succinctness and complexity results and formulating a few open problems. All omitted proofs can be found in Appendix A.

1.4 Some Remarks on Related OBDA Research

In our comprehensive analysis, we slightly simplify the general OBDA setting by assuming that data is given in the form of RDF graph and leave mappings out of the picture (in fact, GAV mappings only polynomially increase the size of FO-rewritings over RDF graphs). In practice, however, mappings play an important role, and their structure can be crucial for the performance of OBDA systems; see Section 10.

As is well-known in database theory, to find answers to an OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ over a data instance $\mathcal{A}$, one can construct the chase of $\mathcal{A}$ with $\mathcal{T}$ and evaluate $\boldsymbol{q}$ over it; see Section 3 for details. This approach to OMQ answering is known as materialisation or forward chaining. In the context of OBDA, there can be two main obstacles to materialisation. First, proprietary data is often not available for manipulations, and second, the chase with *OWL 2 QL* ontologies may be infinite. In the combined approach to OMQ answering, the infinite set of labelled nulls of the chase is encoded by a small number of their representatives and the CQ $\boldsymbol{q}$ is rewritten in order to eliminate spurious answers [60, 68] or a special filtering procedure is used to get rid of them [67]. Gottlob and Schwentick [39] and Gottlob et al. [34] showed that every $\boldsymbol{Q}$ has a polynomial-size PE-rewriting over any given data extended with two special constants, which are used by extra existential quantifiers in the rewriting to 'guess' a derivation of $\boldsymbol{q}$ in the chase (cf. also [8] for a succinctness trick in the same vein). Gottlob et al. [37] extended the polynomial combined approach to OMQs with linear tgds.

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

$$\forall x \left(\tau(x) \to \tau'(x)\right), \qquad \forall x \left(\tau(x) \land \tau'(x) \to \bot\right),$$
$$\forall x, y \left(\varrho(x,y) \to \varrho'(x,y)\right), \qquad \forall x, y \left(\varrho(x,y) \land \varrho'(x,y) \to \bot\right),$$
$$\forall x\, \varrho(x,x), \qquad \forall x \left(\varrho(x,x) \to \bot\right),$$

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

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

(Strictly speaking, *OWL 2 QL* ontologies can also contain inequalities $a \neq b$, for constants a and b. However, they have no impact on the problems considered in this article, and so will be ignored.)

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

$$\begin{aligned}
&\forall x \left(\mathit{GasPipeline}(x) \to \mathit{Pipeline}(x) \right), \\
&\forall x \left(\mathit{FieldOwner}(x) \leftrightarrow \exists y\, \mathit{ownerForField}(x, y) \right), \\
&\forall y \left(\exists x\, \mathit{ownerForField}(x, y) \to \mathit{Field}(y) \right), \\
&\forall x, y \left(\mathit{shallowWellboreForField}(x, y) \to \mathit{wellboreForField}(x, y) \right), \\
&\forall x, y \left(\mathit{isGeometryOfFeature}(x, y) \leftrightarrow \mathit{hasGeometry}(y, x) \right).
\end{aligned}$$

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

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

where

$$\zeta(x) \ ::= \ \tau(x) \ | \ \zeta_1(x) \land \zeta_2(x) \ | \ \exists y \left(\varrho_1(x, y) \land \cdots \land \varrho_k(x, y) \land \zeta(y) \right).$$

It is readily seen that such sentences are syntactic sugar and can be eliminated by means of linearly many extra axioms. Indeed, any axiom of the form (2) with $\zeta(x) = \exists y \left(\varrho_1(x, y) \land \cdots \land \varrho_k(x, y) \land \zeta'(y) \right)$ can be replaced by the following axioms, for a *fresh* P_ζ and $i = 1, \ldots, k$:

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

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 (recursively) 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:

$$\begin{aligned}
\boldsymbol{q}(x_1, x_2, x_3) \ = \ \exists y, z \Big[&\mathit{ProductionLicence}(x_1) \land \mathit{operatorForLicence}(y, x_1) \land \\
&\mathit{ProductionLicenceOperator}(y) \land \mathit{dateOperatorValidFrom}(y, x_2) \land \\
&\mathit{licenceOperatorCompany}(y, z) \land \mathit{name}(z, x_3) \Big].
\end{aligned}$$

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 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 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; $\boldsymbol{q}$ *tree-shaped* if $G_{\boldsymbol{q}}$ is a tree[8], and $\boldsymbol{q}$ is *linear* if $G_{\boldsymbol{q}}$ is a tree with at most two leaves.

An *OWL 2 QL ontology-mediated query* (OMQ) is a pair $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ of an *OWL 2 QL* ontology $\mathcal{T}$ and a CQ $\boldsymbol{q}(\boldsymbol{x})$. 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 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 $\boldsymbol{q}$ 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 [64] 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, then 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 $\Phi(\boldsymbol{x})$, 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 $\mathsf{ind}(\mathcal{A})$,

$$\mathcal{T}, \mathcal{A} \models \boldsymbol{q}(\boldsymbol{a}) \qquad \text{iff} \qquad \mathcal{I}_{\mathcal{A}} \models \Phi(\boldsymbol{a}), \tag{4}$$

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 $\Phi(\boldsymbol{x})$ is a positive existential formula—that is, $\Phi(\boldsymbol{x}) = \exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$ with φ constructed from atoms (possibly with equality) using $\wedge$ and $\vee$ only—we call it a *PE-rewriting of* $\boldsymbol{Q}(\boldsymbol{x})$. A PE-rewriting whose matrix φ is a disjunction of conjunctions ($\vee\wedge$) is known as a *UCQ-rewriting*; if φ takes the form $\vee\wedge\vee$ or $\wedge\vee\wedge\vee$, then we call it a Σ_3*-PE* or Π_4*-PE rewriting*, respectively. The size $|\Phi|$ of a rewriting Φ is the number of symbols in it.

We also consider rewritings in the form of nonrecursive datalog queries. Recall [1] that a *datalog program*, Π, is a finite set of Horn clauses $\forall \boldsymbol{x}\, (\gamma_1 \wedge \dots \wedge \gamma_m \to \gamma_0)$, 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; program Π is *nonrecursive* if this dependence relation is acyclic. We consider only constant-free datalog programs.

Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ, Π a nonrecursive program and G an $|\boldsymbol{x}|$-ary predicate. The pair $\Phi(\boldsymbol{x}) = (\Pi, G(\boldsymbol{x}))$ is an *NDL-rewriting of* $\boldsymbol{Q}(\boldsymbol{x})$ if, for any data instance $\mathcal{A}$ and 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 [1].

Remark 1. As defined, FO- and PE-rewritings are not necessarily domain-independent queries, while NDL-rewritings are not necessarily safe [1]. 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 the signature is $\{A, P\}$, then a domain-independent relativisation of $(x = x)$ is the PE-rewriting $\big(A(x) \vee \exists y\, P(x, y) \vee \exists y\, P(y, x)\big) \wedge (x = x)$. Note that if we exclude from *OWL 2 QL* reflexivity and $\top$ on the left-hand side, then rewritings are guaranteed to be domain-independent, and no

[8] Tree-shaped CQs also go by the name of *acyclic queries* [14, 88].

relativisation is required. In any case, rewritings are 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 [22] and also established theoretically [54] that sometimes the rewritings are prohibitively large—exponentially-large in the size of the original CQ, to be more precise. These observations 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) [85], but also on the combined complexity that takes into account the contribution of ontologies and queries.

3 TREE-WITNESS REWRITINGS

Now we define one particular rewriting of *OWL 2 QL* OMQs that will play a key role in the succinctness and complexity analysis later in the article. This rewriting is a modification of the tree-witness PE-rewriting originally introduced by Kikot et al. [57] (cf. [59, 60, 65] for similar ideas).

We begin with two simple observations that will help us remove unneeded clutter from 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 [20]. Thus, from now on we 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 $\Phi(\boldsymbol{x})$ 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 or the mapping [77]. We say that a data instance $\mathcal{A}$ is *complete*[9] for an ontology $\mathcal{T}$ if $S(\boldsymbol{a}) \in \mathcal{A}$ whenever $\mathcal{T}, \mathcal{A} \models S(\boldsymbol{a})$, 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 $\Phi(\boldsymbol{x})$ 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 $\Phi'(\boldsymbol{x})$ of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data instances with $|\Phi'| \le |\Phi| \cdot |\mathcal{T}|$. 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^*. Let Π' be the union of Π^* and the following clauses for predicates A and P in Π:

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

(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, and $|\Pi'| \le |\Pi| \cdot |\mathcal{T}|$. The cases of PE- and FO-rewritings are similar:

[9] Rodriguez-Muro et al. [77] used the term 'H-completeness'; see also [58].

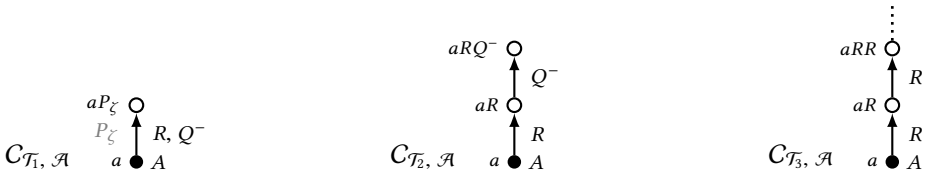


Fig. 3. Canonical models in Example 3.2.

we replace each $A(x)$ with a disjunction of $\tau(x)$, for τ with $\mathcal{T} \models \tau(x) \to A(x)$, and each $P(x,y)$ with a disjunction of $\varrho(x,y)$, for ϱ with $\mathcal{T} \models \varrho(x,y) \to P(x,y)$, and $x = y$ if $\mathcal{T} \models P(x,x)$, where the empty disjunction is $\bot$. □

As is well-known [1], for every pair $(\mathcal{T},\mathcal{A})$, there is a *canonical model* (or *chase*) $C_{\mathcal{T},\mathcal{A}}$ such that $\mathcal{T},\mathcal{A} \models \boldsymbol{q}(\boldsymbol{a})$ iff $C_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(\boldsymbol{a})$, for all CQs $\boldsymbol{q}(\boldsymbol{x})$ and $\boldsymbol{a}$ in $\mathsf{ind}(\mathcal{A})$. In our proofs, we use the following definition of $C_{\mathcal{T},\mathcal{A}}$, where without loss of generality we assume that $\mathcal{T}$ does not contain binary predicates P with $\mathcal{T} \models \forall x, y\, P(x,y)$. 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 $\Delta^{C_{\mathcal{T},\mathcal{A}}}$ of $C_{\mathcal{T},\mathcal{A}}$ 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 $C_{\mathcal{T},\mathcal{A}}$ 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 $C_{\mathcal{T},\mathcal{A}}$ by itself, and unary and binary predicates are interpreted as follows: for any $u, v \in \Delta^{C_{\mathcal{T},\mathcal{A}}}$,

- $C_{\mathcal{T},\mathcal{A}} \models A(u)$ iff either $u \in \mathsf{ind}(\mathcal{A})$ and $\mathcal{T},\mathcal{A} \models A(u)$, or $u = w\varrho$, for some word w and ϱ with $\mathcal{T} \models \exists y\, \varrho(y,x) \to A(x)$;
- $C_{\mathcal{T},\mathcal{A}} \models P(u,v)$ iff one of the three options 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*) $v = u\varrho$ or $u = v\varrho^-$, for ϱ with $\mathcal{T} \models \varrho(x,y) \to P(x,y)$.

Example 3.2. Consider the following ontologies:

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

The canonical models of $(\mathcal{T}_i,\mathcal{A})$ with $\mathcal{A} = \{A(a)\}$, for $i = 1,2,3$, are shown in Fig. 3, where P_ζ is the normalisation predicate for $\zeta(x) = \exists y\, (R(x,y) \wedge Q(y,x))$. 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 $C_{\mathcal{T}}^{\tau(a)}$ 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* n, $1 \leq n < \omega$, if

(*i*) there is no ϱ with $\mathcal{T} \models \varrho(x,x)$,
(*ii*) at least one of the $C_{\mathcal{T}}^{\tau(a)}$ contains a word $a\varrho_1 \dots \varrho_n$, but
(*iii*) none of the $C_{\mathcal{T}}^{\tau(a)}$ contains 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 $C_{\mathcal{T},\mathcal{A}}$ whose domain consists of words of the form $a\varrho_1 \dots \varrho_n$ with $n \leq p(|\boldsymbol{Q}|)$ [20, 48]. (In general, the bounded derivation depth property of an ontology language is a necessary and sufficient condition of FO-rewritability [34].)

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 $C_{\mathcal{T},\mathcal{A}}$ with the domain $\{aw \in C_{\mathcal{T},\mathcal{A}} \mid a \in \mathsf{ind}(\mathcal{A}),\ w \in \Omega_{\boldsymbol{Q}}\}$. We say that a class $\boldsymbol{Q}$ of OMQs has the *polynomial fundamental set property* (PFSP) if there is a polynomial p such that every $\boldsymbol{Q}$ in $\boldsymbol{Q}$ has a fundamental set $\Omega_{\boldsymbol{Q}}$ with $|\Omega_{\boldsymbol{Q}}| \leq p(|\boldsymbol{Q}|)$. The class of OMQs with ontologies of finite depth and tree-shaped CQs does not have the PFSP [54]. 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 3.8 to be proved below:

THEOREM 3.3. *The class of OMQs whose ontologies contain no 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.

3.1 Basic Tree-Witness Rewriting

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[10] of variables in $\boldsymbol{q}$ with $\mathfrak{t}_\mathsf{i} \subseteq \boldsymbol{y}$ and $\mathfrak{t}_\mathsf{i} \neq \emptyset$, let

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

If $\boldsymbol{q}_\mathfrak{t}$ is a minimal subset of $\boldsymbol{q}$ for which there is a homomorphism $h\colon \boldsymbol{q}_\mathfrak{t} \to C_\mathcal{T}^{\tau(a)}$ 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 $\mathfrak{t}$ a *tree witness for* $\boldsymbol{Q}(\boldsymbol{x})$ *generated by* τ (and *induced by* h). Note that the same tree witness 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*. By definition, the answer variables $\boldsymbol{x}$ in $\boldsymbol{q}(\boldsymbol{x})$ cannot be internal. The variables in $\mathfrak{t}_\mathsf{r}$, if any, are called *root variables*. If $\mathfrak{t}$ has no root variables, then $\boldsymbol{q}_\mathfrak{t}$ is a connected component of $\boldsymbol{q}$, in which case we call $\mathfrak{t}$ *detached*. 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*:

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

Note that $\Phi_{\mathsf{tw}}(\boldsymbol{x})$ is essentially a Σ_3-PE formula (because the $\mathsf{tw}_\mathfrak{t}$ contain disjunctions); Proposition 3.1 would then produce a Σ_3-PE rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over arbitrary data.

Remark 2. As the normalisation predicates P_ζ cannot occur in data instances, we 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.

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

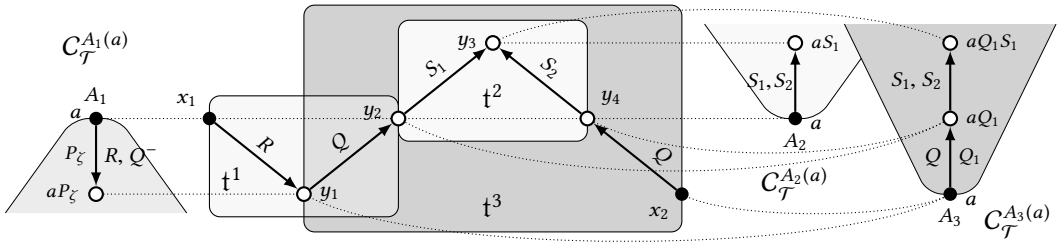


Fig. 4. Tree witnesses in Example 3.4.

Example 3.4. Consider the OMQ $\boldsymbol{Q}(x_1, x_2) = (\mathcal{T}, \boldsymbol{q}(x_1, x_2))$, where

$$\boldsymbol{q}(x_1, x_2) \;=\; \exists y_1, y_2, y_3, y_4 \big(R(x_1, y_1) \land Q(y_1, y_2) \land S_1(y_2, y_3) \land S_2(y_4, y_3) \land Q(x_2, y_4)\big)$$

and $\mathcal{T}$ contains the following sentences:

$$A_1(x) \to \exists y \big(R(x, y) \land Q(y, x)\big), \qquad Q_1(x, y) \to Q(x, y), \tag{7}$$
$$A_2(x) \to \exists y\, S_1(x, y), \qquad S_1(x, y) \to S_2(x, y), \tag{8}$$
$$A_3(x) \to \exists y\, Q_1(x, y), \qquad \exists y\, Q_1(y, x) \to \exists y\, S_1(x, y). \tag{9}$$

The CQ is shown in Fig. 4 alongside the $C_{\mathcal{T}}^{A_i(a)}$, where P_ζ is the normalisation predicate for the first axiom. When depicting CQs, we use ● for answer and ○ for existentially quantified variables. There are three tree witnesses, t^1, t^2 and t^3, for $\boldsymbol{Q}(x_1, x_2)$ with

$$\boldsymbol{q}_{\mathrm{t}^1} = \big\{ R(x_1, y_1),\ Q(y_1, y_2) \big\}, \qquad \boldsymbol{q}_{\mathrm{t}^2} = \big\{ S_1(y_2, y_3),\ S_2(y_4, y_3) \big\} \quad \text{and}$$
$$\boldsymbol{q}_{\mathrm{t}^3} = \big\{ Q(y_1, y_2),\ S_1(y_2, y_3),\ S_2(y_4, y_3),\ Q(x_2, y_4) \big\}$$

shown in Fig. 4 as shaded rectangles. The tree witness $\mathrm{t}^1 = (\mathrm{t}^1_{\mathrm{r}}, \mathrm{t}^1_{\mathrm{i}})$ with $\mathrm{t}^1_{\mathrm{r}} = \{x_1, y_2\}$ and $\mathrm{t}^1_{\mathrm{i}} = \{y_1\}$ is generated by $A_1(x)$, which gives

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

(Although t^1 is also generated by $\exists y\, P_\zeta(z, y)$, it is not included in $\mathrm{tw}_{\mathrm{t}^1}$ because P_ζ cannot occur in data instances.) Similarly, for tree witnesses t^2 and t^3, we have

$$\mathrm{tw}_{\mathrm{t}^2}(y_2, y_4) = \exists z \big((y_2 = z) \land (y_4 = z) \land \big(A_2(z) \lor \exists y\, S_1(z, y) \lor \exists y\, Q_1(y, z)\big)\big),$$
$$\mathrm{tw}_{\mathrm{t}^3}(y_1, x_2) = \exists z \big((y_1 = z) \land (x_2 = z) \land \big(A_3(z) \lor \exists y\, Q_1(z, y)\big)\big).$$

Note that t^2 is generated by $A_2(z)$, $\exists y\, S_1(z, y)$ and $\exists y\, Q_1(y, z)$. As t^3 is conflicting with both t^1 and t^2, the set Θ_Q contains five independent subsets: $\emptyset$, $\{\mathrm{t}^1\}$, $\{\mathrm{t}^2\}$, $\{\mathrm{t}^3\}$ and $\{\mathrm{t}^1, \mathrm{t}^2\}$, each of which gives rise to a disjunct in the following tree-witness rewriting $\Phi_{\mathrm{tw}}(x_1, x_2)$ of $\boldsymbol{Q}(x_1, x_2)$ over complete data instances:

$$\begin{aligned}
&\exists y_1, y_2, y_3, y_4 \big(R(x_1, y_1) \land Q(y_1, y_2) \land S_1(y_2, y_3) \land S_2(y_4, y_3) \land Q(x_2, y_4)\big) \quad \lor \\
&\qquad \exists y_2, y_3, y_4 \big(\mathrm{tw}_{\mathrm{t}^1}(x_1, y_2) \land S_1(y_2, y_3) \land S_2(y_4, y_3) \land Q(x_2, y_4)\big) \quad \lor \\
&\qquad \exists y_1, y_2, y_4 \big(R(x_1, y_1) \land Q(y_1, y_2) \land \mathrm{tw}_{\mathrm{t}^2}(y_2, y_4) \land Q(x_2, y_4)\big) \quad \lor \\
&\quad \exists y_1 \big(R(x_1, y_1) \land \mathrm{tw}_{\mathrm{t}^3}(y_1, x_2)\big) \quad \lor \quad \exists y_2, y_4 \big(\mathrm{tw}_{\mathrm{t}^1}(x_1, y_2) \land \mathrm{tw}_{\mathrm{t}^2}(y_2, y_4) \land Q(x_2, y_4)\big).
\end{aligned}$$

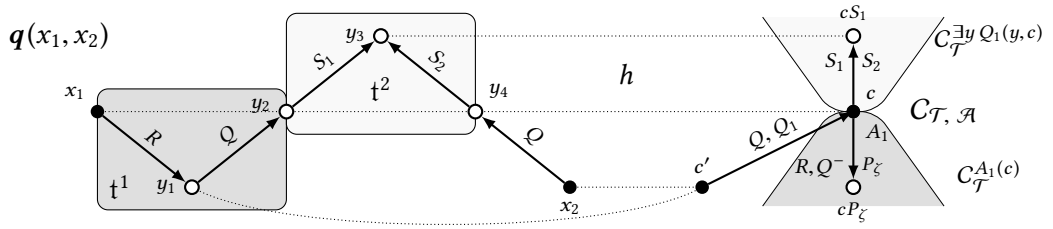


Fig. 5. A homomorphism $h\colon \boldsymbol{q}(c, c') \to C_{\mathcal{T},\mathcal{A}}$ and an independent set $\{\mathfrak{t}^1, \mathfrak{t}^2\}$ of tree witnesses.

Theorem 3.5 ([57]). *For any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, any data instance $\mathcal{A}$ 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 $\mathcal{I}_{\mathcal{A}} \models \Phi_{\mathsf{tw}}(\boldsymbol{a})$. In other words, $\Phi_{\mathsf{tw}}(\boldsymbol{x})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.*

Intuitively, for any homomorphism $h\colon \boldsymbol{q}(\boldsymbol{a}) \to C_{\mathcal{T},\mathcal{A}}$, the sub-CQs of $\boldsymbol{q}$ that are mapped by h to sub-models of the form $C_{\mathcal{T}}^{\tau(a)}$ define an independent set Θ of tree witnesses; see Fig. 5, where $\mathcal{A}$ consists of $Q_1(c', c)$, $Q(c', c)$ and $A_1(c)$. Conversely, if Θ is an independent subset of $\Theta_{\boldsymbol{Q}}$, then the homomorphisms inducing the tree witnesses in Θ can be pieced together into a homomorphism from $\boldsymbol{q}(\boldsymbol{a})$ to $C_{\mathcal{T},\mathcal{A}}$—provided that the $S(\boldsymbol{z})$ from $\boldsymbol{q} \setminus \boldsymbol{q}_\Theta$ and the $\mathsf{tw}_{\mathfrak{t}}(\mathfrak{t}_{\mathsf{r}})$ for $\mathfrak{t} \in \Theta$ hold in $\mathcal{I}_{\mathcal{A}}$: in Fig. 5, the non-conflicting $\mathfrak{t}^1$ and $\mathfrak{t}^2$ are mapped to fragments isomorphic to $C_{\mathcal{T}}^{\exists y\, Q_1(y,a)}$ and $C_{\mathcal{T}}^{A_1(a)}$, respectively, while the remaining $Q(x_2, y_4)$ is mapped to the Q-atom in the data instance $\mathcal{I}_{\mathcal{A}}$.

The *size* of the tree-witness PE-rewriting Φ_{tw} depends on the number of tree witnesses in the given OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$, namely, $|\Phi_{\mathsf{tw}}| = O(2^{|\Theta_{\boldsymbol{Q}}|} \cdot |\boldsymbol{Q}|^2)$ because $|\mathsf{tw}_{\mathfrak{t}}| = O(|\boldsymbol{Q}|)$, for each $\mathfrak{t} \in \Theta_{\boldsymbol{Q}}$ (note that $|\Theta_{\boldsymbol{Q}}| \leq 3^{|\boldsymbol{q}|}$).

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 (that is, $\boldsymbol{q}_{\mathfrak{t}} \subseteq \boldsymbol{q}_{\mathfrak{t}'}$), then the Σ_3-PE rewriting Φ_{tw} can be equivalently transformed to the Π_4-PE rewriting

$$\exists \boldsymbol{y} \bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q}} \Big(S(\boldsymbol{z}) \;\vee \bigvee_{\mathfrak{t} \in \Theta_{\boldsymbol{Q}} \text{ with } S(\boldsymbol{z}) \in \boldsymbol{q}_{\mathfrak{t}}} \mathsf{tw}_{\mathfrak{t}}(\mathfrak{t}_{\mathsf{r}}) \Big)$$

which, unlike Φ_{tw}, is linear in $|\Theta_{\boldsymbol{Q}}|$—of size $O(|\Theta_{\boldsymbol{Q}}| \cdot |\boldsymbol{Q}|^2)$, to be more precise. In Example 3.4 without axioms (9), there are only two tree witnesses, $\mathfrak{t}^1$ and $\mathfrak{t}^2$, which are compatible, and so we obtain the following rewriting:

$$\exists y_1, y_2, y_3, y_4 \Big[\big(R(x_1, y_1) \vee \mathsf{tw}_{\mathfrak{t}^1}(x_1, y_2) \big) \wedge \big(Q(y_1, y_2) \vee \mathsf{tw}_{\mathfrak{t}^1}(x_1, y_2) \big) \wedge \big(S_1(y_2, y_3) \vee \mathsf{tw}_{\mathfrak{t}^2}(y_2, y_4) \big) \wedge \big(S_2(y_4, y_3) \vee \mathsf{tw}_{\mathfrak{t}^2}(y_2, y_4) \big) \wedge Q(x_2, y_4) \Big].$$

Thus, by increasing the alternation depth from Σ_3 to Π_4, we can make PE-rewritings more succinct. In Section 4, we translate the problem of finding succinct rewritings into the setting of Boolean functions, which is concerned with circuit complexity.

3.2 The Number of Tree Witnesses

OMQs with arbitrary axioms (and PFSP) can have *exponentially many* tree witnesses:

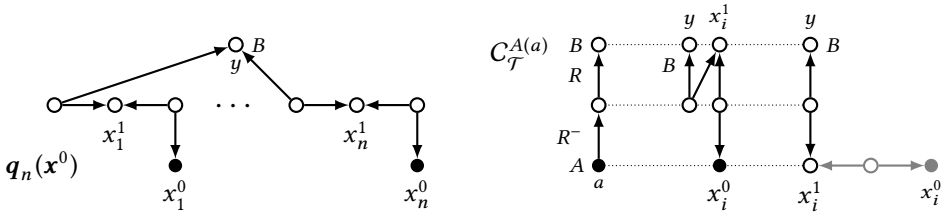


Fig. 6. The query $\boldsymbol{q}_n(\boldsymbol{x}^0)$ (all edges are labelled by R), the canonical model $\mathcal{C}_{\mathcal{T}}^{A(a)}$ (the normalisation predicates are not shown) and two ways of mapping a branch of the query to the canonical model in Example 3.6.

Example 3.6. Consider the tree-shaped OMQ $\boldsymbol{Q}_n(\boldsymbol{x}^0) = (\mathcal{T}, \boldsymbol{q}_n(\boldsymbol{x}^0))$, where

$$\mathcal{T} \;=\; \big\{\, A(x) \to \exists y \,\big(R(y,x) \land \exists z\,(R(y,z) \land B(z))\big) \,\big\},$$
$$\boldsymbol{q}_n(\boldsymbol{x}^0) \;=\; \exists y, \boldsymbol{y}^1, \boldsymbol{x}^1, \boldsymbol{y}^2 \Big[B(y) \;\land \bigwedge_{1 \le i \le n} \big(R(y_i^1, y) \land R(y_i^1, x_i^1) \land R(y_i^2, x_i^1) \land R(y_i^2, x_i^0)\big)\Big]$$

and $\boldsymbol{x}^k$ and $\boldsymbol{y}^k$ denote vectors of n variables x_i^k and y_i^k, for $1 \le i \le n$, respectively. The CQ is shown in Fig. 6 alongside the canonical model $\mathcal{C}_{\mathcal{T}}^{A(a)}$. The 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 $(\mathrm{t}_\mathrm{r}^{\boldsymbol{\alpha}}, \mathrm{t}_\mathrm{i}^{\boldsymbol{\alpha}})$ with $\mathrm{t}_\mathrm{r}^{\boldsymbol{\alpha}} = \{x_i^{\alpha_i} \mid 1 \le i \le n\}$. It is worth observing that the tree-witness rewriting of $\boldsymbol{Q}_n$ can be equivalently transformed into the following *polynomial-size* PE-rewriting:

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

The number of tree witnesses is, however, polynomial in two important cases.

Theorem 3.7. *OMQs $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with $\mathcal{T}$ of depth 1 have at most $|\boldsymbol{q}|$ tree witnesses, and every atom in $\boldsymbol{q}$ belongs to at most two tree witnesses.*

Proof. Suppose $\mathrm{t} = (\mathrm{t}_\mathrm{r}, \mathrm{t}_\mathrm{i})$ is a tree witness for $\boldsymbol{Q}$ and $y \in \mathrm{t}_\mathrm{i}$. Since $\mathcal{T}$ is of depth 1, $\mathrm{t}_\mathrm{i} = \{y\}$ and the set t_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 $A(u) \in \boldsymbol{q}$ belongs to $\boldsymbol{q}_\mathrm{t}$ iff $u = y$. An atom of the form $P(u,v) \in \boldsymbol{q}$ is in $\boldsymbol{q}_\mathrm{t}$ iff either $u = y$ or $v = y$. Therefore, $P(u,v) \in \boldsymbol{q}$ can only be covered by a unique tree witness with internal u and by a unique tree witness with internal v (if they exist). □

Theorem 3.8. *OMQs $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$, where $\mathcal{T}$ contains no 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 is at most 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 by at most one R, and so no point can have more than one R-successor, for any R. It follows that, for any $P(x,y) \in \boldsymbol{q}$, there is at most one tree witness $\mathrm{t} = (\mathrm{t}_\mathrm{r}, \mathrm{t}_\mathrm{i})$ with $P(x,y) \in \boldsymbol{q}_\mathrm{t}$, $x \in \mathrm{t}_\mathrm{r}$ and $y \in \mathrm{t}_\mathrm{i}$ ($P^-(y,x)$ may give another tree witness). □

3.3 Tree-Witness Rewriting Modified

To be able to deal with OMQs that have exponentially many tree witnesses, we slightly modify the tree-witness rewriting in Section 3.1. Suppose $\mathrm{t} = (\mathrm{t}_\mathrm{r}, \mathrm{t}_\mathrm{i})$ is a tree witness for $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ induced by a homomorphism $h\colon \boldsymbol{q}_\mathrm{t} \to \mathcal{C}_{\mathcal{T}}^{\tau(a)}$. We say that t is *ϱ-initiated* if every $h(z)$ with $z \in \mathrm{t}_\mathrm{i}$ is of the form $a\varrho w$, for some w. (Since $\boldsymbol{q}_\mathrm{t}$ is minimal, this is equivalent to having the property

for *some* $z \in \mathsf{t}_\mathsf{i}$.) For such ϱ, let $\varrho^*(x)$ be a disjunction of all $\tau(x)$ with $\mathcal{T} \models \tau(x) \to \exists y\, \varrho(x, y)$. In Example 3.4, the tree witness $\mathfrak{t}^2$ is generated by $A_2(z)$, $\exists y\, S_1(z, y)$, $\exists y\, Q_1(y, z)$; it is S_1-initiated but not Q_1-initiated, and

$$S_1^*(z) = A_2(z) \lor \exists y\, S_1(z, y) \lor \exists y\, Q_1(y, z).$$

Again, the disjunction $\varrho^*(x)$ includes only those $\tau(x)$ that do not contain normalisation predicates (even though ϱ itself can be one, like P_ζ for $\mathfrak{t}^1$ in Example 3.4).

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

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

Note that unlike $\mathsf{tw}_\mathfrak{t}$, the new formula $\mathsf{tw}'_\mathfrak{t}$ contains equalities for the variables from both t_i and t_r (variables t_i were not needed in (5)) as well as $\varrho^*(z)$ for *all* variables $z \in \mathsf{t}_\mathsf{i} \cup \mathsf{t}_\mathsf{r}$ (even though they are all equal under relevant assignments)—these redundancies are required to simplify the construction in the proof of Theorem 5.12 below. For the OMQ $\boldsymbol{Q}(x_1, x_2)$ in Example 3.4, $\Phi'_{\mathsf{tw}}(x_1, x_2)$ will contain disjuncts such as

$$\exists y_1, y_2, y_3, y_4 \left(R(x_1, y_1) \land Q(y_1, y_2) \land \Big[(y_2 = y_3) \land (y_3 = y_4) \land \bigwedge_{z \in \{y_2, y_3, y_4\}} S_1^*(z) \Big] \land Q(x_2, y_4) \right).$$

Although the size of both Φ_{tw} and Φ'_{tw} can be exponential in $|\boldsymbol{q}|$, the basic tree-witness rewriting Φ_{tw} can contain exponentially many distinct subformulas of the form $\mathsf{tw}_\mathfrak{t}$, whereas the modified rewriting Φ'_{tw} contains only a linear number of distinct atoms and subformulas of the form ϱ^*. This property will be used in Section 4.1. The proof of the following theorem is given in Appendix A.1:

Theorem 3.9. *For any OMQ $\boldsymbol{Q}(\boldsymbol{x})$, the formulas $\Phi_{\mathsf{tw}}(\boldsymbol{x})$ and $\Phi'_{\mathsf{tw}}(\boldsymbol{x})$ are equivalent, and so $\Phi'_{\mathsf{tw}}(\boldsymbol{x})$ is a PE-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.*

4 OMQ REWRITINGS AS BOOLEAN FUNCTIONS

Our aim now is to reduce the succinctness problem for OMQ rewritings to the succinctness problem for certain Boolean functions associated with the tree-witness rewritings.

We remind the reader (for details see, e.g., [5, 49]) 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). A gate of out-degree 0 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 $i > n$, 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. Any monotone circuit computes a monotone function, and any monotone Boolean function can be computed by a monotone circuit. *Boolean formulas* can be thought of as circuits in which every logic gate has at most one outgoing edge.

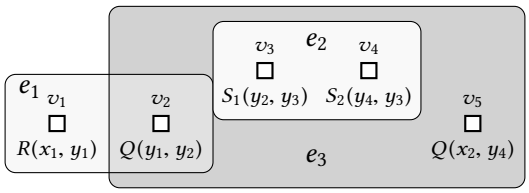


Fig. 7. The hypergraph $\mathcal{H}(\boldsymbol{Q})$ for $\boldsymbol{Q}$ from Example 3.4: each e_i corresponds to $\mathfrak{t}^i$.

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

$$f_H \quad = \bigvee_{E' \text{ independent}} \Big(\bigwedge_{v \in V \setminus V_{E'}} p_v \ \wedge \ \bigwedge_{e \in E'} p_e \Big). \tag{10}$$

The tree-witness PE-rewriting Φ_{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 $\boldsymbol{Q}(\boldsymbol{x})$. 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 (10) 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 $\boldsymbol{Q}(x_1, x_2)$ in Example 3.4, the hypergraph $\mathcal{H}(\boldsymbol{Q})$ has five vertices (one for each atom in the query) and three hyperedges (one for each tree witness) shown in Fig. 7. The tree-witness hypergraph function $f^\triangledown_{\boldsymbol{Q}}$ for $\boldsymbol{Q}$ is as follows:

$$\begin{aligned}\big(p_{R(x_1,y_1)} \wedge p_{Q(y_1,y_2)} \wedge p_{S_1(y_2,y_3)} \wedge p_{S_2(y_4,y_3)} \wedge p_{Q(x_2,y_4)}\big) \ \vee \ \big(p_{\mathfrak{t}^1} \wedge p_{S_1(y_2,y_3)} \wedge p_{S_2(y_4,y_3)} \wedge p_{Q(x_2,y_4)}\big) \\ \vee \ \big(p_{R(x_1,y_1)} \wedge p_{Q(y_1,y_2)} \wedge p_{\mathfrak{t}^2} \wedge p_{Q(x_2,y_4)}\big) \ \vee \ \big(p_{R(x_1,y_1)} \wedge p_{\mathfrak{t}^3}\big) \ \vee \ \big(p_{\mathfrak{t}^1} \wedge p_{\mathfrak{t}^2} \wedge p_{Q(x_2,y_4)}\big).\end{aligned}$$

Suppose the function $f^\triangledown_{\boldsymbol{Q}}$ for an OMQ $\boldsymbol{Q}(\boldsymbol{x})$ is computed by a Boolean formula χ. Consider the FO-formula $\Phi(\boldsymbol{x})$ obtained by replacing each $p_{S(\boldsymbol{z})}$ in χ with $S(\boldsymbol{z})$, each $p_\mathfrak{t}$ with $\mathsf{tw}_\mathfrak{t}$, and adding the appropriate prefix $\exists \boldsymbol{y}$. By comparing (10) and (6), we see that $\Phi(\boldsymbol{x})$ is an FO-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances. This proves the following theorem for FO- and PE-rewritings; NDL-rewritings are dealt with in Appendix A.2:

Theorem 4.2. *If $f^\triangledown_{\boldsymbol{Q}}$ is computed by a Boolean formula (monotone formula or monotone circuit) χ, then $\boldsymbol{Q}$ has an FO- (respectively, PE- or NDL-) rewriting of size $O(|\chi| \cdot |\boldsymbol{Q}|)$.*

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 $\mathfrak{t}$, rendering the reduction inefficient for OMQs with exponentially many tree witnesses. In this case, we associate with the modified rewriting $\Phi'_{\mathsf{tw}}(\boldsymbol{x})$ the monotone Boolean formula $f^{\blacktriangledown}_{\boldsymbol{Q}}$ obtained from $f^\triangledown_{\boldsymbol{Q}}$ by replacing each variable $p_\mathfrak{t}$, for $\mathfrak{t} = (\mathfrak{t}_\mathsf{r}, \mathfrak{t}_\mathsf{i})$, with

$$\bigwedge_{P(z,z') \in \boldsymbol{q}_\mathfrak{t}} p_{z=z'} \quad \wedge \quad \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \ \bigwedge_{z \in \mathfrak{t}_\mathsf{r} \cup \mathfrak{t}_\mathsf{i}} p_{\varrho^*(z)}, \tag{11}$$

where $p_{z=z'}$ and $p_{\varrho^*(z)}$ are propositional variables. Although the size of the resulting formula is exponential in $|\boldsymbol{Q}|$, the number of variables in it is *linear* in $|\boldsymbol{Q}|$, and we show in Appendix A.2:

Proposition 4.3. *The function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ can be computed by a nondeterministic algorithm that runs in polynomial time in the size of $\boldsymbol{Q}$.*

The proof of the following analogue of Theorem 4.2 is given in Appendix A.2:

Theorem 4.4. *If $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is computed by a Boolean formula (monotone formula or monotone circuit) χ, then $\boldsymbol{Q}$ has an FO- (respectively, PE- or NDL-) rewriting of size $O(|\chi| \cdot |\boldsymbol{Q}|)$.*

4.2 Primitive Evaluation Functions

To obtain lower bounds on the size of rewritings, we associate with every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ a third Boolean function, $f^{\triangle}_{\boldsymbol{Q}}$, that describes the result of evaluating $\boldsymbol{Q}$ on data instances with a single constant. Let $\boldsymbol{\gamma}$ be a function assigning a truth-value $\boldsymbol{\gamma}(S_i)$ to each unary or binary predicate S_i in $\boldsymbol{Q}$. We fix some order on the predicate names and assume that $\boldsymbol{\gamma} \in \{0,1\}^n$ for some n. We associate with $\boldsymbol{\gamma}$ the data instance

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

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

Theorem 4.5. *If $\Phi(\boldsymbol{x})$ is an FO- (respectively, PE- or NDL-) rewriting of $\boldsymbol{Q}(\boldsymbol{x})$, then $f^{\triangle}_{\boldsymbol{Q}}$ can be computed by a Boolean formula (respectively, monotone formula or monotone circuit) of size $O(|\Phi|)$.*

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

If $(\Pi, G(\boldsymbol{x}))$ is an NDL-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$, we replace all variables in Π with a and then perform the replacement described above. We now turn the resulting propositional NDL-program Π' into a monotone circuit computing $f^{\triangle}_{\boldsymbol{Q}}$. 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 variables in the body, or, if the body is empty, then we take the gate for constant 1. □

5 FROM OMQS TO 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.

5.1 Hypergraph Programs

A *hypergraph program* (HGP) P is a hypergraph $H = (V, E)$ each of whose vertices is labelled with 0, 1 or a literal over a list $p_1, \dots, p_n$ of propositional variables. (As usual, a *literal* is a propositional variable or its negation.) An *input* for P is a tuple $\boldsymbol{\alpha} \in \{0,1\}^n$, which is regarded as an assignment of truth values to $p_1, \dots, p_n$. 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 $|V| + |E|$.

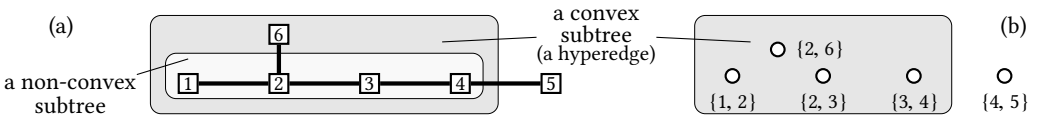


Fig. 8. Tree T underlying hypergraph H in Example 5.3.

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 by renaming (in particular, identifying) some of its variables and/or fixing them to 0 or 1. A hypergraph H and any HGP P based on H are said to be of *degree* (*at most*) d if every vertex in H belongs to (at most) d hyperedges.

Proposition 5.1. *For any hypergraph $H = (V, E)$ of degree at most d, there is a monotone HGP that computes f_H and is of degree at most* $\max(2, d)$ *and size* $O(|H|)$.

Proof. We label each $v \in V$ with the variable p_v. For each $e \in E$, we add a fresh vertex a_e labelled with 1 and a fresh vertex b_e labelled with p_e; then we add a_e to e and create a new hyperedge $e' = \{a_e, b_e\}$. 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 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. □

In general, the hypergraph $\mathcal{H}(\boldsymbol{Q})$ of a given OMQ $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ can be exponential in the size of $\boldsymbol{Q}$, and so Proposition 5.1 is not always helpful. However, if $\mathcal{T}$ is an ontology of depth 1 then, by Theorem 3.7, $\mathcal{H}(\boldsymbol{Q})$ is of degree at most 2 and its size does not exceed $2|\boldsymbol{q}|$. So, by Proposition 5.1, we obtain the following:

Corollary 5.2. *For every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with an ontology $\mathcal{T}$ of depth* 1, *there is a monotone HGP that computes $f^{\forall}_{\boldsymbol{Q}}$ and is of degree at most* 2 *and of size* $O(|\boldsymbol{q}|)$.

5.2 Tree Hypergraph Programs (THGPs)

We call an OMQ *tree-shaped* if its CQ is tree-shaped. We show that tree-shaped OMQs give rise to tree hypergraph programs, which are defined as follows.[11]

Suppose $T = (U, V)$ is an undirected tree with nodes $U \neq \emptyset$ and edges V (possibly none). A leaf is a node of degree 1. A tree $T' = (U', V')$ is a *subtree* of T if $U' \subseteq U$ and $V \subseteq V'$. We call T' *convex* if, for any non-leaf u in T', we have $\{u, u'\} \in V'$ whenever $\{u, u'\} \in V$. For $U' \subseteq U$, denote by $[U']$ the set of edges of the smallest convex subtree of T containing U'. A triple $H = (U, V, E)$ is a *tree hypergraph* if $T_H = (U, V)$ is a tree and (V, E) is a hypergraph with $E \subseteq \{\, [U'] \mid U' \subseteq U \,\}$. By definition, every hyperedge $e \in E$ induces a convex subtree T_e of T_H. The *boundary* of e is the set of leaves in T_e; the set of all other nodes of T_e forms the *interior* of e. We refer to (V, E) and T_H as the *reduct* and the *underlying tree* of H, respectively, and say that H is *based* on T_H. A hypergraph is *isomorphic to a tree hypergraph* (U, V, E) if it is isomorphic to its reduct. By the *size* of a tree hypergraph we understand the size of its reduct, $|V| + |E|$.

Example 5.3. Let $T = (U, V)$ be the tree depicted in Fig. 8a with nodes $\{1, \dots, 6\}$ and edges $\{1, 2\}, \{2, 3\}, \{2, 6\}, \{3, 4\}, \{4, 5\}$. Any tree hypergraph based on T, for instance, the one in Fig. 8b, has the set of vertices V (which are the edges of T) and its hyperedges may include the set

[11] Our definition of tree hypergraph is a minor variant of the notion of (sub)tree hypergraph (aka hypertree) from graph theory [18, 19, 31].

$\{\{1,2\},\{2,6\},\{2,3\},\{3,4\}\}$ (which can be denoted by $[1,4]$ or $[6,4]$ and which is shown by dark shading in Fig. 8) because the induced subtree is convex. On the other hand, $\{\{1,2\},\{2,3\},\{3,4\}\}$ (light shading in Fig. 8a) is not convex.

Recall that, by Theorem 3.7, if an OMQ $\boldsymbol{Q}$ has an ontology of depth 1, then $\mathcal{H}(\boldsymbol{Q})$ is of degree at most 2. The following analogue for tree-shaped OMQs is immediate from the definitions of tree witnesses and tree hypergraphs; see Appendix A.4:

Proposition 5.4. *If an OMQ $\boldsymbol{Q}$ has a tree-shaped CQ $\boldsymbol{q}$ with ℓ leaves, then $\mathcal{H}(\boldsymbol{Q})$ is isomorphic to a tree hypergraph based on a tree with* $\max(2,\ell)$ *leaves.*

Tree hypergraph programs (THGPs) are HGPs based on a tree hypergraph; they capture the computational power of tree hypergraph functions similarly to Proposition 5.1. The following is proved in Appendix A.5:

Proposition 5.5. *For any tree hypergraph H of degree at most d, there is a monotone THGP that computes f_H and is of degree at most* $\max(2,d)$ *and size $O(|H|)$.*

For tree hypergraphs of degree at most 2, we can even obtain *linear THGPs* of degree at most 2, which are based on tree hypergraphs with two leaves:

Theorem 5.6. *For any tree hypergraph H of degree at most 2, there is a monotone linear THGP that computes f_H and is of degree at most 2 and size $|H|^{O(1)}$.*

Proof. The construction of the THGPs uses *obstructions*, that is, sequences $(e_0,\dots,e_{2n-1})$, $n \geq 1$, of distinct hyperedges of H with $e_i \cap e_{i+1} \neq \emptyset$ for $0 \leq i < 2n-1$. Since no vertex belongs to more than two hyperedges, which induce subtrees of T_H, we have $e_i \cap e_j = \emptyset$ for $|i-j| > 1$; moreover, an obstruction is uniquely determined by e_0 and e_{2n-1}. So, there are $O(|H|^2)$ obstructions. An input $\boldsymbol{\alpha}$ *meets an obstruction* $(e_0,\dots,e_{2n-1})$ *from* v_0 *to* v_{2n-1} if

(O1) $\boldsymbol{\alpha}(p_{v_0}) = 0$ and $\boldsymbol{\alpha}(p_{e'}) = 0$, for any hyperedge $e' \neq e_0$ with $v_0 \in e'$;
(O2) $\boldsymbol{\alpha}(p_{v_{2n-1}}) = 0$ and $\boldsymbol{\alpha}(p_{e'}) = 0$, for any hyperedge $e' \neq e_{2n-1}$ with $v_{2n-1} \in e'$;
(O3) for every k, $1 \leq k < n$, there is $v \in e_{2k-1} \cap e_{2k}$ with $\boldsymbol{\alpha}(p_v) = 0$.

Intuitively, by (O1), any independent cover of zeros under $\boldsymbol{\alpha}$ contains e_0; but it cannot contain e_1, and so, by (O3), it contains e_2, and so on. Thus, e_{2n-1} cannot be in the independent cover contrary to (O2). We say that an input $\boldsymbol{\alpha}$ is *degenerate* if

(D) there is $v \in V$ such that $\boldsymbol{\alpha}(p_v) = 0$ and $\boldsymbol{\alpha}(p_e) = 0$ for all $e \in E$ with $v \in e$.

We claim (see Lemma A.3 in Appendix A.6) that $f_H(\boldsymbol{\alpha}) = 1$ iff $\boldsymbol{\alpha}$ neither is degenerate nor meets any obstruction. We construct a linear THGP of degree at most 2 for checking these conditions by using 'gadgets' of the form shown in Fig. 9. Such a gadget encodes a DNF: it outputs zero iff none of its disjuncts is true. The condition '$\boldsymbol{\alpha}$ does not meet an obstruction $(e_0,\dots,e_{2n-1})$ from v_0 to v_{2n-1}' can be expressed as a DNF as follows:

$$\underbrace{p_{v_0} \vee \gamma_{v_0,e_0}}_{\text{negation of (O1)}} \vee \underbrace{p_{v_{2n-1}} \vee \gamma_{v_{2n-1},e_{2n-1}}}_{\text{negation of (O2)}} \vee \underbrace{\bigvee\nolimits_{1\leq k<n} \Big[\bigwedge\nolimits_{v\in e_{2k-1}\cap e_{2k}} p_v\Big]}_{\text{negation of (O3)}},$$

where $\gamma_{v,e} = p_{e'}$ if $v \in e'$ and $e \neq e'$ (such e' is determined uniquely) or $\bot$ otherwise. Negation of (D) gives rise to a similar gadget. It remains to combine the gadgets for the obstructions in H and negated (D) into a single linear THGP of degree 2 by ordering the gadgets linearly and inserting edges labelled with 1 between them (these hypergraph vertices do not belong to any hyperedges). The resulting THGP is of polynomial size. □

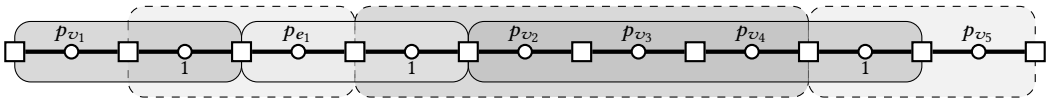


Fig. 9. Gadget encoding a DNF $p_{v_1} \vee p_{e_1} \vee (p_{v_2} \wedge p_{v_3} \wedge p_{v_4}) \vee p_{v_5}$: squares are the nodes of the underlying tree (a path graph), and circles are the vertices of the hypergraph (that is, the edges in the underlying tree).

Table 1. Monotone HGPs that compute hypergraph functions f_H for hypergraphs H.

	hypergraph H	monotone HGP P computing f_H	$\lvert P\rvert$
Prop. 5.1	hypergraph of degree $\leq d$	HGP of degree $\leq \max(2, d)$	$O(\lvert H\rvert)$
Prop. 5.5	tree hypergraph of degree $\leq d$	THGP of degree $\leq \max(2, d)$	$O(\lvert H\rvert)$
Th. 5.6	tree hypergraph of degree ≤ 2	linear THGP of degree ≤ 2	$\lvert H\rvert^{O(1)}$
Th. 5.8	tree hypergraph with $\leq \ell$ leaves	linear THGP	$O(\lvert H\rvert^{3\ell+1})$

We apply Theorem 5.6 as follows. Let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ with $\mathcal{T}$ of depth 1 and tree-shaped $\boldsymbol{q}$. By Proposition 5.4, $\mathcal{H}(\boldsymbol{Q})$ is isomorphic to a tree hypergraph. By Theorem 3.7, $\mathcal{H}(\boldsymbol{Q})$ is of linear size and degree at most 2. Therefore, we obtain:

Corollary 5.7. *For every tree-shaped OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with an ontology of depth 1, there is a monotone linear THGP that computes $f^{\triangledown}_{\boldsymbol{Q}}$ and is of degree at most 2 and size polynomial in $\lvert\boldsymbol{q}\rvert$.*

In fact, we can transform any THGP into a linear THGP (of exponential size in the number of leaves; cf. Theorem 5.6). Let $H = (U, V, E)$ be a tree hypergraph. Pick a node $r \in U$ and fix it as a root of the underlying tree T_H. An independent subset $F \subseteq E$ is called *flat* if every simple path from r in T_H intersects at most one $e \in F$. Flat subsets can be partially ordered by taking $F \prec F'$ if the sets $\bigcup F$ and $\bigcup F'$ of hypergraph vertices are disjoint and every simple path from r to $\bigcup F'$ intersects $\bigcup F$. Then a non-empty $E' \subseteq E$ is independent iff it can be partitioned into flat 'layers'

$$F_1 \prec F_2 \prec \cdots \prec F_m.$$

Indeed, any disjoint flat subsets $F_1, \dots, F_m$ with this property give rise to an independent subset E'. Conversely, if E' is independent, then we first take the set F_1 of all hyperedges from E' that are accessible from r via paths not intersecting any other $e \in E'$, then we take the set F_2 of all hyperedges from $E' \setminus F_1$ that are accessible from r via paths not intersecting any other $e \in E' \setminus F_1$, and so on. As the number of flat subsets is $O(\lvert H\rvert^{\ell})$ (each contains at most ℓ hyperedges), we obtain the following result for linear THGPs (proven in Appendix A.7) in the vein of the preceding results (see Table 1):

Theorem 5.8. *For any tree hypergraph H based on a tree with at most ℓ leaves, there is a monotone linear THGP that computes f_H and has size $O(\lvert H\rvert^{3\ell+1})$.*

Proposition 5.4 and Theorem 5.8 with $\lvert\mathcal{H}(\boldsymbol{Q})\rvert = O(\lvert\boldsymbol{q}\rvert^{\ell})$ give us the following:

Corollary 5.9. *For every tree-shaped OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ such that CQ $\boldsymbol{q}$ has ℓ leaves, there is a monotone linear THGP that computes $f^{\triangledown}_{\boldsymbol{Q}}$ and is of size $\lvert\boldsymbol{q}\rvert^{O(\ell^2)}$.*

Example 3.6 shows that the exponential bound on the size of the hypergraph $\mathcal{H}(\boldsymbol{Q})$ cannot be reduced, and Corollary 5.9 does not give us polynomial-size THGPs for tree-shaped OMQs; moreover, $f^{\triangledown}_{\boldsymbol{Q}}$ may have exponentially many variables. In Section 5.3, we devise a direct construction for THGPs computing $f^{\blacktriangledown}_{\boldsymbol{Q}}$ for OMQs with CQs of bounded treewidth and ontologies with the polynomial fundamental set property (PFSP).

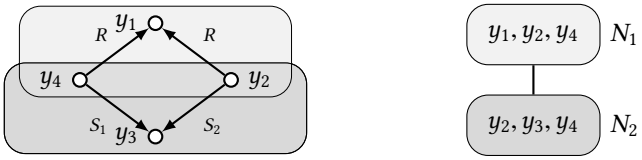


Fig. 10. Tree decomposition in Example 5.10.

5.3 THGPs for OMQs of Bounded Treewidth and PFSP

Recall (see, e.g., [32]) 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 is a node N with $v \in \lambda(N)$;
- for every $e \in E$, there is 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 5.10. The Boolean CQ $\boldsymbol{q} = \{ R(y_2, y_1),\ R(y_4, y_1),\ S_1(y_4, y_3),\ S_2(y_2, y_3) \}$ and its tree decomposition (T, λ) of width 2 are shown in Fig. 10, 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 $\lambda(N_2) = \{y_2, y_3, y_4\}$.

We show that, for any OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with a CQ 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 the Gaifman graph $G_{\boldsymbol{q}}$ of $\boldsymbol{q}$ of width $m - 1$. We fix an order of variables in every bag $\lambda(N)$ and define an 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 $\boldsymbol{w} \in \Omega_{\boldsymbol{Q}}^m$: its ith component $w_i \in \Omega_{\boldsymbol{Q}}$ indicates that the ith variable in the bag is mapped to a domain element aw_i in the canonical model $C_{\mathcal{T}, \mathcal{A}}$. We say that a type $\boldsymbol{w}$ is *compatible with a node* N if, for all $z, z' \in \lambda(N)$, the following conditions hold:

(C1) if $A(z) \in \boldsymbol{q}$ and $\boldsymbol{w}[z] \neq \varepsilon$, then $\boldsymbol{w}[z] = \sigma\varrho$ for some ϱ with $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$;
(C2) if $P(z, z') \in \boldsymbol{q}$ and either $\boldsymbol{w}[z] \neq \varepsilon$ or $\boldsymbol{w}[z'] \neq \varepsilon$, then
- $\boldsymbol{w}[z] = \boldsymbol{w}[z']$ and $\mathcal{T} \models P(x, x)$, or
- $\boldsymbol{w}[z'] = \boldsymbol{w}[z] \cdot \varrho$ or $\boldsymbol{w}[z] = \boldsymbol{w}[z'] \cdot \varrho^-$ for some ϱ with $\mathcal{T} \models \varrho(x, y) \to P(x, y)$,

where $\boldsymbol{w}[z]$ stands for $w_{\nu_N(z)}$. For a type $\boldsymbol{w}$ compatible with N, we use the abbreviation $\boldsymbol{w}[z]$ if N is clear from the context. Clearly, the type with all components equal to ε is compatible with any node N and corresponds to mapping the variables in $\lambda(N)$ to individual constants in $\mathsf{ind}(\mathcal{A})$.

Example 5.11. Let $\mathcal{T} = \{ A(x) \to \exists y\, R(x, y) \}$ and $\boldsymbol{q}$ be the same as in Example 5.10. Assume ν_{N_1} and ν_{N_2} respect the order of the variables in the bags in Fig. 10. The bag types compatible with N_1 are $(\varepsilon, \varepsilon, \varepsilon)$ and $(R, \varepsilon, \varepsilon)$, and only $(\varepsilon, \varepsilon, \varepsilon)$ is compatible with N_2.

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_H the tree obtained from T by replacing every edge $\{N_i, N_j\}$ with the following sequence of edges:

$$\{N_i, u^1_{ij}\}, \quad \{u^k_{ij}, v^k_{ij}\} \text{ and } \{v^k_{ij}, u^{k+1}_{ij}\}, \text{ for } 1 \leq k < M, \quad \{u^M_{ij}, v^M_{ij}\}, \quad \{v^M_{ij}, v^M_{ji}\},$$
$$\{v^M_{ji}, u^M_{ji}\}, \quad \{u^{k+1}_{ji}, v^k_{ji}\} \text{ and } \{v^k_{ji}, u^k_{ji}\}, \text{ for } 1 \leq k < M, \quad \{u^1_{ji}, N_j\},$$

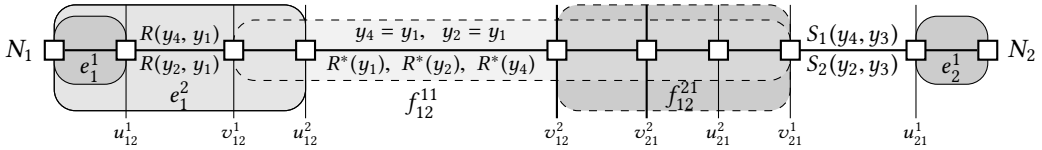


Fig. 11. THGP in Example 5.13: non-zero labels of hypergraph vertices are given on the edges of the tree.

for some fresh nodes u_{ij}^k, v_{ij}^k, u_{ji}^k and v_{ji}^k. We now define a *generalised* monotone THGP based on a hypergraph H with underlying tree T_H (in generalised THGPs, vertices are labelled with *conjunctions* of literals, which is convenient but does not add any expressive power; see Appendix A.3 for details). The hypergraph has the following hyperedges:

- $e_i^k = [N_i, u_{ij_1}^k, \dots, u_{ij_n}^k]$ (the minimal convex subtree with $N_i, u_{ij_1}^k, \dots, u_{ij_n}^k$, see Section 5.2) 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;
- $f_{ij}^{k\ell} = [v_{ij}^k, v_{ji}^\ell]$ 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 $\boldsymbol{w}_k[z] = \boldsymbol{w}_\ell[z]$, for all $z \in \lambda(N_i) \cap \lambda(N_j)$.

Each vertex $\{u_{ij}^k, v_{ij}^k\}$ in H is labelled with the conjunction of the following variables:

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

all other vertices are labelled with 0. The obtained generalised THGP can be equivalently represented as a linearly large THGP, and the following is proved in Appendix A.8:

Theorem 5.12. *For every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with a fundamental set Ω_Q and a CQ of treewidth t, there is a monotone THGP that computes $f_Q^{\blacktriangledown}$ and is of degree polynomial in $|\Omega_Q|^t$ and size polynomial in $|\boldsymbol{q}|$ and $|\Omega_Q|^t$.*

Example 5.13. Let $\boldsymbol{Q} = (\mathcal{T}, \boldsymbol{q})$ be the OMQ from Example 5.11. 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 shown in Fig. 11, where the omitted labels are 0. To explain the meaning of the THGP, let $\mathcal{T}, \mathcal{A} \models \boldsymbol{q}$, for some data instance $\mathcal{A}$. Then a homomorphism $h\colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T},\mathcal{A}}$ defines the type of 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 $a \in \mathsf{ind}(\mathcal{A})$). The two cases are represented by 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 for 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 any 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 rise to $f_{12}^{11} = [v_{12}^1, v_{21}^1]$ and $f_{12}^{21} = [v_{12}^2, v_{21}^1]$. Thus, if N_1 is of type $\boldsymbol{w}_1$, then we include e_1^1 and f_{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 we take f_{12}^{21}, and so $p_{y_4=y_1} \wedge p_{y_2=y_1} \wedge p_{R^*(y_1)} \wedge p_{R^*(y_2)} \wedge p_{R^*(y_4)}$ should be true. Since $\{v_{21}^1, u_{21}^1\}$ is not in any hyperedge, $p_{S_1(y_4,y_3)} \wedge p_{S_2(y_2,y_3)}$ should hold in either case.

For OMQs with ontologies of depth 1, a similar construction gives us the following:

Theorem 5.14. *For every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with an ontology of depth 1 and a CQ of treewidth t, there is a monotone THGP that computes $f_Q^{\triangledown}$ and is of degree $2^{O(t)}$ and size polynomial in $|\boldsymbol{q}|$ and 2^t.*

A crucial observation in the proof of Theorem 5.14 (see Appendix A.8) is that any OMQ with an ontology of depth 1 can equivalently be replaced by an *explicit* OMQ whose tree witnesses $\mathfrak{t}$ are

Table 2. HGPs computing tree-witness hypergraph functions for OMQs.

	OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$	monotone HGP	of size	computes
Cor. 5.2	$\mathcal{T}$ of depth 1	HGP of degree 2	$O(\lvert\boldsymbol{q}\rvert)$	$f^{\triangledown}_{\boldsymbol{Q}}$
Cor. 5.9	tree-shaped $\boldsymbol{q}$ with ℓ leaves	linear THGP	$\lvert\boldsymbol{q}\rvert^{O(\ell^2)}$	$f^{\triangledown}_{\boldsymbol{Q}}$
Cor. 5.7	tree-shaped $\boldsymbol{q}$ and $\mathcal{T}$ of depth 1	linear THGP of degree 2	$\lvert\boldsymbol{q}\rvert^{O(1)}$	$f^{\triangledown}_{\boldsymbol{Q}}$
Th. 5.12	$\boldsymbol{q}$ of treewidth t $\Omega_{\boldsymbol{Q}}$ a fundamental set	THGP	$\lvert\boldsymbol{q}\rvert^{O(1)} \cdot \lvert\Omega_{\boldsymbol{Q}}\rvert^{O(t)}$	$f^{\blacktriangledown}_{\boldsymbol{Q}}$
Th. 5.14	$\boldsymbol{q}$ of treewidth t and $\mathcal{T}$ of depth 1	THGP of degree $2^{O(t)}$	$\lvert\boldsymbol{q}\rvert^{O(1)} \cdot 2^{O(t)}$	$f^{\triangledown}_{\boldsymbol{Q}}$

initiated by special predicates $P_{\mathfrak{t}}$. As each tree witness $\mathfrak{t} = (\mathfrak{t}_{\mathsf{r}}, \mathfrak{t}_{\mathsf{i}})$ for an OMQ of depth 1 is uniquely determined by a variable z with $\mathfrak{t}_{\mathsf{i}} = \{z\}$, type compatibility for explicit OMQs can be refined: $\boldsymbol{w}$ is *strongly compatible with a node* N if (C1) and (C2) hold for all $z, z' \in \lambda(N)$, and, for all $z \in \lambda(N)$,

(C3) $\boldsymbol{w}[z]$ is either ε or $P_{\mathfrak{t}}$ for the tree witness $\mathfrak{t} = (\mathfrak{t}_{\mathsf{r}}, \mathfrak{t}_{\mathsf{i}})$ such that $\mathfrak{t}_{\mathsf{i}} = \{z\}$.

The binary choice in (C3) gives the $2^{O(t)}$ bound on the degree of the resulting THGP.

The obtained results on HGPs computing $f^{\triangledown}_{\boldsymbol{Q}}$ and $f^{\blacktriangledown}_{\boldsymbol{Q}}$ are summarised in Table 2 .

6 REPRESENTING HYPERGRAPHS AS OMQS

Going in the opposite direction, we represent every hypergraph H as a 'small' OMQ $\boldsymbol{Q}$ such that any monotone HGP based on H computes a subfunction of $f^{\triangle}_{\boldsymbol{Q}}$, and each hyperedge of H corresponds to a tree witness for $\boldsymbol{Q}$, making H a subgraph of $\mathcal{H}(\boldsymbol{Q})$. Here, H is a *subgraph* of a hypergraph $H' = (V', E')$ if H is obtained from H' by removing some of its hyperedges and some of its vertices from both V' and E'.

6.1 Arbitrary Hypergraphs as 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 $e \in E$:

$$B_e(x) \;\to\; \exists y \Big[\bigwedge_{e \cap e' \neq \emptyset,\ e \neq e'} R_{e'}(x, y) \;\wedge\; \bigwedge_{v \in e} A_v(y) \;\wedge\; \exists z\, R_e(z, y) \Big].$$

We also take the Boolean CQ $\boldsymbol{q}$ with variables y_v and z_e, for $v \in V$ and $e \in E$:

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

Example 6.1. Consider the hypergraph from Example 4.1, which we now denote by $H = (V, E)$. The CQ $\boldsymbol{q}$ and the canonical models $\mathcal{C}_{\mathcal{T}}^{B_{e_i}(a)}$, for $i = 1, 3$, are shown in Fig. 12 along with the five tree witnesses for $\boldsymbol{Q}_H$: each square vertex ◻ represents the variable y_{v_i} with a unary atom $A(y_{v_i})$ and the black arrows with a number i represent a binary atom in the query with the predicate R_{e_i}. The dashed grey arrows, which duplicate query edges, are shown only as an aid for identifying tree witnesses.

Observe that all the tree witnesses for $\boldsymbol{Q}_H$ fall into two types:

$\mathfrak{t}^e = (\mathfrak{t}^e_{\mathsf{r}}, \mathfrak{t}^e_{\mathsf{i}})$ with $\mathfrak{t}^e_{\mathsf{r}} = \{z_{e'} \mid e \cap e' \neq \emptyset, e \neq e'\}$ and $\mathfrak{t}^e_{\mathsf{i}} = \{z_e\} \cup \{y_v \mid v \in e\}$, for $e \in E$;

$\mathfrak{t}^v = (\mathfrak{t}^v_{\mathsf{r}}, \mathfrak{t}^v_{\mathsf{i}})$ with $\mathfrak{t}^v_{\mathsf{r}} = \{z_e \mid v \in e\}$ and $\mathfrak{t}^v_{\mathsf{i}} = \{y_v\}$, for $v \in V$ that belong to a single $e \in E$.

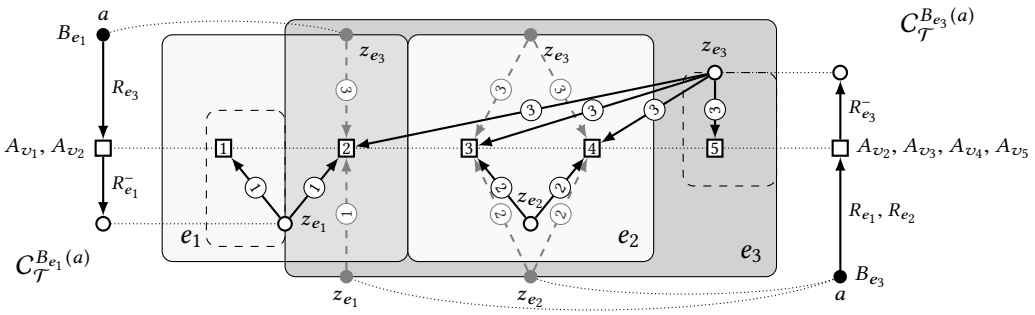


Fig. 12. The OMQ $\mathcal{Q}_H$ for H from Example 4.1, its hyperedge tree witnesses $\mathfrak{t}^{e_1}$, $\mathfrak{t}^{e_2}$, $\mathfrak{t}^{e_3}$ (solid rectangles), and vertex tree witnesses $\mathfrak{t}^{v_1}$ and $\mathfrak{t}^{v_5}$ (dashed rectangles). Dotted lines are homomorphisms for $\mathfrak{t}^{e_1}$ and $\mathfrak{t}^{e_3}$.

In Example 6.1, the tree witness $\mathfrak{t}^{e_1} = (\mathfrak{t}^{e_1}_{\mathrm{r}}, \mathfrak{t}^{e_1}_{\mathrm{i}})$ with $\mathfrak{t}^{e_1}_{\mathrm{r}} = \{z_{e_3}\}$ and $\mathfrak{t}^{e_1}_{\mathrm{i}} = \{z_{e_1}, y_{v_1}, y_{v_2}\}$ is generated by B_{e_1}—see the homomorphism on the left; however, as explained in Remark 2, we ignore the tree witnesses generated only by normalisation predicates, e.g., the tree witness $\mathfrak{t} = (\mathfrak{t}_{\mathrm{r}}, \mathfrak{t}_{\mathrm{i}})$ with $\mathfrak{t}_{\mathrm{r}} = \{y_{v_1}, y_{v_2}\}$ and $\mathfrak{t}_{\mathrm{i}} = \{z_{e_1}\}$ (the bottom half of $\mathfrak{t}^{e_1}$). The tree witness $\mathfrak{t}^{v_1} = (\mathfrak{t}^{v_1}_{\mathrm{r}}, \mathfrak{t}^{v_1}_{\mathrm{i}})$ with $\mathfrak{t}^{v_1}_{\mathrm{r}} = \{z_{e_1}\}$ and $\mathfrak{t}^{v_1}_{\mathrm{i}} = \{y_{v_1}\}$ is generated by B_{e_3}.

Theorem 6.2. *Any hypergraph H is isomorphic to a subgraph of $\mathcal{H}(\mathcal{Q}_H)$, and any monotone HGP based on H computes a subfunction of $f^{\wedge}_{\mathcal{Q}_H}$.*

Proof. An isomorphism between $H = (V, E)$ and a subgraph of $\mathcal{H}(\mathcal{Q}_H)$ 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 $e \in E$.

Given an input $\boldsymbol{\alpha}$ for a monotone HGP P based on H, we define an assignment $\boldsymbol{\gamma}$ for the predicates in $\mathcal{Q}_H$ by taking $\boldsymbol{\gamma}(A_v)$ to be the value of the label of each $v \in V$ under $\boldsymbol{\alpha}$, and $\boldsymbol{\gamma}(B_e) = 1$ and $\boldsymbol{\gamma}(R_e) = 1$ for each $e \in E$ (of course, $\boldsymbol{\gamma}(P_\zeta) = 0$ for 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 $f^{\wedge}_{\mathcal{Q}_H}(\boldsymbol{\gamma}) = 1$. Suppose first that $P(\boldsymbol{\alpha}) = 1$, 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 $f^{\wedge}_{\mathcal{Q}_H}(\boldsymbol{\gamma}) = 1$.

Conversely, if $f^{\wedge}_{\mathcal{Q}_H}(\boldsymbol{\gamma}) = 1$, 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 subset $E' = \{e \in E \mid h(z_e) = w'_e\}$ 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 $A_v(y_v) \in \boldsymbol{q}$. □

6.2 Hypergraphs of Degree 2 as OMQs with Ontologies of Depth 1

We show now that any hypergraph H of degree 2 is isomorphic to $\mathcal{H}(\mathcal{S}_H)$, for some OMQ $\mathcal{S}_H = (\mathcal{T}, \boldsymbol{q})$ 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 $v \in i_1(v)$, $v \in i_2(v)$ but $i_1(v) \neq i_2(v)$. For any $v \in V$, we fix a

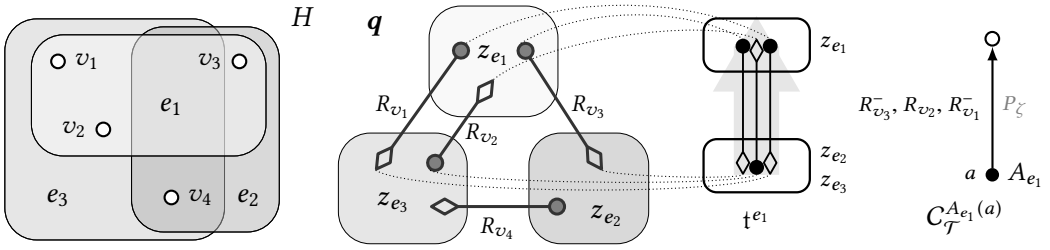


Fig. 13. Hypergraph H in Example 6.3, its CQ $\boldsymbol{q}$, tree witness $\mathfrak{t}^{e_1}$ for S_H and canonical model $\mathcal{C}^{A_{e_1}(a)}_{\mathcal{T}}$.

binary predicate R_v and, for any $e \in E$, a unary predicate A_e. Let the ontology $\mathcal{T}$ in S_H contain the following axioms, for $e \in E$:

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

The Boolean CQ $\boldsymbol{q}$ contains variables z_e, for $e \in E$, and is defined by taking

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

Example 6.3. Let $H = (V, E)$, where $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3\}$ and $e_1 = \{v_1, v_2, v_3\}$, $e_2 = \{v_3, v_4\}$, $e_3 = \{v_1, v_2, v_4\}$. Suppose

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

The hypergraph H and the query $\boldsymbol{q}$ are shown in Fig. 13: 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,

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

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

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

The canonical model $\mathcal{C}^{A_{e_1}(a)}_{\mathcal{T}}$ is shown on the right-hand side of Fig. 13. 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 $\mathcal{H}(S_H)$.

Theorem 6.4. *Any hypergraph $H = (V, E)$ of degree 2 is isomorphic to $\mathcal{H}(S_H)$, and any monotone HGP based on H computes a subfunction of $f^{\Delta}_{S_H}$.*

Proof. 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 $\mathcal{H}(S_H)$. By the definition of S_H, g is a bijection between V and the atoms of $\boldsymbol{q}$. For any $e \in E$, there is a tree witness $\mathfrak{t}^e = (\mathfrak{t}^e_{\mathrm{r}}, \mathfrak{t}^e_{\mathrm{i}})$ generated by A_e with $\mathfrak{t}^e_{\mathrm{i}} = \{z_e\}$ and $\mathfrak{t}^e_{\mathrm{r}} = \{z_{e'} \mid e \cap e' \neq \emptyset, e \neq e'\}$, and $\boldsymbol{q}_{\mathfrak{t}^e}$ consists of the $g(v)$, for $v \in e$. Conversely, every tree witness $\mathfrak{t} = (\mathfrak{t}_{\mathrm{r}}, \mathfrak{t}_{\mathrm{i}})$ for S_H contains $z_e \in \mathfrak{t}_{\mathrm{i}}$, for some $e \in E$, and so $\boldsymbol{q}_{\mathfrak{t}} = \{g(v) \mid v \in e\}$.

Given an input $\boldsymbol{\alpha}$ for a monotone HGP P based on H, we define $\boldsymbol{\gamma}$ by taking $\boldsymbol{\gamma}(R_v)$ to be the value of the label of v under $\boldsymbol{\alpha}$ for $v \in V$ and $\boldsymbol{\gamma}(A_e) = 1$ for $e \in E$ (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 $C_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$ contains a labelled null w_e such that

$$C_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})} \models \bigwedge_{v \in V \text{ with } i_1(v)=e} R_v(w_e, a) \quad \wedge \bigwedge_{v \in V \text{ with } i_2(v)=e} R_v(a, w_e).$$

We prove that $P(\boldsymbol{\alpha}) = 1$ iff $f^{\wedge}_{S_H}(\boldsymbol{\gamma}) = 1$. Suppose $P(\boldsymbol{\alpha}) = 1$, that is, there is an independent subset E' of E such that the label of each $v \in V \setminus V_{E'}$ evaluates to 1 under $\boldsymbol{\alpha}$. Define $h\colon \boldsymbol{q} \to C_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$ 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 $\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma}) \models \boldsymbol{q}$, whence $f^{\wedge}_{S_H}(\boldsymbol{\gamma}) = 1$.

Conversely, if $f^{\wedge}_{S_H}(\boldsymbol{\gamma}) = 1$, then there is a homomorphism $h\colon \boldsymbol{q} \to C_{\mathcal{T},\mathcal{A}(\boldsymbol{\gamma})}$. We show that the subset $E' = \{ e \in E \mid h(z_e) \neq a \}$ 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 E' makes $P(\boldsymbol{\alpha}) = 1$. Indeed, for each $v \in V \setminus V_{E'}$, the map h sends both ends of the R_v-atom to a, and so the value of the label of v under $\boldsymbol{\alpha}$ is 1. □

6.3 Tree Hypergraphs as Tree-Shaped OMQs

We show that any tree hypergraph $H = (U, V, E)$ is isomorphic to a subgraph of $\mathcal{H}(\boldsymbol{T}_H)$, for some tree-shaped OMQ $\boldsymbol{T}_H$. Suppose $U = \{1, \dots, n\}$, for $n > 1$, and 1 is a leaf of T_H. Let $T^* = (U, V^*)$ be the *directed* tree obtained from T_H by fixing 1 as the root and orienting the edges away from it. We associate with H a tree-shaped OMQ $\boldsymbol{T}_H = (\mathcal{T}, \boldsymbol{q})$, in which $\boldsymbol{q}$ is the Boolean CQ

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

where the z_i, for $i \in U$, and the y_{ij}, for $(i,j) \in V^*$, are variables for the nodes and the arcs of the directed tree, respectively. To define $\mathcal{T}$, suppose a hyperedge $e \in E$ induces a convex directed subtree $T_e = (U_e, V_e)$ of T^* with root $r_e \in U_e$ and leaves $L_e \subseteq U_e$. Denote by $\mathcal{T}$ the ontology that contains the following axiom, for each $e \in E$:

$$A_e(x) \;\to\; \exists y \Big[\bigwedge_{(r_e,j) \in V_e} R_{r_e j}(x, y) \;\wedge \bigwedge_{\substack{(i,j) \in V_e \\ j \in L_e}} S_{ij}(y, x) \;\wedge\; \exists z \Big(\bigwedge_{\substack{(i,j) \in V_e \\ i \neq r_e}} R_{ij}(z, y) \;\wedge \bigwedge_{\substack{(i,j) \in V_e \\ j \notin L_e}} S_{ij}(y, z) \Big) \Big].$$

Since T_e is convex, its root r_e has only one outgoing arc, (r_e, j), for some j, and so the first conjunct of the axiom contains a single atom, $R_{r_e j}(x, y)$. Since the boundary of e comprises $\{r_e\} \cup L_e$ and the interior all other elements of U_e, these axioms (and convexity of hyperedges) ensure that $\boldsymbol{T}_H$ has a tree witness $\mathfrak{t}^e = (\mathfrak{t}^e_{\mathrm{r}}, \mathfrak{t}^e_{\mathrm{i}})$ with

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

Example 6.5. Let H be the tree hypergraph from Example 5.3 with root 1 and one hyperedge $e = [1, 4]$; see Fig. 8. The CQ $\boldsymbol{q}$ and the canonical model $\mathcal{C}^{A_e(a)}_{\mathcal{T}}$ for $\boldsymbol{T}_H$ are shown in Fig. 14. Note the tree witness $\mathfrak{t}^e$ and the homomorphism from $\boldsymbol{q}_{\mathfrak{t}^e}$ into $\mathcal{C}^{A_e(a)}_{\mathcal{T}}$.

The following THGP analogue of Theorem 6.2 is proved in Appendix A.9:

Theorem 6.6. *Any tree hypergraph H is isomorphic to a subgraph of $\mathcal{H}(\boldsymbol{T}_H)$, and any monotone THGP based on H computes a subfunction of $f^{\wedge}_{T_H}$.*

Table 3 summarises the representation results of Theorems 6.2, 6.4 and 6.6 that show how abstract hypergraphs can be embedded into tree-witness hypergraphs of polynomial-size OMQs; see Appendix A.10 for details on the size of the OMQs.

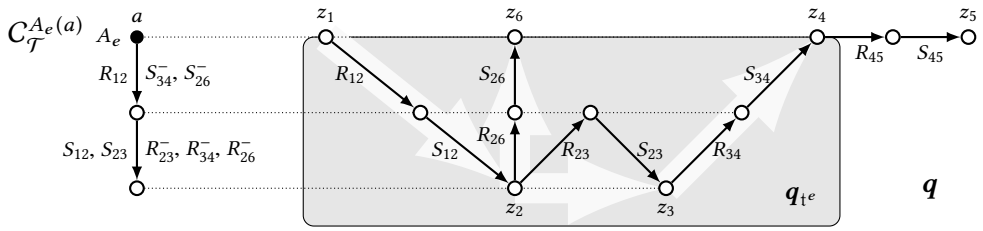


Fig. 14. The canonical model $\mathcal{C}_{\mathcal{T}}^{A_e(a)}$ and the CQ $\boldsymbol{q}$ (y_{ij} is the half-way point between z_i and z_j) for the tree hypergraph H in Example 6.5.

Table 3. Hypergraph representation results: any monotone HGP based on H computes a subfunction of $f_Q^{\wedge}$.

	hypergraph H	is isomorphic to	OMQ Q	$\lvert\mathcal{H}(Q)\rvert$	OMQ class
Th. 6.2	any	a subgraph of $\mathcal{H}(Q)$	Q_H	$O(\lvert H^2\rvert)$	$\mathcal{T}$ of depth 2
Th. 6.4	of degree 2	$\mathcal{H}(Q)$	S_H	$\lvert H\rvert$	$\mathcal{T}$ of depth 1
Th. 6.6	tree hypergraph with ℓ leaves	a subgraph of $\mathcal{H}(Q)$	T_H	$O(\lvert H^2\rvert)$	$\mathcal{T}$ of depth 2 and tree-shaped $\boldsymbol{q}$ with ℓ leaves

7 HYPERGRAPH PROGRAMS AND CIRCUIT COMPLEXITY

In Section 5, 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. We remind the reader that the complexity classes we use in this section form the chain

$$\Pi_3 \subsetneq \mathsf{AC}^0 \subsetneq \mathsf{NC}^1 \subseteq \mathsf{NL/poly} \subseteq \mathsf{LogCFL/poly} \subseteq \mathsf{P/poly} \subseteq \mathsf{NP/poly}, \tag{12}$$

and that whether any of the non-strict inclusions is actually strict remains a major open problem in complexity theory; see, e.g., [5, 49]. All these classes are non-uniform in the sense that they are defined in terms of polynomial-size non-uniform sequences of Boolean circuits or programs of certain shape and depth. The suffix '/poly' comes from an alternative definition of 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 for every $n < \omega$. (It will always be clear from the 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 $|\boldsymbol{C}_n| \le p(n)$, 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 the Boolean functions computable by polynomial circuits, and NC^1 consists of functions computed by polynomial formulas (that is, circuits whose logic gates have 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

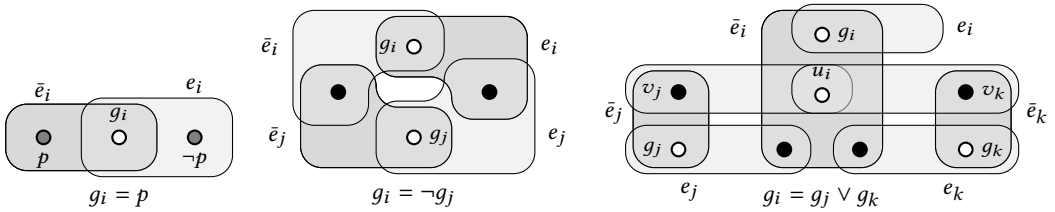


Fig. 15. HGP in the proof of Theorem 7.1: black vertices are labelled with 1 and white vertices with 0.

arbitrarily many inputs (*unbounded fan-in*) and NOT-gates can only be applied to inputs of the circuit [87]. 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. The class NL/poly consists of functions computed by polynomial-size nondeterministic branching programs (NBPs) to be defined in Section 7.2.

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 $C = \{C_{n+p(n)}\}_{n<\omega}$ such that, for any n and $\boldsymbol{\alpha} \in \{0,1\}^n$,

$$f_n(\boldsymbol{\alpha}) = 1 \quad \text{iff} \quad \text{there is } \boldsymbol{\beta} \in \{0,1\}^{p(n)} \text{ such that } C_{n+p(n)}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1 \tag{13}$$

(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 counterparts: 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 counterparts of the classes in (12) also form a chain [4, 51, 74]:

$$\mathsf{m}\Pi_3 \subsetneq \mathsf{mAC}^0 \subsetneq \mathsf{mNC}^1 \subsetneq \mathsf{mNL/poly} \subseteq \mathsf{mLogCFL/poly} \subsetneq \mathsf{mP/poly} \subsetneq \mathsf{mNP/poly}. \tag{14}$$

Whether the inclusion $\mathsf{mNL/poly} \subseteq \mathsf{mLogCFL/poly}$ is proper remains open.

7.1 NP/poly and HGP^3

Denote by HGP (mHGP) the class of Boolean functions computable by polynomial-size (monotone) HGPs; the functions computable by polynomial-size (monotone) HGPs of degree at most d comprise the class HGP^d (respectively, mHGP^d).

THEOREM 7.1. $\mathsf{NP/poly} = \mathsf{HGP} = \mathsf{HGP}^3$ *and* $\mathsf{mNP/poly} = \mathsf{mHGP} = \mathsf{mHGP}^3$.

PROOF. Suppose P is a (monotone) HGP. We construct a nondeterministic circuit C of size polynomial in $|P|$ whose input variables are the variables in P, certificate inputs correspond to the hyperedges of P and such that $C(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1$ 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

$$P(\boldsymbol{\alpha}) = 1 \quad \text{iff} \quad \text{there is } \boldsymbol{\beta} \text{ such that } C(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1. \tag{15}$$

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 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 C.

Conversely, let C be a circuit with certificate inputs. We construct an HGP P of degree at most 3 satisfying (15) as follows. For each gate g_i in 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. For each 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 e_j, and a vertex labelled with 1 to hyperedges $\bar{e}_i$ and $\bar{e}_j$;
- 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 the pattern is dual to the case of $\vee$: 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.

Then, 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 (15) 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 $\bar{e}_i$ in 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 HGP 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, we obtain $P(\boldsymbol{\alpha}) = 1$ and, clearly, $P'(\boldsymbol{\alpha}) = 1$. Conversely, suppose $P'(\boldsymbol{\alpha}) = 1$. Each of the vertices g_i in P' for the inputs of $\boldsymbol{C}$ is covered by either e_i or $\bar{e}_i$; so, let $\boldsymbol{\alpha}'$ be such that $\boldsymbol{\alpha}'(g_i) = 1$ if g_i is covered by e_i, and $\boldsymbol{\alpha}'(g_i) = 0$ if g_i is covered by $\bar{e}_i$. 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, $\boldsymbol{C}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = 1$. □

7.2 NL/poly and HGP²

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 (see [49] for more details) that an NBP B is a directed graph $G = (V, E)$, whose arcs are labelled with 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 the 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 $\boldsymbol{\alpha}$. An NBP B *computes* a Boolean function f in case $f(\boldsymbol{\alpha}) = 1$ 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 computable by polynomial-size monotone NBPs is denoted by co-mNL/poly.[12] We now show that NL/poly coincides with the classes HGP²

[12] Observe that the functions computable by monotone NBPs are monotone by definition, and so using the *dual* of f is the only natural choice for the definition of co-mNL/poly. Another (equivalent) option is to define monotone NBPs by restricting all labels either to positive or to negative (computing monotone and anti-monotone functions, respectively) and use the *negation* $\neg f(p_1, \dots, p_n)$, as done by Grigni and Sipser [41].

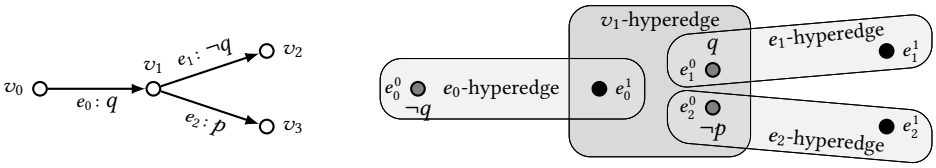


Fig. 16. HGP in the proof of Theorem 7.2: black vertices are labelled with 1.

and $\mathsf{HGP}^{=2}$ of functions computable by polynomial-size HGPs of degree at most 2 and exactly 2, respectively, while co-mNL/poly coincides with the classes mHGP^2 and $\mathsf{mHGP}^{=2}$ of functions computable by polynomial-size monotone HGPs of degree at most 2 and exactly 2.

Theorem 7.2. $\mathsf{NL/poly} = \mathsf{HGP}^2 = \mathsf{HGP}^{=2}$ *and* $\text{co-mNL/poly} = \mathsf{mHGP}^2 = \mathsf{mHGP}^{=2}$.

Proof. First, we prove $\mathsf{NL/poly} \subseteq \mathsf{HGP}^2$. One can show [46, 83] that 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 e leading to v, and all vertices e^0 for the arcs e leaving v. We also add to 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 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.

To show $\mathsf{HGP}^2 \subseteq \mathsf{HGP}^{=2}$, consider an HGP P of degree at most 2 computing f. We extend its hypergraph with three vertices, x, y and z, labelled with 1, 0 and 0, respectively, and three hyperedges $e_1 = \{v_1, \dots, v_l, x, y\}$, $e_2 = \{v_1, \dots, v_k, x, z\}$ and $e_3 = \{y, z\}$, where $v_1, \dots, v_k$ are the vertices of degree 0 and $v_{k+1}, \dots, v_l$ the vertices of degree 1. It is easy to see that each cover should contain e_3 but cannot contain e_1 and e_2. Indeed, y and z should both be covered. However, hyperedges e_1 and e_2 intersect and cannot be both in the same cover. Thus, y and z should be covered by e_3, while e_1 and e_2, intersecting e_3, are not in the cover. After these choices we are left with the original hypergraph.

To show $\mathsf{HGP}^{=2} \subseteq \mathsf{NL/poly}$, 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 [7]. For every hyperedge e_i, take a propositional variable u_i and associate the following set $\Phi_{\boldsymbol{\alpha}}$ of propositional binary clauses with every assignment $\boldsymbol{\alpha}$:

$$\neg u_i \vee \neg u_j, \ \text{ if } \ e_i \cap e_j \neq \emptyset, \qquad \text{and} \qquad u_i \vee u_j, \ \text{ if } \boldsymbol{\alpha}(v) = 0 \text{ for some } v \in e_i \cap e_j.$$

Informally, the former means that intersecting hyperedges cannot be chosen at the same time, while the latter that all zeros must be covered. By definition, E' is an independent set covering all zeros iff $E' = \{e_i \mid \boldsymbol{\gamma}(u_i) = 1\}$, 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

$$V = \big\{u_i, \bar{u}_i \mid 1 \leq i \leq k\big\}, \quad E_{\boldsymbol{\alpha}} = \big\{(u_i, \bar{u}_j) \mid e_i \cap e_j \neq \emptyset\big\} \cup \big\{(\bar{u}_i, u_j) \mid \boldsymbol{\alpha}(v) = 0, v \in e_i \cap e_j\big\}.$$

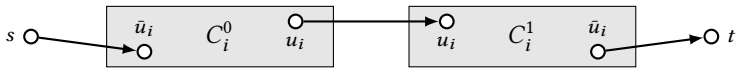


Fig. 17. The NBP in the proof of Theorem 7.2.

(V is the set of all 'literals' for the variables of $\Phi_{\boldsymbol{\alpha}}$, and $E_{\boldsymbol{\alpha}}$ are the arcs for the implicational form of the clauses: e.g., $\neg u_i \vee \neg u_j$ in $\Phi_{\boldsymbol{\alpha}}$ gives rise to two implications, $u_i \to \neg u_j$ and $u_j \to \neg u_i$, and thus to two arcs in $C_{\boldsymbol{\alpha}}$; similarly for $u_i \vee u_j$.) By [7, Theorem 1], $\Phi_{\boldsymbol{\alpha}}$ is satisfiable iff there is no (directed) cycle through some u_i and $\bar{u}_i$. We represent the $C_{\boldsymbol{\alpha}}$, for assignments $\boldsymbol{\alpha}$, as a single labelled directed graph C with

$$\text{vertices } u_i \text{ and } \bar{u}_i, \text{ for } 1 \le i \le k, \text{ and } w_{ij}^{\upsilon}, \text{ for } 1 \le i, j \le k \text{ and } \upsilon \in e_i \cap e_j, \text{ and with}$$
$$\text{arcs } (u_i, \bar{u}_j), \text{ for } e_i \cap e_j \neq \emptyset, \text{ and } (\bar{u}_i, w_{ij}^{\upsilon}) \text{ and } (w_{ij}^{\upsilon}, u_j), \text{ for each } \upsilon \text{ in } e_i \cap e_j \neq \emptyset;$$

arcs of the form $(u_i, \bar{u}_j)$ and $(\bar{u}_i, w_{ij}^{\upsilon})$ are labelled with 1 and arcs of the form (w_{ij}^{υ}, u_j) with the negation of the label of υ in P. It should be clear that $C_{\boldsymbol{\alpha}}$ has a cycle 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 $C_{\boldsymbol{\alpha}}$ contains a cycle through u_i and $\bar{u}_i$, for a hyperedge e_i in P. We thus have a polynomial NBP B computing $\neg f$, and so f is also computable by a polynomial NBP.

As to co-mNL/poly = $\mathsf{mHGP}^2 = \mathsf{mHGP}^{=2}$, observe that the first construction, if applied to a monotone NBP for f^*, produces a polynomial-size HGP of degree at most 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 preserves monotonicity. The third construction allows us to transform a monotone HGP of degree 2 for f into an NBP that computes $\neg f$ and has only negative literals. By inverting the polarity of the labels, we obtain a monotone NBP computing f^*. □

7.3 NL/poly and THGP(ℓ)

For any natural $\ell \ge 2$, 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 7.3. NL/poly = $\mathsf{THGP}(\ell)$ *and* mNL/poly = $\mathsf{mTHGP}(\ell)$, *for any* $\ell \ge 2$.

Proof. Suppose a polynomial-size THGP P computes a Boolean function f. Consider the function f_H for the underlying hypergraph H of P. By Theorem 5.8, f_H is computed by a polynomial-size monotone linear THGP P'. We can assume that the vertices $v_1, \dots, v_n$ of P' are consecutive edges of the path graph underlying P', and so every hyperedge in P' is of the form $[v_i, v_{i+m}] = \{v_i, \dots, v_{i+m}\}$, for some $m \ge 0$. We add two extra vertices, v_0 and v_{n+1}, to P' (thereby extending the underlying two-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 monotone linear THGP P'' computes f_H. 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 THGP P''. It is not hard to see that a path from s to t evaluated to 1 under a given assignment $\boldsymbol{\alpha}$ corresponds to a cover of zeros in P'' under $\boldsymbol{\alpha}$. 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 consecutive arcs labelled with $\boldsymbol{l}_{i_1}, \dots, \boldsymbol{l}_{i_k}$. Finally, we replace the vertex variables p_v in the labels by the corresponding vertex labels in P and fix all edge variables p_e to 1. The resulting NBP B is as required. Finally, observe that if P is monotone, then B is also monotone.

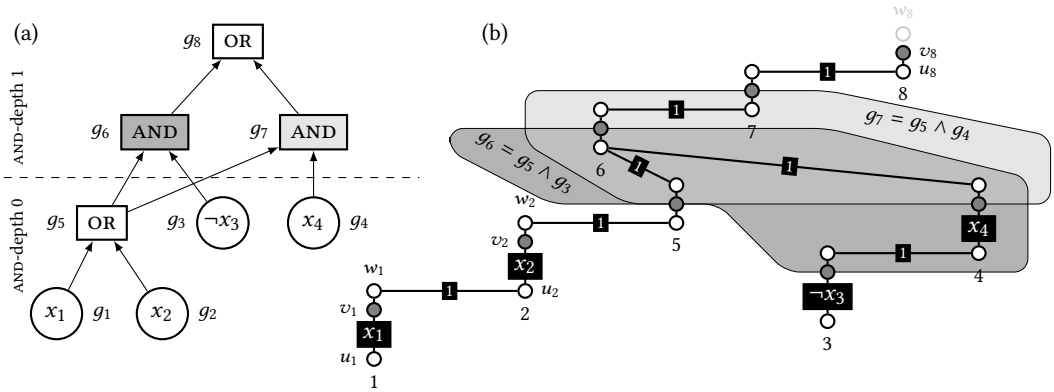


Fig. 18. (a) A circuit $\boldsymbol{C}$. (b) The labelled tree T for $\boldsymbol{C}$: the nodes in the ith triple are u_i, v_i, w_i and the omitted edge labels are zeros. The vertices of the THGP are the edges of T (with the same labels) and the hyperedges are sets of edges of T (two of them are shown).

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 B has 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 linear THGP computing f. The nodes of its underlying tree T are arranged into n vertex blocks $VB^1, \dots, VB^n$ and $n - 1$ edge blocks $EB^1, \dots, EB^{n-1}$. Each vertex block contains two nodes, $v_j^k, \bar{v}_j^k$, for each vertex $v_j \in V$, and each edge block contains two nodes, $e_i^k, \bar{e}_i^k$, for each edge $e_i \in E$:

$$VB^k = v_1^k, \bar{v}_1^k, v_2^k, \bar{v}_2^k, \dots, v_n^k, \bar{v}_n^k \qquad \text{and} \qquad EB^k = e_1^k, \bar{e}_1^k, e_2^k, \bar{e}_2^k, \dots, e_m^k, \bar{e}_m^k.$$

To construct the undirected tree T, we alternate the vertex and edge blocks in the following way: $VB^1, EB^1, VB^2, EB^2, \dots, EB^{n-1}, VB^n$ and then connect the adjacent nodes by edges. Consider now a tree hypergraph H based on T whose hyperedges are of the form

$$h_i^k = [\bar{v}_j^k, e_i^k] \quad \text{and} \quad g_i^k = [\bar{e}_i^k, v_{j'}^{k+1}], \qquad \text{for } e_i = (v_j, v_{j'}) \in E \text{ and } 1 \leq k < n.$$

Note that $|H| = O(|B|^2)$. The vertices of H of the form $\{v_i^k, \bar{v}_i^k\}$ are labelled with 1, the vertices of the form $\{e_i^k, \bar{e}_i^k\}$, which separate the hyperedges h_i^k and g_i^k, are labelled with the label of e_i in B, and all other vertices with 0. So, P is monotone whenever B is. We show 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 covering 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 the 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 $\boldsymbol{\alpha}$. □

7.4 LogCFL/poly and THGP

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

Theorem 7.4. LogCFL/poly = THGP *and* mLogCFL/poly = mTHGP.

Proof. To show that 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 $\boldsymbol{C}_n^{\mathsf{left}}$ and $\boldsymbol{C}_n^{\mathsf{right}}$ be the unions of sub-circuits computing the left and, respectively, right inputs of the AND-gates of AND-depth n. Without loss of generality (see Lemma A.6 in Appendix A.11) we can assume that $\boldsymbol{C}_n^{\mathsf{left}} \cap \boldsymbol{C}_n^{\mathsf{right}} = \emptyset$, for any $n \leq d$. 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 nodes u_i, v_i, w_i and arranging them into a tree as shown in Fig. 18. More precisely, we first arrange the nodes associated with the gates of the maximal AND-depth, d, 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 $\boldsymbol{C}_d^{\mathsf{left}}$ and the other with $\boldsymbol{C}_d^{\mathsf{right}}$, and so forth. We obtain the tree T by removing the node w_m from the result, where g_m is the output gate of $\boldsymbol{C}$ ($m = |\boldsymbol{C}|$); the tree T has v_m as its root and contains $3|\boldsymbol{C}| - 1$ nodes. The THGP P is based on the hypergraph whose vertices are the edges of T and whose hyperedges comprise the following (see Fig. 18):

- $[u_i, w_i]$, for each $i < m$ (pairs of edges in each triple of nodes 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 $g_i = g_{j_1} \vee \dots \vee g_{j_k}$.

Finally, if an input gate g_i is a literal $\boldsymbol{l}$, then we label the vertex $\{u_i, v_i\}$ with $\boldsymbol{l}$; we label all other $\{u_i, v_i\}$- and $\{v_i, w_i\}$-vertices with 0, and the remaining ones with 1. Clearly, the size of P is polynomial in $|\boldsymbol{C}|$. By Lemma A.7, for any input $\boldsymbol{\alpha}$, the output of g_i is 1 iff there is an independent set of hyperedges entirely inside the subtree rooted in v_i such that it covers all zeros in the subtree. Thus, P computes the same function as $\boldsymbol{C}$.

To show THGP $\subseteq$ LogCFL/poly, suppose a THGP P is based on a tree hypergraph $H = (U, V, E)$. Given an input $\boldsymbol{\alpha}$ for P and a non-empty subtree T of the underlying tree T_H of H, we set cover_T true iff there exists an independent subset of hyperedges in H that lie in T and cover all zeros in T. It follows that, for any edge v of T_H, $\mathsf{cover}_{(v,\{v\})}$ is true if $\{v\}$ is a hyperedge of H; otherwise, it is the value of the label of v in P under $\boldsymbol{\alpha}$.

Our aim is to recursively construct a polynomial-size SAC^1-circuit $\boldsymbol{C}$ computing the function cover_{T_H}. Let T be a convex subtree of T_H. Given a convex subtree T_0 of T, we define the T_0-*splitting* of T as a (uniquely determined) set $\{T_1, \dots, T_k\}$ of maximal convex subtrees such that the set of edges of T is a disjoint union of the non-empty sets of edges of $T_0, T_1, \dots, T_k$ and each T_i shares a leaf with T_0, in which case the conjunction $\mathsf{cover}_{T_1} \wedge \dots \wedge \mathsf{cover}_{T_k}$ is denoted by split_{T,T_0}. We also say that a node u of T *splits* T into the $(\{u\}, \emptyset)$-splitting of T. Observe that cover_T is equivalent to

$$\mathsf{split}_{T,(\{u\},\emptyset)} \quad \vee \quad \bigvee_{e \in E \text{ and } u \text{ is in the interior of } T_e} \mathsf{split}_{T,T_e}, \tag{16}$$

for any node u in T. By the *degree* $\deg(T)$ of T we understand the number of its leaves that are not leaves of T_H; in other words, the degree is the number of nodes shared with other subtrees of a splitting of T_H. Note that T_H is its only convex subtree of degree 0. The following lemma shows that to compute cover_{T_H} we only need subtrees of degree at most 2 and that the depth of recursion is $O(\log |P|)$.

Lemma 7.5. *Let T be a subtree of T_H with m edges and $\deg(T) \leq 2$. If $\deg(T) \leq 1$, then there is a node u splitting T into subtrees with at most $(m+1)/2$ edges and degree at most 2. If $\deg(T) = 2$, then there is a node u splitting T into subtrees with at most $(m+1)/2$ edges and degree at most 2 and, possibly, one subtree with less than m edges and degree 1.*

Proof. If $\deg(T) \leq 1$, then let a node u_1 split T into subtrees one of which, say T_1, has more than $(m+1)/2$ edges and all others have less than $(m-1)/2$ edges in total. Let u_2 be the (unique) node in T_1 adjacent to u_1 in T. Then u_2 splits T into subtrees lying inside T_1 and a subtree with the edge $\{u_1, u_2\}$, which has less than $(m+1)/2$ edges; all of the subtrees are of degree at most 2. If there is still a subtree with more than $(m+1)/2$ edges, then its size has decreased. The process is repeated until all subtrees have at most $(m+1)/2$ edges.

If $\deg(T) = 2$, then let b_1 and b_2 be the leaves of T that are not leaves of T_H. We proceed as above starting from $u_1 = b_1$, but stop when either the largest subtree has at most $(m+1)/2$ edges, or u_{i+1} leaves the path between b_1 and b_2, in which case u_i splits T into subtrees of degree at most 2 (and at most $(m+1)/2$ edges) and one subtree of degree 1 (whose number of edges does not exceed $m-1$). □

By applying (16) to T_H recursively and choosing the splitting nodes u as prescribed by Lemma 7.5, 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 at most 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 T is a subtree with m edges constructed on the recursion step i. To compute cover_T using (16), 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 $\log m + i$. Let $\{T_1, \dots, T_k\}$ be the $(\{u\}, \emptyset)$-splitting of T and m_j the number of edges in T_j, for all $j \leq k$. We have $m = m_1 + \dots + m_k$. By the induction hypothesis, we can compute each cover_{T_j} within the AND-depth of at most $\log m_j + i - 1$. Assign the probability m_j/m to T_j. As shown by Huffman [45], there is a prefix binary code such that each T_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 T_j so that the length of the branch ending at T_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 (16) whose depth does not exceed

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

The conjunction split_{T,T_e} is considered analogously. □

7.5 NC^1, Π_3 and THGP^d

Denote by THGP^d and mTHGP^d the classes of functions computable by polynomial-size THGPs and, respectively, monotone THGPs of degree at most d. The proof of the following theorem, given in Appendix A.11, is a simplified version of the proof of Theorem 7.4:

Theorem 7.6. $\mathsf{NC}^1 = \mathsf{THGP}^d$ *and* $\mathsf{mNC}^1 = \mathsf{mTHGP}^d$, *for any* $d \geq 3$.

The subclasses of THGP^2 and mTHGP^2 with *linear* HGPs are denoted, respectively, by $\mathsf{THGP}^2(2)$ and $\mathsf{mTHGP}^2(2)$. THGPs of degree 2 turn out to be less expressive than of degree 3:

Theorem 7.7. $\Pi_3 = \mathsf{THGP}^2 = \mathsf{THGP}^2(2)$ *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 based on a tree hypergraph H of degree at most 2. By Lemma A.3, $f_H(\boldsymbol{\alpha}) = 1$ iff neither $\boldsymbol{\alpha}$ is degenerate nor meets any of its polynomially many obstructions. This property can be computed by a Π_3-circuit where, for each obstruction $(e_0, \dots, e_{2n-1})$ from v_0 to v_{2n-1}, we create a circuit corresponding to the DNF expressing that one of the conditions (O1)–(O3) is falsified (see Section 5.2). We collect the outputs of the OR-gates of these circuits and input them to a fresh AND-gate. It remains to add to this AND-gate the conjunction of the labels of vertices of P that are not covered by any hyperedge; see condition (D).

To show $\Pi_3 \subseteq \mathsf{THGP}^2(2)$, let $\boldsymbol{C}$ be a Π_3-circuit. We can assume that $\boldsymbol{C}$ is a conjunction of DNFs. 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 the $h_{i,j}$

Table 4. Complexity classes, models of computation and the corresponding classes of HGPs.

complexity class	model of computation	theorem	class of HGPs
NP/poly	nondeterministic Boolean circuits	7.1	$\mathsf{HGP} = \mathsf{HGP}^d$, $d \geq 3$
P/poly	Boolean circuits		—
LogCFL/poly (SAC^1)	logarithmic-depth circuits with unbounded fan-in AND-gates, and NOT-gates only on inputs	7.4	THGP
NL/poly	nondeterministic branching programs	7.2 / 7.3	$\mathsf{HGP}^2 = \mathsf{THGP}(\ell)$, $\ell \geq 2$
NC^1	Boolean formulas	7.6	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 & output AND-gate	7.7	$\mathsf{THGP}^2 = \mathsf{THGP}^2(2)$

are AND-gates. We construct a generalised THGP P whose underlying path graph consists of k consecutive blocks of edges $V_1, \dots, V_k$, where each V_i comprises edges $v_{i,1}, \bar{v}_{i,1}, \dots, v_{i,l_i}, \bar{v}_{i,l_i}$, and whose hyperedges are of the form $\{v_{i,j}, \bar{v}_{i,j}\}$ and $\{\bar{v}_{i,j}, v_{i,j+1}\}$, for $j < l_i$. We label each $v_{i,j}$ with a conjunction of the inputs of $h_{i,j}$ and each $\bar{v}_{i,j}$ with 1. For an input for $\boldsymbol{C}$, we can cover all zeros among each V_i with an independent set of hyperedges iff at least one of the gates $h_{i,1}, \dots, h_{i,l_i}$ outputs 1; cf. Fig. 9. For different i, the corresponding V_i are covered independently. Thus, P computes the same function as $\boldsymbol{C}$. We convert P to a HGP from $\mathsf{THGP}^2(2)$ using Proposition A.2. □

Table 4 summarises the results obtained in this section. For example, its first row says that a function is computable by a polynomial-size nondeterministic circuit iff it can also be computed by a polynomial-size HGP (of degree at most d, for $d \geq 3$).

8 THE SIZE OF OMQ REWRITINGS

We now bring together the results from Sections 4–7 to obtain the upper and lower bounds on the size of PE-, NDL- and FO-rewritings in Fig. 2a. 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 of size polynomial in n, and by a rewriting Φ of $\boldsymbol{Q}$ we mean a sequence $\{\Phi_n\}_{n<\omega}$ of rewritings Φ_n of $\boldsymbol{Q}_n$. We call Φ a *polynomial rewriting of* $\boldsymbol{Q}$ if there is a polynomial p such that $|\Phi_n| \leq p(n)$, for $n < \omega$. Theorem 4.5 can now be recast in the complexity-theoretic setting as follows (see Table 4):

- if $\boldsymbol{Q}$ has a polynomial FO-rewriting, then $f^{\triangle}_{\boldsymbol{Q}} \in \mathsf{NC}^1$;
- if $\boldsymbol{Q}$ has a polynomial NDL-rewriting, then $f^{\triangle}_{\boldsymbol{Q}} \in \mathsf{mP/poly}$;
- if $\boldsymbol{Q}$ has a polynomial PE-rewriting, then $f^{\triangle}_{\boldsymbol{Q}} \in \mathsf{mNC}^1$.

Similarly, Theorems 4.2 and 4.4 give the 'converse' implications:

- if $f^{\triangledown}_{\boldsymbol{Q}} \in \mathsf{NC}^1$ or $f^{\blacktriangledown}_{\boldsymbol{Q}} \in \mathsf{NC}^1$, then $\boldsymbol{Q}$ has a polynomial FO-rewriting;
- if $f^{\triangledown}_{\boldsymbol{Q}} \in \mathsf{mP/poly}$ or $f^{\blacktriangledown}_{\boldsymbol{Q}} \in \mathsf{mP/poly}$, then $\boldsymbol{Q}$ has a polynomial NDL-rewriting;
- if $f^{\triangledown}_{\boldsymbol{Q}} \in \mathsf{mNC}^1$ or $f^{\blacktriangledown}_{\boldsymbol{Q}} \in \mathsf{mNC}^1$, then $\boldsymbol{Q}$ has a polynomial PE-rewriting.

In the sequel, we use these results without explicit references to the theorems.

The relations of the various classes of OMQs with the complexity of the Boolean functions associated with these OMQs and their equivalent classes of monotone HGPs are summarised

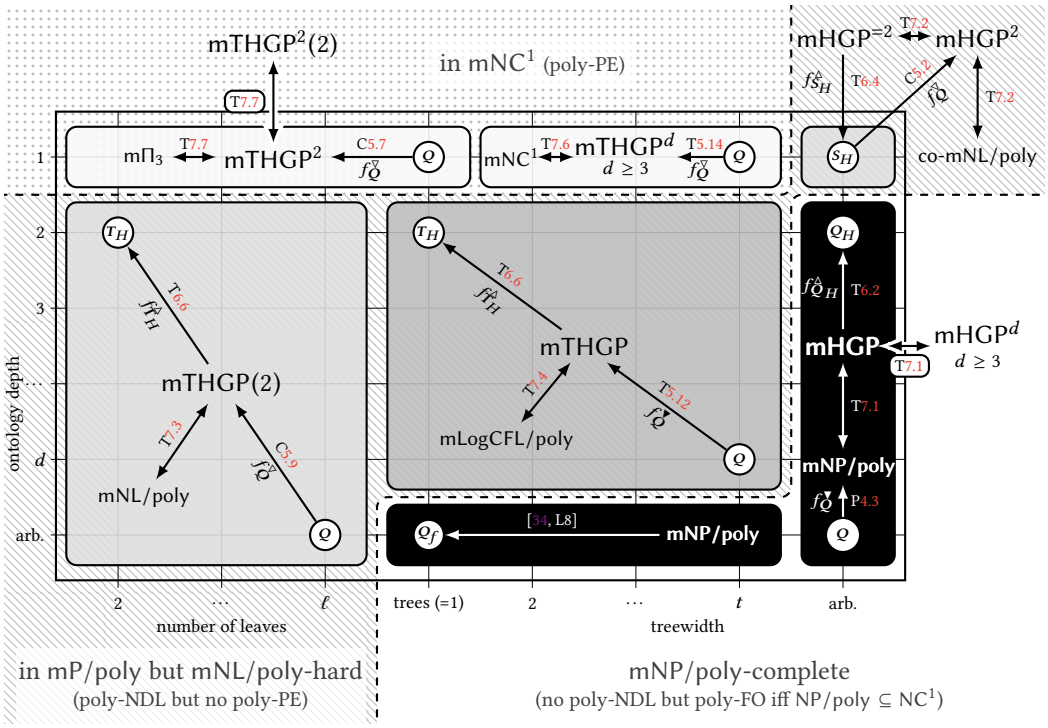


Fig. 19. Roadmap for the proofs of succinctness results.

in Fig. 19. The circles in the figure stand for classes of OMQs and the arrows for polynomial transformations. For example, if we take an OMQ $\boldsymbol{Q}$ with ontologies of unbounded depth and CQs of unbounded treewidth (the most general class, in the right bottom corner) then, by Proposition 4.3, $f^{\blacktriangledown}_{\boldsymbol{Q}} \in$ mNP/poly.[13] Conversely, by Theorem 7.1, any function in mNP/poly is computable by a polynomial-size monotone HGP based on some hypergraph H and, by Theorem 6.2, this function is a subfunction of $f^{\triangle}_{\boldsymbol{Q}_H}$, for a polynomial-size OMQ $\boldsymbol{Q}_H$, which is of depth 2 and unbounded treewidth—see the $\boldsymbol{Q}_H$ node in Fig. 19. Now, to obtain the succinctness landscape of Fig. 2a, it remains to recall suitable results from circuit complexity; we shall do this in the remainder of this section. That every function in mNP/poly is computable by some (polynomial-size) tree-shaped OMQ was shown by Gottlob et al. [34, Lemma 8].

8.1 All OMQs

We begin with the class of all OMQs. By Theorem 7.1, monotone HGPs can compute any function in mNP/poly, in particular, the function Clique with $n(n-1)/2$ variables $e_{jj'}$, $1 \leq j < j' \leq n$, 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. Therefore, by Theorem 6.2, Clique is a subfunction of $f^{\triangle}_{\boldsymbol{Q}_{\text{Clique}}}$ for some polynomial-size OMQ $\boldsymbol{Q}_{\text{Clique}}$ with ontologies of depth 2 and polynomially many tree witnesses.

[13] The class mNP/poly of functions computable by monotone Boolean circuits coincides with the class of *monotone functions* in NP/poly [41].

On the other hand, Clique is known to be NP/poly-complete under non-uniform AC^0-reductions.[14] A series of papers started by Razborov [74] gave an exponential lower bound for the size of monotone circuits computing Clique, namely, $2^{\Omega(\sqrt{k})}$ for $k \leq \frac{1}{4}(n/\log n)^{2/3}$ [4]. Thus,

$$\text{Clique} \notin \mathsf{mP/poly}. \tag{17}$$

For monotone formulas, an even better lower bound is known: $2^{\Omega(k)}$ for $k = 2n/3$ [73]. Using these complexity results, we can establish:

Theorem 8.1. (*i*) *There is an OMQ with ontologies of depth* 2 *and polynomially-many tree witnesses, any PE- and NDL-rewritings of which are of exponential size.*

(*ii*) *The following conditions are equivalent*:

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

Proof. (*i*) If $\boldsymbol{Q}_{\text{Clique}}$ had a polynomial PE- or NDL-rewriting, then Clique would be in mP/poly, contrary to (17).

(*b*) $\Rightarrow$ (*c*) If $\boldsymbol{Q}_{\text{Clique}}$ has a polynomial-size FO-rewriting, then Clique $\in \mathsf{NC}^1$. But since Clique is NP/poly-complete under non-uniform AC^0-reductions, we obtain $\mathsf{NP/poly} \subseteq \mathsf{NC}^1$.

(*c*) $\Rightarrow$ (*a*) By Proposition 4.3, we have $f^{\blacktriangledown}_{\boldsymbol{Q}} \in \mathsf{mNP/poly}$, for any OMQ $\boldsymbol{Q}$. Thus, if $\mathsf{NP/poly} \subseteq \mathsf{NC}^1$, then $f^{\blacktriangledown}_{\boldsymbol{Q}} \in \mathsf{NC}^1$, and so $\boldsymbol{Q}$ has a polynomial FO-rewriting. □

8.2 OMQs with Ontologies of Depth 1

Consider the monotone function Reach that takes the adjacency matrix of a directed graph G with two distinguished vertices, s and t, and returns 1 iff G contains a directed path from s to t. The function Reach and its dual, Reach*, are both NL/poly-complete under non-uniform AC^0-reductions [75]. It is known [49, 51] that Reach is computable by a polynomial-size monotone NBP, but any monotone Boolean formula for Reach is of size $n^{\Omega(\log n)}$. Thus,

$$\text{Reach} \in \mathsf{mNL/poly} \qquad \text{but} \qquad \text{Reach} \notin \mathsf{mNC}^1. \tag{18}$$

By observing that mNC^1 is closed under taking dual functions (that is, by swapping and and or), we also obtain:

$$\text{Reach}^* \in \mathsf{co\text{-}mNL/poly} \qquad \text{but} \qquad \text{Reach}^* \notin \mathsf{mNC}^1. \tag{19}$$

Then, by Theorems 7.2 and 6.4, we construct a polynomial-size OMQ $\boldsymbol{S}_{\text{Reach}^*}$ of depth 1 such that Reach* is a subfunction of $f^{\Delta}_{\boldsymbol{S}_{\text{Reach}^*}}$. Using (19), we now prove the following:

Theorem 8.2. (*i*) *All OMQs* $\boldsymbol{Q}$ *of depth* 1 *have polynomial NDL-rewritings.*

(*ii*) *There is an OMQ of depth* 1 *any PE-rewriting of which is of superpolynomial size.*

(*iii*) *All OMQs of depth* 1 *have polynomial FO-rewritings iff* $\mathsf{NL/poly} \subseteq \mathsf{NC}^1$.

Proof. (*i*) By Theorem 7.2 and Corollary 5.2, $f^{\nabla}_{\boldsymbol{Q}} \in \mathsf{co\text{-}mNL/poly} \subseteq \mathsf{mP/poly}$, and so there is a polynomial NDL-rewriting of $\boldsymbol{Q}$.

(*ii*) If $\boldsymbol{S}_{\text{Reach}^*}$ had a polynomial PE-rewriting, then Reach* would be in mNC^1, contrary to (19).

[14] It is a standard observation in computational complexity that all well-known complete problems in conventional complexity classes are complete under non-uniform AC^0-reductions [2, 3]. Also, once we consider non-uniform reductions, all problems that are complete for the uniform version of some complexity class are also complete for the non-uniform version. In particular, NP and NP/poly have the same complete problems under non-uniform AC^0-reductions. The same is true of NL and NL/poly and of LogCFL and LogCFL/poly. We use these observations throughout Section 8.

(*iii*) If NL/poly $\subseteq$ NC^1 and $\boldsymbol{Q}$ is an OMQ of depth 1, then, by Theorem 7.2 and Corollary 5.2, $f^{\vee}_{\boldsymbol{Q}} \in$ co-mNL/poly $\subseteq$ mNC$^1 \subseteq$ NC^1, and so there is a polynomial FO-rewriting of $\boldsymbol{Q}$. Conversely, if $\boldsymbol{S}_{\text{REACH}^*}$ has a polynomial FO-rewriting, then $f^{\wedge}_{\boldsymbol{S}_{\text{REACH}^*}} \in$ NC^1. Since REACH* is NL/poly-complete under AC^0-reductions, we obtain NL/poly $\subseteq$ NC^1. □

8.3 Tree-Shaped OMQs with a Bounded Number of Leaves

By using (18) in conjunction with Theorems 7.3, 6.6 and Corollary 5.9, we get the following result:

THEOREM 8.3. *Fix any $\ell \geq 2$.*

(*i*) *All tree-shaped OMQs $\boldsymbol{Q}$ with at most ℓ leaves have polynomial NDL-rewritings.*

(*ii*) *There is an OMQ with ontologies of depth 2 and linear CQs, any PE-rewriting of which is of superpolynomial size.*

(*iii*) *The following are equivalent*:

(*a*) *all tree-shaped OMQs with at most ℓ leaves have polynomial FO-rewritings*;
(*b*) *all OMQs with ontologies of depth 2 and linear CQs have polynomial FO-rewritings*;
(*c*) NL/poly $\subseteq$ NC^1.

8.4 OMQs with PFSP and Bounded Treewidth

Consider the Boolean function HCFL that corresponds to the hardest context-free language L_0 [40]: it takes a word in the alphabet of L_0 as input and returns 1 iff the word is in L_0. It is known that HCFL is LogCFL/poly-complete under non-uniform AC^0-reductions [47]. It is also not hard to define a monotone version mHCFL of this function that is also LogCFL/poly-complete. Indeed, if HCFL has n inputs $\boldsymbol{x} = x_1, \dots, x_n$, then we construct a monotone function mHCFL with $2n$ inputs $\boldsymbol{x}, \boldsymbol{y}$, where $\boldsymbol{y} = y_1, \dots, y_n$, by taking

$$\text{mHCFL}(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 0, & \text{if there is } i \text{ with } x_i = y_i = 0, \\ \text{HCFL}(\boldsymbol{x}), & \text{if } y_i = \neg x_i, \text{ for all } i, \\ 1, & \text{otherwise.} \end{cases}$$

Now, mHCFL together with Theorems 5.12, 7.4 and 6.6 give us the following:

THEOREM 8.4. *Fix any $t > 0$.*

(*i*) *All OMQs $\boldsymbol{Q}$ with the PFSP and CQs of treewidth at most t have polynomial NDL-rewritings.*

(*ii*) *The following are equivalent*:

(*a*) *all OMQs with the PFSP and CQs of treewidth at most t have polynomial FO-rewritings*;
(*b*) *all tree-shaped OMQs with ontologies of depth 2 have polynomial FO-rewritings*;
(*c*) LogCFL/poly $\subseteq$ NC^1.

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

COROLLARY 8.5. *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.[15] As concerns PE-rewritings for OMQs with CQs of bounded treewidth, Theorem 8.3

[15] A positive answer to this question [56] is based on a flawed proof.

sends a negative message if their ontologies are of depth at least 2. However, for ontologies of depth 1, we obtain the following positive result:

THEOREM 8.6. (*i*) *For any fixed $t > 0$, all OMQs $\boldsymbol{Q}$ with ontologies of depth* 1 *and CQs of treewidth at most t have polynomial PE-rewritings.*

(*ii*) *All tree-shaped OMQs $\boldsymbol{Q}$ of depth* 1 *have polynomial Π_4-PE rewritings.*

PROOF. (*i*) By Theorems 5.14 and 7.6, $f^{\forall}_{Q} \in \mathsf{mNC}^1$, and so there is a polynomial PE-rewriting.

(*ii*) By Corollary 5.7 and Theorem 7.7, $f^{\forall}_{Q} \in \mathsf{m}\Pi_3$. By a simple unravelling argument, $f^{\forall}_{Q}$ is computed by a polynomial monotone Boolean Π_3-formula. It remains to repeat the argument in the proof of Theorem 4.2 to obtain a polynomial Π_4-PE rewriting of $\boldsymbol{Q}$. □

9 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. 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 8.2 (*ii*) 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 $\mathsf{ind}(\mathcal{A})$, 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 $C_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(\boldsymbol{a})$ iff there exists a homomorphism from $\boldsymbol{q}(\boldsymbol{a})$ to $C_{\mathcal{T},\mathcal{A}}$.

The combined complexity of CQ evaluation has been thoroughly investigated in relational database theory. In general, evaluating CQs is NP-complete [24], but the problem becomes tractable for tree-shaped CQs [88] and bounded treewidth CQs [25, 42]—LogCFL-complete, to be more precise [36].

The emerging combined complexity landscape for OMQ answering is summarised in Fig. 2b 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. [23] and Artale et al. [6], while the matching lower bound for tree-shaped OMQs by Kikot et al. [56] and Gottlob et al. [34]. 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.

9.1 OMQs with Bounded-Depth Ontologies

We begin by showing that the LogCFL upper bound for CQs of bounded treewidth [36] is preserved even in the presence of ontologies of bounded depth.

THEOREM 9.1. *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, $C_{\mathcal{T},\mathcal{A}}$ is finite for any $\mathcal{A}$. As LogCFL is closed under $\mathsf{L}^{\mathsf{LogCFL}}$-reductions [35] and evaluation of CQs of bounded treewidth is LogCFL-complete, it suffices to show that $C_{\mathcal{T},\mathcal{A}}$ 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 $aw \in \Delta^{C_{\mathcal{T},\mathcal{A}}}$ (since $|w| \leq d$ and d is fixed). Finally, as entailment in *OWL 2 QL* is in NL [6], each of the following problems can be decided by making a call to an NL (hence LogCFL) oracle:

- decide whether $a\varrho_1 \dots \varrho_n \in \Delta^{C_{\mathcal{T},\mathcal{A}}}$, for any $n \leq d$ and roles ϱ_i from $\mathcal{T}$;
- decide whether $u \in \Delta^{C_{\mathcal{T},\mathcal{A}}}$ belongs to $A^{C_{\mathcal{T},\mathcal{A}}}$, for a unary predicate A from $\mathcal{T}$ and $\mathcal{A}$;
- decide whether $(u_1, u_2) \in \Delta^{C_{\mathcal{T},\mathcal{A}}} \times \Delta^{C_{\mathcal{T},\mathcal{A}}}$ is in $P^{C_{\mathcal{T},\mathcal{A}}}$, 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 improved to NL:

Theorem 9.2. *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 nondeterministic 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 $\boldsymbol{q}(\boldsymbol{a})$ to $C_{\mathcal{T},\mathcal{A}}$ 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 $aw \in \Delta^{C_{\mathcal{T},\mathcal{A}}}$ with $|w| \leq d$. 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 fully materialised. Instead, we assume that the sequence u is guessed element-by-element and the condition $u \in U$ is verified along the sequence of guesses. We first ensure 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 we verify[16] that u_0 satisfies all atoms in $\boldsymbol{q}(\boldsymbol{x})$ that involve only z_0. 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 a child z' is compatible with (*i*) the binary atoms linking z to z' and (*ii*) the atoms that involve only z'. If the checks succeed, then 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, that is, when an element of $C_{\mathcal{T},\mathcal{A}}$ has been assigned to each variable in $\boldsymbol{q}(\boldsymbol{x})$.

Correctness and termination of the algorithm are straightforward. Membership in NL 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, as noted in the proof of Theorem 9.1, the checks $u \in A^{C_{\mathcal{T},\mathcal{A}}}$ and $(u, u') \in P^{C_{\mathcal{T},\mathcal{A}}}$ can be implemented in NL. □

9.2 OMQs with Bounded-Leaf CQs

It remains to settle the complexity of answering OMQs with arbitrary ontologies and tree-shaped CQs with a bounded number of leaves, for which neither the upper bounds from Section 9.1 nor the NP lower bound by Kikot et al. [56] are applicable.

Theorem 9.3. *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 nondeterministic auxiliary pushdown automata (NAuxPDAs). An NAuxPDA [29] is a nondeterministic Turing machine with an additional work tape constrained to operate as a pushdown store. Sudborough [82] showed that LogCFL coincides with the class of problems that can be solved by

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

ALGORITHM 1: Nondeterministic 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 $U = \left\{aw \in \Delta^{\mathcal{C}_{\mathcal{T},\mathcal{A}}} \mid a \in \mathsf{ind}(\mathcal{A}) \text{ and } |w| \leq d\right\}$; `/* not computed */`
guess $u_0 \in U$; `/* use the definition of U to check if the guess is allowed */`
if z_0 *is the* i^{th} *answer variable* **and** $u_0 \neq a_i$ **then**
  return false
check $u_0 \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, *for all* $A(z_0) \in \boldsymbol{q}$, **and** $(u_0, u_0) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, *for all* $P(z_0, z_0) \in \boldsymbol{q}$;
frontier $\longleftarrow \{z_0 \mapsto u_0\}$;
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 definition of U to check if the guess is allowed */`
    if z' *is the* i^{th} *answer variable* **and** $u' \neq a_i$ **then**
      return false
    check $(u, u') \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$, *for all* $P(z, z') \in \boldsymbol{q}$;
    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* $P(z', z') \in \boldsymbol{q}$;
    frontier $\longleftarrow$ frontier $\cup \{z' \mapsto u'\}$
return true;

NAuxPDAs running in logarithmic space and polynomial time (note that the pushdown tape is not subject to the logarithmic space bound). Algorithm 2 gives 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 T and iteratively construct a homomorphism from $\boldsymbol{q}(\boldsymbol{a})$ 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 that the call top(stack) on empty stack returns ε. During the execution of BLQuery, the height of the stack will never exceed $2|\mathcal{T}| + |\boldsymbol{q}|$ (which is enough to find a homomorphism if one exists [6]), 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 canMap(z_0, a_0, top(stack)). If the check succeeds, then 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, does not 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,

ALGORITHM 2: Nondeterministic 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 $a_0 \in \mathsf{ind}(\mathcal{A})$ **and** $n_0 < 2|\mathcal{T}| + |\boldsymbol{q}|$; `/* guess the ABox element and max distance */`
foreach *n in* $1, \dots, n_0$ **do** `/* guess the initial element in a step-by-step fashion */`
  guess *a role* ϱ *in* $\mathcal{T}$ **such that** `isGenerated(`ϱ, a_0, `top(stack))`;
  push ϱ on stack
check `canMap(`z_0, a_0, `top(stack))`;
frontier $\longleftarrow \left\{(z_0 \mapsto (a_0, |\mathsf{stack}|), z_i) \mid z_i \text{ is a child of } z_0 \text{ in } T\right\}$;
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** `canMap(`$z'$, a', ε`)`;
    frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a', 0), z_i') \mid z_i' \text{ is a child of } z' \text{ in } T\}$
  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* ϱ *in* $\mathcal{T}$ **such that** `isGenerated(`ϱ, a, `top(stack))`;
    push ϱ on stack;
    check $\mathcal{T} \models \varrho(x, y) \to P(x, y)$, *for all* $P(z, z') \in \boldsymbol{q}$, **and** `canMap(`$z'$, a, `top(stack))`;
    frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a, |\mathsf{stack}|), z_i') \mid z_i' \text{ is a child of } z' \text{ in } T\}$
  else if *Option 3* **and** $|\mathsf{stack}| > 0$ **then** `/* a step backward in the tree part */`
    let deepest $= \{(z \mapsto (a, n), z') \in \text{frontier} \mid n = |\mathsf{stack}|\}$; `/* may be empty */`
    remove all deepest from frontier;
    pop ϱ 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** `canMap(`$z'$, a, `top(stack))`;
      frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a, |\mathsf{stack}|), z_i') \mid z_i' \text{ is a child of } z' \text{ in } T\}$
  else if *Option 4* **and** $|\mathsf{stack}| > 0$ **then** `/* 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** `canMap(`$z'$, a, `top(stack))`;
    frontier $\longleftarrow$ frontier $\cup \{(z' \mapsto (a, |\mathsf{stack}|), z_i') \mid z_i' \text{ is a child of } z' \text{ in } T\}$
  else return false;
return true;

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

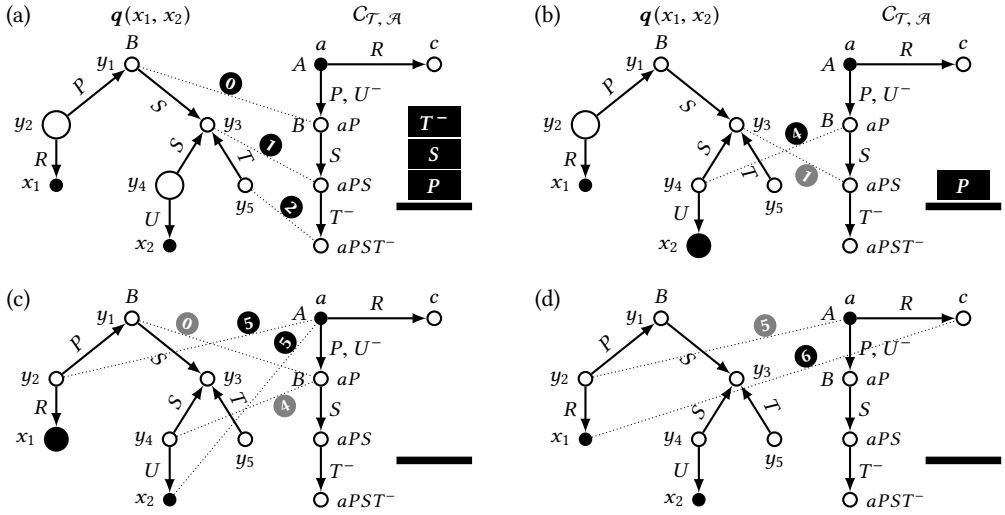


Fig. 20. Partial homomorphisms from a tree-shaped CQ $\boldsymbol{q}(x_1, x_2)$ to the canonical model $C_{\mathcal{T},\mathcal{A}}$ and the contents of stack in Example 9.4: (a) before the third iteration, (b) before the fifth iteration, (c) before and (d) after the final (sixth) iteration. Large nodes indicate the last component of the tuples in frontier.

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 $C_{\mathcal{T},\mathcal{A}}$, we can define a successful execution of BLQuery. We prove in Appendix A.12 that BLQuery terminates (Proposition A.8), is correct (Proposition A.9) and can be implemented by an NAuxPDA (Proposition A.10). The following example illustrates the construction.

Example 9.4. Suppose $\mathcal{T}$ has the following axioms:

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

the query $\boldsymbol{q}(x_1, x_2)$ consists of the following atoms with quantified variables $y_1, \ldots, y_5$:

$$R(y_2, x_1),\ \ P(y_2, y_1),\ \ S(y_1, y_3),\ \ T(y_5, y_3),\ \ S(y_4, y_3),\ \ U(y_4, x_2)$$

and $\mathcal{A} = \{\, A(a),\ R(a, c)\,\}$. Observe that $C_{\mathcal{T},\mathcal{A}} \models \boldsymbol{q}(c, a)$. We show how to define an execution of BLQuery that returns true on $((\mathcal{T}, \boldsymbol{q}), \mathcal{A}, (c, a))$ 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 $\mathsf{canMap}(y_1, a, P)$ 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 the first tuple in frontier and Option 2 for the second. 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,

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

at the start of the second iteration of the loop. We are going to use Option 3 for the second tuple in frontier and Option 2 for the third. 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 (see Fig. 20a),

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

We apply Option 3 and, since deepest $= \emptyset$, we pop T^- from stack but make no other changes. In the fourth iteration, we again apply Option 3. Since deepest $= \{(y_3 \mapsto (a,2), y_4)\}$, we remove this tuple from frontier and pop S from stack. Since the checks succeed for S, we add $(y_4 \mapsto (a,1), x_2)$ to frontier. The fifth iteration (see Fig. 20b) begins with

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

We apply Option 3 with 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. Thus,

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

at the start of the final, sixth, iteration; see Fig. 20c. 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. 20d. 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 [86]. This problem was used by Gottlob et al. [36] 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. [36], 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 [36, 86] that a circuit in normal form accepts an input $\boldsymbol{\alpha}$ iff there is a labelled rooted tree (called a *proof tree*) such that

- the root is labelled with the output AND-gate;
- if a node is labelled with an AND-gate $g_i = g_j \wedge g_k$, then it has two children labelled with g_j and g_k, respectively;
- if a node is labelled with an OR-gate $g_i = g_{j_1} \vee \ldots \vee g_{j_k}$, then it has a unique child that is labelled with one of $g_{j_1}, \ldots, g_{j_k}$;

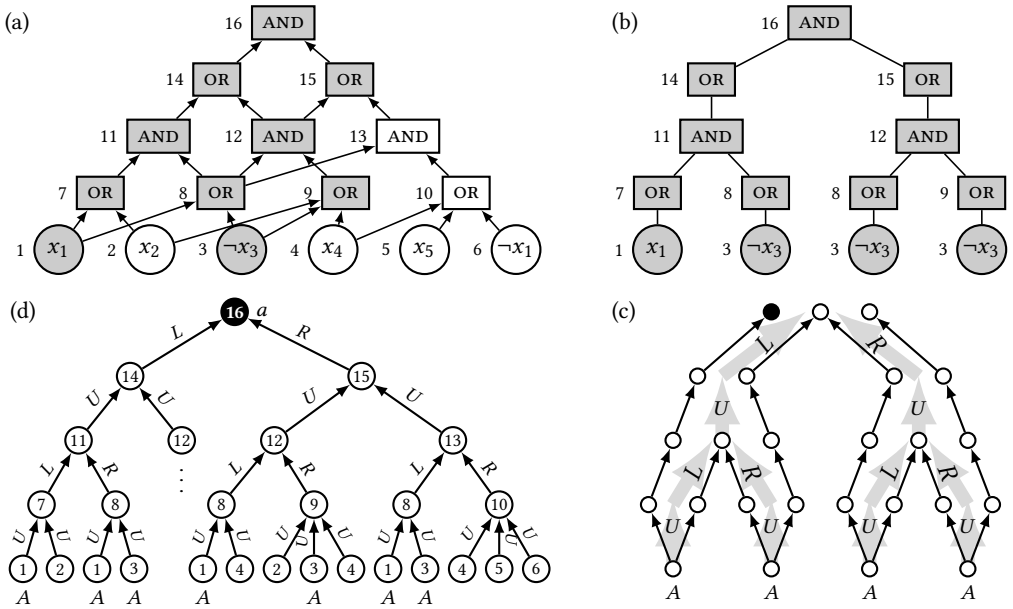


Fig. 21. (a) A circuit C of five levels with input $\boldsymbol{\alpha}$: $x_1 \mapsto 1$, $x_2 \mapsto 0$, $x_3 \mapsto 0$, $x_4 \mapsto 0$, $x_5 \mapsto 0$ (the gate number is indicated on the left and gates with value 1 under $\boldsymbol{\alpha}$ are shaded); (b) a proof tree for C and $\boldsymbol{\alpha}$; (c) CQs $\boldsymbol{q}$ (thick grey arrows) and $\boldsymbol{q}'$ (black arrows); (d) the canonical model of $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$ with the subscript of G_i inside the nodes.

– every leaf is labelled with an input gate whose literal evaluates to 1 under $\boldsymbol{\alpha}$.

For example, the circuit in Fig. 21a accepts $(1, 0, 0, 0, 0)$, as witnessed by the proof tree in Fig. 21b. While a circuit-input pair may admit multiple proof trees, they are all isomorphic modulo the labelling. Thus, with every circuit C, we can associate a *skeleton proof tree* T such that C accepts $\boldsymbol{\alpha}$ iff some labelling of T is a proof tree for C and $\boldsymbol{\alpha}$. Note that T depends only on the number of levels in C. The reduction [36], which is for presentation purposes reproduced here with minor modifications, encodes 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 C as constants and consists of the following facts:

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

The CQ $\boldsymbol{q}$ uses the nodes of T as variables, has an atom $U(z_j, z_i)$ ($L(z_j, z_i)$ or $R(z_j, z_i)$, respectively) for every node z_i with unique (left or right, respectively) child z_j, and has an atom $A(z_i)$ for every leaf z_i. These definitions guarantee that $D(\boldsymbol{\alpha}) \models \boldsymbol{q}$ iff 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:

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

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

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

see Fig. 21c. We now define $(\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A})$. Suppose $\boldsymbol{C}$ has gates $g_1, \dots, g_m$, with the output gate g_m. In addition to predicates U, L, R, A, we take a unary predicate G_i for each gate g_i. We set $\mathcal{A} = \{G_m(a)\}$ and include the following axioms in $\mathcal{T}_{\boldsymbol{\alpha}}$:

$$\begin{aligned} &G_i(x) \to \exists y \left(S(y, x) \wedge G_j(y) \right), && \text{for every } S(g_j, g_i) \in D(\boldsymbol{\alpha}),\ S \in \{U, L, R\}, \\ &G_i(x) \to A(x), && \text{for every } A(g_i) \in D(\boldsymbol{\alpha}); \end{aligned}$$

see Fig. 21d for an illustration. When restricted to predicates U, L, R and 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 A.13 that $\mathcal{T}_{\boldsymbol{\alpha}}$ and $\boldsymbol{q}'$ can be constructed by logspace transducers (Proposition A.11), and that $\boldsymbol{C}$ accepts $\boldsymbol{\alpha}$ iff $\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A} \models \boldsymbol{q}'(a)$ (Proposition A.12). □

10 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, as well as 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 $\mathsf{NL/poly} \subseteq^? \mathsf{NC}^1$, $\mathsf{LogCFL/poly} \subseteq^? \mathsf{NC}^1$ and $\mathsf{NP/poly} \subseteq^? \mathsf{NC}^1$.

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 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 [36]).

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 the follow-up paper [10], we give polynomial-size NDL-rewritings for these cases, which can be constructed and evaluated

in LogCFL and NL, respectively, and study the parametrised and query complexities of OMQ answering with CQs of bounded treewidth.

Although the present article 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) 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 identification of new classes of well-behaved OMQs.

As far as practical OBDA is concerned, our experience with the query answering framework Ontop [61, 77], 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 [15, 63, 76, 79] and constitutes another promising avenue for future work.

Finally, the focus of this article was on the ontology language *OWL 2 QL* that has been designed specifically for OBDA via query rewriting. However, in practice ontology engineers often require constructs that are not available in *OWL 2 QL*. Typical examples include axioms with $\vee$ and $\wedge$ such as $A(x) \to B(x) \vee C(x)$ and $P(x, y) \wedge A(y) \to B(x)$. The former is a standard covering constraint in conceptual modelling, while the latter occurs in biomedical ontologies such as SNOMED CT. There are at least three ways of extending the applicability of rewriting techniques to a wider class of ontology languages. One is the combined approach discussed in Section 1.4. A second 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 admit rewritings. This has motivated the development of diverse methods for identifying particular ontologies and OMQs for which (first-order or Datalog) rewritings exist [12, 16, 44, 50, 66]. A third 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 [17], and thus provides the exact set of answers. Alternatively, one can use a simpler ontology to approximate the answers for the full one [17, 28] (possibly employing a more costly complete algorithm to decide the status of the remaining candidate answers [89]).

A SUPPLEMENTARY MATERIALS

A.1 Proof of Theorem 3.9

THEOREM 3.9. *For any OMQ $\boldsymbol{Q}(\boldsymbol{x})$, the formulas $\Phi_{\mathrm{tw}}(\boldsymbol{x})$ and $\Phi'_{\mathrm{tw}}(\boldsymbol{x})$ are equivalent, and so $\Phi'_{\mathrm{tw}}(\boldsymbol{x})$ is a PE-rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances.*

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:

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

$$\equiv\; \exists z_0 \Big(\bigwedge_{z\in \mathfrak{t}_{\mathrm{r}}\cup\mathfrak{t}_{\mathrm{i}}} (z = z_0) \;\wedge \bigvee_{\mathfrak{t} \text{ is } \varrho\text{-initiated}} \varrho^*(z_0) \Big) \;\equiv\; \exists z_0 \Big(\bigwedge_{z\in \mathfrak{t}_{\mathrm{r}}\cup\mathfrak{t}_{\mathrm{i}}} (z = z_0) \;\wedge \bigvee_{\mathfrak{t} \text{ is generated by } \tau} \tau(z_0) \Big), \tag{20}$$

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 role ϱ that initiates $\mathfrak{t}$ and is such that $\mathcal{T} \models \tau(x) \to \exists y\, \varrho(x, y)$.

By (20), $\Phi'_{\mathrm{tw}}(\boldsymbol{x})$ can be equivalently expressed as follows:

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

Finally, we observe that, for every independent $\Theta \subseteq \Theta_{\boldsymbol{Q}}$, the variables that occur in some $\mathfrak{t}_{\mathrm{i}}$, for $\mathfrak{t} \in \Theta$, do not occur in $\mathfrak{t}'_{\mathrm{i}}$, for any other $\mathfrak{t}' \in \Theta$. It follows that if $z \in \mathfrak{t}_{\mathrm{i}}$ 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 $\Phi_{\mathrm{tw}}(\boldsymbol{x})$. In particular, this means that $\Phi'_{\mathrm{tw}}(\boldsymbol{x})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$ over complete data instances. □

A.2 Proofs of Theorems 4.2 and 4.4 and of Proposition 4.3

THEOREM 4.2. *If $f^{\nabla}_{\boldsymbol{Q}}$ is computed by a Boolean formula (monotone formula or monotone circuit) χ, then $\boldsymbol{Q}$ has an FO- (respectively, PE- or NDL-) rewriting of size $O(|\chi| \cdot |\boldsymbol{Q}|)$.*

PROOF. The cases of FO- and PE-rewritings are dealt with in Section 4.1. So, let $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ be an OMQ and $\boldsymbol{C}$ a monotone circuit computing $f^{\nabla}_{\boldsymbol{Q}}$. Let $\mathfrak{t}^1, \dots, \mathfrak{t}^l$ be tree witnesses for $\boldsymbol{Q}(\boldsymbol{x})$ with $\boldsymbol{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \bigwedge_{i=1}^{n} S_i(z_i)$. We assume that the gates $g_1, \dots, g_n$ of $\boldsymbol{C}$ are the inputs $p_{S_1(z_1)}, \dots, p_{S_n(z_n)}$ 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, whereas $g_{n+l+1}, \dots, g_m$ are AND- and OR-gates. Denote by Π the following NDL-program, where $\boldsymbol{z} = \boldsymbol{x} \cup \boldsymbol{y}$:

- $S_i(z_i) \to G_i(\boldsymbol{z})$, for $0 < i \le n$;
- $\tau(u) \to G_{n+j}(\boldsymbol{z}[\mathfrak{t}^j_{\mathrm{r}}/u])$, for $0 < j \le l$ and τ generating $\mathfrak{t}^j$, where $\boldsymbol{z}[\mathfrak{t}^j_{\mathrm{r}}/u]$ is the result of replacing each $z \in \mathfrak{t}^j_{\mathrm{r}}$ in $\boldsymbol{z}$ with u;
- $\begin{cases} G_j(\boldsymbol{z}) \wedge G_k(\boldsymbol{z}) \to G_i(\boldsymbol{z}), & \text{if } g_i = g_j \wedge g_k, \\ G_j(\boldsymbol{z}) \to G_i(\boldsymbol{z}) \text{ and } G_k(\boldsymbol{z}) \to G_i(\boldsymbol{z}), & \text{if } g_i = g_j \vee g_k, \end{cases}$ 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 $\boldsymbol{Q}(\boldsymbol{x})$. □

Theorem 4.4. *If $f^{\blacktriangledown}_{\boldsymbol{Q}}$ is computed by a Boolean formula (monotone formula or monotone circuit) χ, then $\boldsymbol{Q}$ has an FO- (respectively, PE- or NDL-) rewriting of size $O(|\chi| \cdot |\boldsymbol{Q}|)$.*

Proof. Let χ be a Boolean formula that computes

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

and let $\Phi(\boldsymbol{x})$ be the FO-formula obtained by replacing each $p_{\boldsymbol{z}=\boldsymbol{z}'}$ in χ with $\boldsymbol{z} = \boldsymbol{z}'$, each $p_{S(\boldsymbol{z})}$ with $S(\boldsymbol{z})$, each $p_{\varrho^*(\boldsymbol{z})}$ with $\bigvee_{\mathcal{T} \models \tau(\boldsymbol{x}) \to \exists \boldsymbol{y}\, \varrho(\boldsymbol{x},\boldsymbol{y})} \tau(\boldsymbol{z})$, and prefixing the result with $\exists \boldsymbol{y}$. Recall that the modified rewriting $\Phi'_{\mathsf{tw}}(\boldsymbol{x})$ was obtained by applying this same transformation to the original monotone Boolean formula for $f^{\blacktriangledown}_{\boldsymbol{Q}}$. Since χ computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$, $\Phi(\boldsymbol{x})$ and $\Phi'_{\mathsf{tw}}(\boldsymbol{x})$ are equivalent FO-formulas. As we have already established that $\Phi'_{\mathsf{tw}}(\boldsymbol{x})$ is a rewriting of $\boldsymbol{Q}(\boldsymbol{x})$, the same must be true of $\Phi(\boldsymbol{x})$. The statement regarding NDL-rewritings can be shown similarly to the proof of Theorem 4.2. □

Proposition 4.3. *The function $f^{\blacktriangledown}_{\boldsymbol{Q}}$ can be computed by a nondeterministic algorithm that runs in polynomial time in the size of $\boldsymbol{Q}$.*

Proof. Given truth-values for the $p_{S(\boldsymbol{z})}$, $p_{\boldsymbol{z}=\boldsymbol{z}'}$ and $p_{\varrho^*(\boldsymbol{z})}$ in $f^{\blacktriangledown}_{\boldsymbol{Q}}$, we first guess k, for $k \leq |\boldsymbol{q}|$, subsets $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k$ of $\boldsymbol{q}$, concepts $\tau_1, \dots, \tau_k$ and maps $h_1, \dots, h_k$ and check that the $h_i \colon \boldsymbol{q}_i \to \mathcal{C}_{\mathcal{T}}^{\tau_i(a)}$ are homomorphisms, for $1 \leq i \leq k$, and the $\mathrm{t}_1, \dots, \mathrm{t}_k$ corresponding to $\boldsymbol{q}_1, \dots, \boldsymbol{q}_k$ are tree witnesses. Then, we check whether $\Theta = \{\mathrm{t}_1, \dots, \mathrm{t}_k\}$ is independent. Finally, we check whether the polynomial-size formula (11) is true under the given truth-values for every $\mathrm{t} \in \Theta$, and every $S(\boldsymbol{z})$ with $p_{S(\boldsymbol{z})} = 0$ belongs to some $\mathrm{t} \in \Theta$. □

A.3 Generalised HGPs and THGPs

In some cases, it is convenient to use *generalised HGPs* and *THGPs* that allow hypergraph vertices labelled with conjunctions $\bigwedge_i \boldsymbol{l}_i$ of literals $\boldsymbol{l}_i$. The following two propositions show that this generalisation does not increase the computational power of HGPs and THGPs.

Proposition A.1. *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. Let P be based on a hypergraph $H = (V, E)$. To construct P', we split every vertex $v \in V$ 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 $1 \leq i \leq k$; 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 we can assume without loss of generality that $\boldsymbol{l}_i$ and $\boldsymbol{l}_j$ in each $\bigwedge_{i=1}^{k} \boldsymbol{l}_i$ have distinct variables for $i \neq j$, we have $|P'| \leq n \cdot |P|$. □

Proposition A.2. *For every generalised THGP P over n variables, there is a THGP P' computing the same function and such that $|P'| \leq n \cdot |P|$. Moreover, the degree of P' and the number of leaves in it are the same as in P.*

Proof. Let P be based on a tree hypergraph $H = (U, V, E)$. To construct P', we proceed as in the proof of Proposition A.1: we split every vertex $v \in V$ (which is an edge of the underlying tree T_H) 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 $1 \leq i \leq k$—these vertices form consecutive edges in the underlying tree in P'; each hyperedge containing v will now contain all the v_i. As before, we have $P(\boldsymbol{\alpha}) = P'(\boldsymbol{\alpha})$, for any $\boldsymbol{\alpha}$, and $|P'| \leq n \cdot |P|$. Moreover, it should be clear that the degree of P' and the number of leaves in it are the same as in P. □

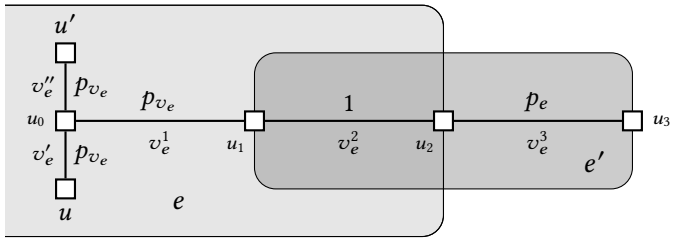


Fig. 22. Splitting an edge v_e in a tree hypergraph in the proof of Proposition 5.5.

A.4 Proof of Proposition 5.4

Proposition 5.4. *If an OMQ $\boldsymbol{Q}$ has a tree-shaped CQ $\boldsymbol{q}$ with ℓ leaves, then $\mathcal{H}(\boldsymbol{Q})$ is isomorphic to a tree hypergraph based on a tree with* $\max(2, \ell)$ *leaves.*

Proof. The case when $\boldsymbol{q}$ has no binary atoms is trivial. Otherwise, we begin with the Gaifman graph $G_{\boldsymbol{q}}$ of $\boldsymbol{q}$, which is a tree, and label its nodes u with the unary atoms $\xi(u)$ in $\boldsymbol{q}$ of the form $A(u)$ and $P(u, u)$, and its edges $\{u, v\}$ with the atoms of the form $P(u, v)$ and $P'(v, u)$ in $\boldsymbol{q}$. Then, 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 node 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 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 $\mathcal{H}(\boldsymbol{Q})$. □

A.5 Proof of Proposition 5.5

Proposition 5.5. *For any tree hypergraph H of degree at most d, there is a monotone THGP that computes f_H and is of degree at most* $\max(2, d)$ *and size $O(|H|)$.*

Proof. Let $H = (U, V, E)$. For any hyperedge $e \in E$, we fix a vertex $v_e \in e$ and perform the following for all $e \in E$. We split the edge $v_e = \{u, u'\}$ in the underlying tree T_H of H into a pair of edges $v'_e = \{u, u^0\}$ and $v''_e = \{u^0, u'\}$ and add a new chain of edges $v^i_e = \{u^{i-1}, u^i\}$, for $1 \leq i \leq 3$, to T_H, where the u^i are fresh nodes (if the same vertex is chosen for distinct hyperedges e and e', then the edge $v_e = v_{e'}$ is split only once, and the same u^0 is used for both chains of length 3). The vertices v'_e, v''_e and v^1_e are labelled with p_{v_e} and added to all hyperedges that contained v_e. We label v^2_e with 1 and add it to e. We label v^3_e with p_e and introduce a new hyperedge $e' = \{v^2_e, v^3_e\}$; see Fig. 22 All other vertices are labelled with the corresponding variables of f_H. The resulting THGP P is of degree at most $\max(2, d)$.

We claim that P computes f_H. Indeed, for any input $\boldsymbol{\alpha}$ with $\boldsymbol{\alpha}(p_e) = 0$, we have to include the hyperedge e' into the cover, and so cannot include e itself. Thus, $P(\boldsymbol{\alpha}) = 1$ iff there is an independent set $E' \subseteq \{ e \in E \mid \boldsymbol{\alpha}(p_e) = 1 \}$ covering the zeros of variables p_v. □

A.6 Proof of Theorem 5.6

Lemma A.3. *For any tree hypergraph $H = (U, V, E)$ of degree at most 2, we have $f_H(\boldsymbol{\alpha}) = 1$ iff $\boldsymbol{\alpha}$ neither is degenerate nor meets any obstruction.*

Proof. It should be clear that f_H outputs 0 if either $\boldsymbol{\alpha}$ is degenerate or meets an obstruction.

Conversely, if $\boldsymbol{\alpha}$ is neither degenerate and nor meets any obstruction, then we construct an independent subset $E' \subseteq \{e \in E \mid \boldsymbol{\alpha}(p_e) = 1\}$ covering all vertices labelled with zeros under $\boldsymbol{\alpha}$ as follows.

Let E_0 be the set of all $e_0 \in E$ such that $\boldsymbol{\alpha}(p_{e_0}) = 1$ and there is $v_0 \in e_0$ with $\boldsymbol{\alpha}(p_{v_0}) = 0$ and $\boldsymbol{\alpha}(p_{e'}) = 0$ for any hyperedge $e' \neq e_0$ with $v_0 \in e'$; cf. (O1) and (O2). Let E_1 be the result of iteratively (until a fixpoint) extending E_0 with hyperedges $e_2 \in E$ such that $\boldsymbol{\alpha}(p_{e_2}) = 1$ and

$$\text{there are } e_0 \in E_0 \text{ and } e_1 \in E \setminus E_0 \text{ with } e_0 \cap e_1 \neq \emptyset \text{ and } v \in e_1 \cap e_2 \text{ with } \boldsymbol{\alpha}(p_v) = 0; \tag{21}$$

cf. (O3). We claim that E_1 is independent. For the sake of contradiction, suppose there are distinct $e, e' \in E_1$ with $e \cap e' \neq \emptyset$. Then there are sequences $(e_0, e_1, \dots, e_{2k})$ and $(e'_0, e'_1, \dots, e'_{2k'})$ satisfying (O1) and (O3) and such that $e = e_{2k}$ and $e' = e'_{2k'}$. Consider now the sequence

$$(e_0, e_1, \dots, e_{2k}, e'_{2k'}, \dots, e'_1, e'_0).$$

By construction, it satisfies conditions (O1)–(O3). However, it may contain duplicating hyperedges, say, e_i and $e'_{i'}$. If that is the case, then we remove $e_{i+1}, \dots, e'_{i'}$ from the sequence. By construction, the result of the exhaustive removal of repetitions is an obstruction, and $\boldsymbol{\alpha}$ meets it from v_0 to v'_0.

Next, if there is a vertex v with $\boldsymbol{\alpha}(p_v) = 0$ but $v \notin \bigcup E_1$, then we extend E_1: since $\boldsymbol{\alpha}$ is non-degenerate, there is some $e_* \in E$ such that $v \in e_*$, $\boldsymbol{\alpha}(p_{e_*}) = 1$ and $e_* \cap \bigcup E_1 = \emptyset$. (Indeed, there are two hyperedges e'_* and e''_* containing v with $\boldsymbol{\alpha}(p_{e'_*}) = \boldsymbol{\alpha}(p_{e''_*}) = 1$; for otherwise, a single such hyperedge would be in E_0. Next, if $e'_* \cap \bigcup E_1 \neq \emptyset$, then, by (21), we would get $e''_* \in E_1$; thus, neither e'_* nor e''_* intersects $\bigcup E_1$.) Now we apply the extension procedure to $E_1 \cup \{e_*\}$, that is, we iteratively (until a fixpoint) extend the subset of hyperedges by all $e_2 \in E$ such that $\boldsymbol{\alpha}(p_{e_2}) = 1$ and

$$\text{there are } e_0 \in E_1 \text{ and } e_1 \in E \setminus E_1 \text{ with } e_0 \cap e_1 \neq \emptyset \text{ and } v \in e_1 \cap e_2 \text{ with } \boldsymbol{\alpha}(p_v) = 0. \tag{22}$$

We claim that the resulting set E_2 is independent. Indeed, if E_2 contains e and e' with $e \cap e' \neq \emptyset$, then one of them, say e, was added to E_2 due to (22) with some e_0, e_1 and v. We claim that the second hyperedge (in our case, e') could not have been added to E_2. The intuition is as follows: if both e and e' were added to E_2, then they are 'reachable' from e^* by two different paths (we may speak of paths because the hyperedges are subtrees, and each vertex can belong to at most 2 hyperedges). Since $e \cap e' \neq \emptyset$, these two paths intersect, forming a 'loop', which contradicts the fact that the underlying tree of the hypergraph has no cycles. It follows that e' could not have been added to E_2, and so we must have $e' \in E_1$. This, however, means that e_1 should have been added to E_1 by using (21) for e', e and v (in place of, respectively, e_0, e_1 and v in (21)) contrary to the assumption that e was added only to E_2 using it.

The process of extending E_k to E_{k+1} is repeated until there are no vertices v such that $\boldsymbol{\alpha}(p_v) = 0$ but $v \notin \bigcup E_k$. It can be seen that the resulting set E_k is independent. □

A.7 Proof of Theorem 5.8

Theorem 5.8. *For any tree hypergraph H based on a tree with at most ℓ leaves, there is a monotone linear THGP that computes the function f_H and is of size $O(|H|^{3\ell+1})$.*

Proof. Let $H = (U, V, E)$ be a tree hypergraph: each $e \in E$ induces a convex subtree T_e of the underlying tree T_H. Pick some $r \in U$ and fix it as a root of T_H. Let $\prec$ be the order on the flat subsets of E defined in Section 5.2. For a flat F, let before(F) be the edges of T_H that lie outside $\bigcup F$ and are accessible from the root r via paths not passing through $\bigcup F$; we denote by after(F) the edges of T_H outside $\bigcup F$ that are accessible from r only via paths passing through $\bigcup F$. For flat F and F' with $F \prec F'$, we denote by between(F, F') the set of edges in T_H 'between' $\bigcup F$ and $\bigcup F'$, that is, the edges outside $\bigcup F$ and $\bigcup F'$ that are accessible from $\bigcup F$ via paths not passing through $\bigcup F'$

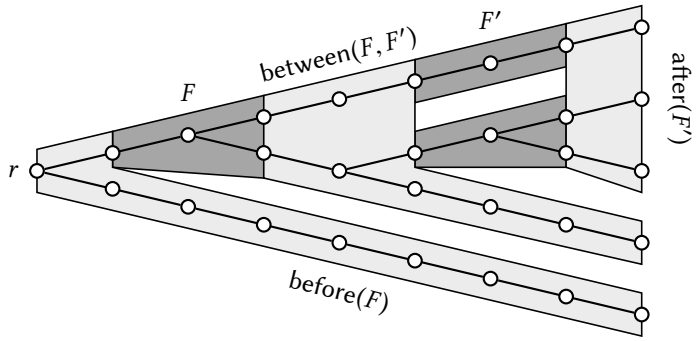


Fig. 23. Parts of the underlying tree in the proof of Theorem 5.8: $F \prec F'$ for F with one hyperedge and F' with two hyperedges.

but are inaccessible from the root r via a path not passing through $\bigcup F$; see Fig. 23. Recall that the edges of T_H are the vertices of the reduct (V, E) of H and so, the defined sets are subsets of V.

Let $F_1, \dots, F_m$ be the flat subsets of E, where $m = O(|H|^\ell)$. The required linear THGP will use a linear gadget based on the following nodes:

$$m+2 \text{ pairs } f_i, \bar{f}_i, \quad (m+2)^2 \text{ pairs } r_{ij}, \bar{r}_{ij}, \quad m+2 \text{ pairs } f'_j, \bar{f}'_j,$$

where $0 \le i, j \le m+1$; the three groups are ordered as above, and the nodes within each group are ordered lexicographically: e.g., $f_i, \bar{f}_i$ is followed by $f_{i+1}, \bar{f}_{i+1}$, whereas $r_{ij}, \bar{r}_{ij}$ is followed by $r_{i(j+1)}, \bar{r}_{i(j+1)}$ if $j < m+1$ and by $r_{(i+1)0}, \bar{r}_{(i+1)0}$ if $j = m+1$ (and $i < m+1$). The gadget vertices are pairs of consecutive elements in the order and are labelled as follows: for all $1 \le i, j \le m$,

- $\{f_0, \bar{f}_0\}$, $\{f_i, \bar{f}_i\}$, $\{f'_j, \bar{f}'_j\}$ and $\{f'_{m+1}, \bar{f}'_{m+1}\}$ with 1,
- $\{r_{ij}, \bar{r}_{ij}\}$ with all p_v, for $v \in \text{between}(F_i, F_j)$, and all p_e, for $e \in F_j$, provided that $F_i \prec F_j$,
- $\{r_{0j}, \bar{r}_{0j}\}$ with all p_v, for $v \in \text{before}(F_j)$, and all p_e, for $e \in F_j$,
- $\{r_{i,m+1}, \bar{r}_{i,m+1}\}$ with all p_v, for $v \in \text{after}(F_i)$,
- $\{r_{0,m+1}, \bar{r}_{0,m+1}\}$ with p_v, for all $v \in V$,
- $\{r_{m+1,m+1}, \bar{r}_{m+1,m+1}\}$ with 1,
- all other vertices are labelled with 0.

Note that the vertices are labelled with conjunctions of variables, and so the constructed linear THGP will be generalised. The hyperedges are of the form

$$g_{ij} = [\bar{f}_i, r_{ij}] \quad \text{and} \quad \bar{g}_{ij} = [\bar{r}_{ij}, f'_j] \qquad \text{if } F_i \prec F_j \text{ or } i = 0 \text{ or } j = m+1,$$

for $0 \le i, j \le m+1$. To construct the required generalised linear THGP P (see Appendix A.3), we take $m+2$ isomorphic copies of the gadget placing them in a linear order one after another and identify each f'_j and $\bar{f}'_j$ with f_j and $\bar{f}_j$ in the successive copy.

We show that P computes f_H. Indeed, if $f_H(\boldsymbol{\alpha}) = 1$, then there is an independent subset $E' \subseteq E$ such that $\boldsymbol{\alpha}(p_v) = 1$, for $v \in V \setminus V_{E'}$ and $\boldsymbol{\alpha}(p_e) = 1$, for $e \in E'$. If $E' = \emptyset$, then we can choose hyperedges $g_{0,m+1}$ and $\bar{g}_{0,m+1}$ in every copy to cover all zeros under $\boldsymbol{\alpha}$. Otherwise, E' is partitioned into flat $F_{i_1} \prec \dots \prec F_{i_k}$. As $i_k \le m$, we can extend the sequence of indices $i_0 = 0, i_1, \dots, i_k, i_{k+1} = m+1$ by repeating $m+1$ so that the result is of length $m+2$. It then can be verified that by choosing g_{i_{j-1}, i_j} and $\bar{g}_{i_{j-1}, i_j}$ in the jth copy ($1 \le j \le m+2$), we cover all zeros under $\boldsymbol{\alpha}$. Conversely, if there is a cover of all zeros under $\boldsymbol{\alpha}$ in P, then it must contain exactly one matching pair of hyperedges g_{ij} and $\bar{g}_{ij}$ from each copy of the gadget and, for each such $\bar{g}_{ij}$, the successive copy of the gadget will contain a hyperedge of the form g_{jk}. Moreover, the first copy

can only choose hyperedges of the form g_{0i_1} and $\bar{g}_{0i_1}$, with $1 \leq i_1 \leq m+1$, and the final copy can only choose hyperedges of the form $g_{i_{m+1},m+1}$ and $\bar{g}_{i_{m+1},m+1}$, with $1 \leq i_{m+1} \leq m+1$. If $i_1 = m+1$, then $E' = \emptyset$ is independent and $\boldsymbol{\alpha}(p_v) = 1$, for all $v \in V$. Otherwise, there is a sequence of indices $i_1, \dots, i_k$ that determines $F_{i_1} \prec F_{i_2} \prec \cdots \prec F_{i_k}$. It can be verified that taking $E' = \bigcup_j F_{i_j}$ guarantees $f_H(\boldsymbol{\alpha}) = 1$. Observe that $|P| = O(|H|^{3\ell})$.

Finally, it remains to apply Proposition A.2 to convert the generalised linear THGP P into a linear THGP of size $O(|H|^{3\ell+1})$. □

A.8 Proofs of Theorems 5.12 and 5.14

Theorem 5.12. *For every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with a fundamental set $\Omega_{\boldsymbol{Q}}$ and a CQ of treewidth t, there is a monotone THGP that computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$ and is of degree polynomial in $|\Omega_{\boldsymbol{Q}}|^t$ and size polynomial in $|\boldsymbol{q}|$ and $|\Omega_{\boldsymbol{Q}}|^t$.*

Proof. We consider the generalised THGP P constructed in Section 5.3, which is based on a tree hypergraph $H = (U, V, E)$ with the underlying tree $T_H = (U, V)$.

First, we consider the size of P. Recall that (T, λ) is a tree decomposition of $G_{\boldsymbol{q}}$ of width $m-1$ and $M = |\Omega_{\boldsymbol{Q}}|^m$ is the number of bag types. Let K be the number of nodes in the tree T. By [32, Lemma 11.9], we can assume that $K \leq |\boldsymbol{q}|$. We claim that

- P contains at most $(4M+1) \cdot K$ vertices and at most $(M + M^2) \cdot K$ hyperedges, and
- P has labels with at most $3|\boldsymbol{q}|$ conjuncts.

The vertices of the hypergraph of P correspond to the edges of T_H, and there are at most $(4M+1) \cdot K$ of them: indeed, there are no more than K edges in T, each of which is replaced by a sequence of $4M+1$ edges in T_H. The hyperedges are of two types: e^k_i, for $1 \leq i \leq K$ and $1 \leq k \leq M$, and $f^{k\ell}_{ij}$, for an edge $\{i, j\}$ in T and $1 \leq k, \ell \leq M$. It follows that the total number of hyperedges does not exceed $(M + M^2) \cdot K$. Finally, a simple examination of the labelling function shows that there are at most $3|\boldsymbol{q}|$ conjuncts in each label: indeed, given i, j and k, each atom $S(\boldsymbol{z})$ with $\boldsymbol{z} \subseteq \lambda(N_i)$ gives rise to 1, 2 or 3 propositional variables in the label of $\{u^k_{ij}, v^k_{ij}\}$, and $|\boldsymbol{q}|$ is the upper bound for the number of such atoms.

Next, we show that P computes $f^{\blacktriangledown}_{\boldsymbol{Q}}$: for any assignment $\boldsymbol{\alpha}$,

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

($\Rightarrow$) 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 $f^{\blacktriangledown}_{\boldsymbol{Q}}$:

$$\bigwedge_{S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta} p_{S(\boldsymbol{z})} \quad \wedge \bigwedge_{\mathsf{t} \in \Theta} \Bigg(\bigwedge_{P(\boldsymbol{z}, \boldsymbol{z}') \in \boldsymbol{q}_\mathsf{t}} p_{\boldsymbol{z}=\boldsymbol{z}'} \quad \wedge \bigvee_{\mathsf{t} \text{ is } \varrho\text{-initiated}} \bigwedge_{\boldsymbol{z} \in \mathsf{t}_\mathsf{r} \cup \mathsf{t}_\mathsf{i}} p_{\varrho^*(\boldsymbol{z})} \Bigg). \tag{23}$$

For every $\mathsf{t} \in \Theta$, let ϱ_t be a role that makes the disjunction hold. We use the following lemma to construct bag types for all nodes in T.

Lemma A.4. *Let $\Theta \subseteq \Theta_{\boldsymbol{Q}}$ be an independent set of tree witnesses for $\boldsymbol{Q}$, and, for each $\mathsf{t} \in \Theta$, let ϱ_t be a role and $h_\mathsf{t} \colon \boldsymbol{q}_\mathsf{t} \to \mathcal{C}_{\mathcal{T}}^{\exists y \varrho_\mathsf{t}(a,y)}$ a homomorphism such that each $h_\mathsf{t}(\boldsymbol{z})$ for $\boldsymbol{z} \in \mathsf{t}_\mathsf{i}$ is of the form $a\varrho_\mathsf{t}\sigma$, for some σ with $\varrho_\mathsf{t}\sigma \in \Omega_{\boldsymbol{Q}}$ (in other words, t is ϱ_t-initiated and induced by h_t). Then, with each node N in the tree decomposition (T, λ), we can associate a bag type $\tau(N)$ by taking, for all $\boldsymbol{z} \in \lambda(N)$:*

$$\tau(N)[\boldsymbol{z}] = \begin{cases} w, & \text{if } \boldsymbol{z} \in \mathsf{t}_\mathsf{i} \text{ and } h_\mathsf{t}(\boldsymbol{z}) = aw, \text{ for some } \mathsf{t} \in \Theta, \\ \varepsilon, & \text{otherwise,} \end{cases} \tag{24}$$

such that $\tau(N)$ is compatible with N and the pairs $(\tau(N), \tau(N'))$ of types associated with pairs (N, N') of nodes in T are compatible.

Proof. Observe that $\tau(N)$ is well-defined since the independence of Θ guarantees that every variable in $\boldsymbol{q}$ can occur in at most one t_i, for $\mathsf{t} \in \Theta$. We show that $\tau(N)$ is compatible with N. For condition (C1), consider $A(z) \in \boldsymbol{q}$ with $z \in \lambda(N)$ and $\tau(N)[z] \neq \varepsilon$. Then there is $\mathsf{t} \in \Theta$ with $z \in \mathsf{t}_\mathsf{i}$, in which case $h_\mathsf{t}(z) = a \cdot \tau(N)[z]$. Let ϱ be the final symbol in $h_\mathsf{t}(z)$. Since $h_\mathsf{t} \colon \boldsymbol{q}_\mathsf{t} \to \mathcal{C}_\mathcal{T}^{\exists y \varrho_\mathsf{t}(a,y)}$ is a homomorphism, we have $\mathcal{T} \models \exists y\, \varrho(y,x) \to A(x)$. For condition (C2), consider $P(z,z') \in \boldsymbol{q}$ with $z, z' \in \lambda(N)$ and either $\tau(N)[z] \neq \varepsilon$ or $\tau(N)[z'] \neq \varepsilon$. We assume without loss of generality that the former is true (the latter is handled analogously). By definition, there is $\mathsf{t} \in \Theta$ such that $z \in \mathsf{t}_\mathsf{i}$ and $h_\mathsf{t}(z) = a \cdot \tau(N)[z]$. Since $z \in \mathsf{t}_\mathsf{i}$ and $P(z,z') \in \boldsymbol{q}$, by the definition of tree witnesses, $z' \in \mathsf{t}_\mathsf{r} \cup \mathsf{t}_\mathsf{i}$. Since $h_\mathsf{t} \colon \boldsymbol{q}_\mathsf{t} \to \mathcal{C}_\mathcal{T}^{\exists y \varrho_\mathsf{t}(a,y)}$ is a homomorphism, one of the following holds:

- $\tau(N)[z'] = \tau(N)[z]$ and $\mathcal{T} \models P(x,x)$;
- $\tau(N)[z'] = \tau(N)[z] \cdot \varrho$ or $\tau(N)[z] = \tau(N)[z'] \cdot \varrho^-$ for some ϱ with $\mathcal{T} \models \varrho(x,y) \to P(x,y)$.

This establishes compatibility of $\tau(N)$ with N. Next, by construction, the pairs $(\tau(N), \tau(N'))$ of types associated with pairs (N, N') of nodes in T are compatible: $\tau(N)[z] = \tau(N')[z]$, for all z in $\lambda(N) \cap \lambda(N')$. □

Recall that $\boldsymbol{w}_1, \dots, \boldsymbol{w}_M$ are all the bag types for Ω_Q, and consider the following subset $E' \subseteq E$ of hyperedges in H:

- $e_i^k = [N_i, u_{ij_1}^k, \dots, u_{ij_n}^k]$, where k is such that $\boldsymbol{w}_k = \tau(N_i)$ and $N_{j_1}, \dots, N_{j_n}$ are the neighbours of N_i, for every node N_i in T;
- $f_{ij}^{k\ell} = [v_{ij}^k, v_{ji}^\ell]$, where k and l are such that $\boldsymbol{w}_k = \tau(N_i)$ and $\boldsymbol{w}_\ell = \tau(N_j)$, for every edge $\{N_i, N_j\}$ in T.

Note that, by Lemma A.4, these are hyperedges of H. It is easy to see that E' is independent: whenever we include e_i^k or $f_{ij}^{k\ell}$, we do not include any $e_i^{k'}$ or $f_{ij}^{k'\ell}$ for $k' \neq k$. It remains to show that every vertex of H that is not covered by E' evaluates to 1 under $\boldsymbol{\alpha}$. Observe first that most of the vertices are covered by E'. Specifically:

- $\{N_i, u_{ij}^1\}$ is covered by e_i^k;
- $\{v_{ij}^k, u_{ij}^{k+1}\}$ is covered either by e_i^n (if $n \leq k+1$) or by $e_{ij}^{n\ell}$ (if $n > k+1$);
- $\{v_{ij}^M, v_{ji}^M\}$ is covered by $e_{ij}^{k\ell}$;
- $\{u_{ij}^k, v_{ij}^k\}$ is covered by e_i^n if $k < n$, and by $e_{ij}^{n\ell}$ if $n > k$.

So, the only type of vertex not covered by E' is of the form $\{u_{ij}^k, v_{ij}^k\}$, where $\boldsymbol{w}_k = \tau(N_i)$. Recall that the vertex $\{u_{ij}^k, v_{ij}^k\}$ is labelled with the following variables:

(*i*) $p_{S(\boldsymbol{z})}$, whenever $S(\boldsymbol{z}) \in \boldsymbol{q}$, $\boldsymbol{z} \subseteq \lambda(N_i)$ and $\boldsymbol{w}_k[z] = \varepsilon$, for all $z \in \boldsymbol{z}$;
(*ii*) $p_{\varrho^*(z)}$, whenever $A(z) \in \boldsymbol{q}$, $z \in \lambda(N_i)$ and $\boldsymbol{w}_k[z] = \varrho\sigma$ for some σ;
(*iii*) $p_{\varrho^*(z)}$, $p_{\varrho^*(z')}$ and $p_{z=z'}$, whenever $P(z,z') \in \boldsymbol{q}$ (possibly with $z = z'$), $z, z' \in \lambda(N_i)$ and either $\boldsymbol{w}_k[z] = \varrho\sigma$ or $\boldsymbol{w}_k[z'] = \varrho\sigma$ for some σ.

First suppose that $p_{S(\boldsymbol{z})}$ appears in the label of $\{u_{ij}^k, v_{ij}^k\}$. Then $\boldsymbol{w}_k[z] = \varepsilon$, for all $z \in \boldsymbol{z}$, and hence no variable in $S(\boldsymbol{z})$ belongs to any t_i for $\mathsf{t} \in \Theta$. It follows that $S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta$, and since (23) 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 come from a binary atom (item (*iii*) above), but the proof for the case of a unary atom (item (*ii*) above) is similar. So, there is some atom $P(z,z') \in \boldsymbol{q}$ with $z, z' \in \lambda(N_i)$ and either $\boldsymbol{w}_k[z] = \varrho\sigma$ or $\boldsymbol{w}_k[z'] = \varrho\sigma$ for some σ. It follows that there is a tree witness $\mathsf{t} \in \Theta$ with $z, z' \in \mathsf{t}_\mathsf{r} \cup \mathsf{t}_\mathsf{i}$. This means that $p_{z=z'}$ is a conjunct of (23), and so it is satisfied under $\boldsymbol{\alpha}$. Also, either $\boldsymbol{w}_k[z]$ or $\boldsymbol{w}_k[z']$ is of the form $\varrho\sigma$, and, since all non-empty words in the image of h_t begin by ϱ_t, we obtain $\varrho = \varrho_\mathsf{t}$. Since ϱ_t was chosen so that $\bigwedge_{z \in \mathsf{t}_\mathsf{r} \cup \mathsf{t}_\mathsf{i}} p_{\varrho^*(z)}$ 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 $P(\boldsymbol{\alpha}) = 1$.

($\Leftarrow$) Suppose $P(\boldsymbol{\alpha}) = 1$, that is, there is an independent subset E' of the hyperedges in H that covers all zeros under $\boldsymbol{\alpha}$. By construction, the subset 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 bag type $\boldsymbol{w}_k$, denoted by $\tau(N_i)$. Also, E' contains exactly one hyperedge of the form $e_{ij}^{k\ell}$ for every edge $\{N_i, N_j\}$ in T. Moreover, if E' contains hyperedges e_i^k and $e_{ij}^{k'\ell}$ (respectively, e_j^ℓ and $e_{ij}^{k\ell'}$), then $k = k'$ (respectively, $\ell = \ell'$). By the definition of H, every $\tau(N_i)$ is compatible with N_i, and every $(\tau(N_i), \tau(N_j))$ are compatible with (N_i, N_j) for adjacent nodes N_i, N_j in T. Using the compatibility properties and the connectedness condition of tree decompositions, we can conclude that the pairs of types assigned to *any two nodes* N_i and N_j in T are compatible. Since every variable occurs in at least one node label, we can associate with each variable z in $\boldsymbol{q}$ a *unique* word $w_z \in \Omega_{\mathcal{Q}}$ such that $w_z = \tau(N)[z]$, for all nodes N with $z \in \lambda(N)$.

Denote by $\equiv$ 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.

Lemma A.5. *For every $\boldsymbol{q}_i$ that contains a variable z with $w_z \neq \varepsilon$,*

(T1) *there is a role ϱ_i such that all $w_z \neq \varepsilon$ begin by ϱ_i, for variables z in $\boldsymbol{q}_i$;*
(T2) *there is a homomorphism $h_i \colon \boldsymbol{q}_i \to \mathcal{C}_{\mathcal{T}}^{\exists y \varrho_i(a,y)}$ with $h_i(z) = a w_z$ for all z in $\boldsymbol{q}_i$;*
(T3) *there is a (ϱ_i-initiated and induced by h_i) tree witness $\mathfrak{t}^i$ for $\boldsymbol{Q}$ such that $\boldsymbol{q}_i = \boldsymbol{q}_{\mathfrak{t}^i}$.*

Proof. By the definition of $\boldsymbol{q}_i$, there is a sequence $\Xi_0, \dots, \Xi_k$ of subsets of $\boldsymbol{q}_i$ such that Ξ_0 consists of $S_0(\boldsymbol{z}_0)$ and contains a variable z_0 with $w_{z_0} \neq \varepsilon$; each Ξ_{j+1} is obtained from Ξ_j by adding an atom that contains a variable z that occurs in Ξ_j and is such that $w_z \neq \varepsilon$; and finally $\Xi_k = \boldsymbol{q}_i$. 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 $0 \leq j \leq k$, let h_j be the function that maps every variable z in Ξ_j to $a w_z$.

Properties (T1) and (T2) are shown by induction on j. The base case is trivial. For the induction step, suppose that, at stage j, for every variable z in Ξ_j, the word $w_z \neq \varepsilon$ begins by ϱ_i, and h_j is a homomorphism $\Xi_j \to \mathcal{C}_{\mathcal{T}}^{\exists y \varrho_i(a,y)}$ such that $h_j(z) = a w_z$ for all z. Let $S(\boldsymbol{z})$ be the unique atom in $\Xi_{j+1} \setminus \Xi_j$. Then $S(\boldsymbol{z})$ contains a variable z from Ξ_j such that $w_z \neq \varepsilon$. If $S(\boldsymbol{z}) = A(\boldsymbol{z})$, then (T1) for w_z is by induction hypothesis. For (T2), let N be a node in T with $z \in \lambda(N)$. Since $\tau(N)$ is compatible with N, the word w_z ends by a role ϱ with $\mathcal{T} \models \exists y\, \varrho(y, x) \to A(x)$, which proves (T2). If $S(\boldsymbol{z}) = P(z, z')$ (possibly with $z = z'$), then let N be a node in T with $z, z' \in \lambda(N)$. Since $\tau(N)$ is compatible with N, we have one of the following: (*a*) $w_{z'} = w_z$ and $\mathcal{T} \models P(x, x)$ or (*b*) either $w_{z'} = w_z \varrho$ or $w_z = w_{z'} \varrho^-$ for some ϱ with $\mathcal{T} \models \varrho(x, y) \to P(x, y)$. If (*a*) holds, then the arguments for (T1) and (T2) are similar to the previous case, except that we use $\mathcal{T} \models P(x, x)$. If (*b*) holds, then, whenever w_z begins with ϱ_i, the same holds for $w_{z'}$ unless $w_{z'} = \varepsilon$, and other way round, which proves (T1). By construction, h_{j+1} is a homomorphism from Ξ_{j+1} to $\mathcal{C}_{\mathcal{T}}^{\exists y \varrho_i(a,y)}$, so (T2) holds. If $S(\boldsymbol{z}) = P(z', z)$, then the argument is analogous.

Property (T3) follows from (T1) and (T2) by the definitions of $\boldsymbol{q}_i$ and tree witnesses. □

Let Θ consist of all the tree witnesses $\mathfrak{t}^i$ obtained by Lemma A.5. As the $\boldsymbol{q}_i$ are disjoint, the set Θ is independent. We show that $\boldsymbol{\alpha}$ satisfies the disjunct (23) of $f^{\blacktriangledown}_{\boldsymbol{Q}}$ that corresponds to Θ. To this end, observe that since all zeros are covered by E', the following variables are assigned 1 under $\boldsymbol{\alpha}$:

$$p_{S(\boldsymbol{z})}, \quad \text{if } S(\boldsymbol{z}) \in \boldsymbol{q} \text{ and } w_{\boldsymbol{z}} = \varepsilon, \text{ for all } \boldsymbol{z} \in \boldsymbol{z}; \tag{25}$$

$$p_{\varrho^*(z)}, \quad \text{if } A(z) \in \boldsymbol{q} \text{ and } w_z = \varrho\sigma, \text{ for some } \sigma; \tag{26}$$

$$p_{\varrho^*(z)}, p_{\varrho^*(z')}, p_{z=z'} \quad \text{if } P(z,z') \in \boldsymbol{q} \text{ (possibly with } z = z') \text{ and either } w_z = \varrho\sigma \text{ or } w_{z'} = \varrho\sigma, \text{ for some } \sigma. \tag{27}$$

Consider first $S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta$. We have $w_{\boldsymbol{z}} = \varepsilon$, for each variable z in $S(\boldsymbol{z})$, whence, by (25), $\boldsymbol{\alpha}(p_{S(\boldsymbol{z})}) = 1$. Next, consider a ϱ_i-initiated tree witness $\mathfrak{t}^i \in \Theta$, where ϱ_i is the role provided by Lemma A.5. If $\boldsymbol{q}_i = \{A(z)\}$, then, by (26), $\boldsymbol{\alpha}(p_{\varrho_i^*(z)}) = 1$, and we are done. Otherwise, $\boldsymbol{q}_i$ contains a binary atom. Consider any $P(z,z') \in \boldsymbol{q}_i$: by construction, either $w_z \neq \varepsilon$ or $w_{z'} \neq \varepsilon$, whence, by (27), $\boldsymbol{\alpha}(p_{z=z'}) = 1$, as required. It remains to show that $\boldsymbol{\alpha}(p_{\varrho_i^*(z)}) = 1$ for each $z \in \mathfrak{t}^i_{\mathrm{r}} \cup \mathfrak{t}^i_{\mathrm{i}}$. By construction, $\boldsymbol{q}_i$ contains some $P(\boldsymbol{z})$ with $\boldsymbol{z} = \{z, z'\}$ and either $w_z \neq \varepsilon$ or $w_{z'} \neq \varepsilon$. Then, by Lemma A.5 (T1), either $w_z = \varrho_i\sigma$ or $w_{z'} = \varrho_i\sigma$, whence, by (27), $\boldsymbol{\alpha}(p_{\varrho_i^*(z)}) = 1$, as required.

To complete the proof of Theorem 5.12, we convert the generalised THGP P into a THGP using Proposition A.2. □

Theorem 5.14. *For every OMQ $\boldsymbol{Q}(\boldsymbol{x}) = (\mathcal{T}, \boldsymbol{q}(\boldsymbol{x}))$ with an ontology of depth 1 and a CQ of treewidth t, there is a monotone THGP that computes $f^{\triangledown}_{\boldsymbol{Q}}$ and is of degree $2^{O(t)}$ and size polynomial in $|\boldsymbol{q}|$ and 2^t.*

Proof. For every tree witness $\mathfrak{t} = (\mathfrak{t}_{\mathrm{r}}, \mathfrak{t}_{\mathrm{i}})$ for $\boldsymbol{Q}(\boldsymbol{x})$, we take a fresh binary predicate $P_{\mathfrak{t}}$ (which cannot occur in any data instance) and extend $\mathcal{T}$ with the following axioms:

$$\tau(x) \to \exists y\, P_{\mathfrak{t}}(x,y), \qquad \text{if } \tau \text{ generates } \mathfrak{t},$$

$$P_{\mathfrak{t}}(x,y) \to \varrho(x,y), \qquad \text{if } \varrho(z',z) \in \boldsymbol{q}_{\mathfrak{t}},\ z' \in \mathfrak{t}_{\mathrm{r}} \text{ and } z \in \mathfrak{t}_{\mathrm{i}}.$$

Denote the resulting ontology by $\mathcal{T}'$ and let $\boldsymbol{Q}'(\boldsymbol{x}) = (\mathcal{T}', \boldsymbol{q}(\boldsymbol{x}))$. Intuitively, $P_{\mathfrak{t}}$ becomes the single most specific role that initiates $\mathfrak{t}$. It is easily verified that $\mathcal{T}'$ is also of depth 1. By Theorem 3.7, the number of tree witnesses for $\boldsymbol{Q}(\boldsymbol{x})$ does not exceed $|\boldsymbol{q}|$, and so the size of $\boldsymbol{Q}'$ is polynomial in $|\boldsymbol{Q}|$. It is easy to see that $f^{\triangledown}_{\boldsymbol{Q}'}$ coincides with $f^{\triangledown}_{\boldsymbol{Q}}$. Thus, in the sequel we assume that the given OMQ is of the form $\boldsymbol{Q}'$, which we call *explicit*.

Let (T, λ) be a tree decomposition of the Gaifman graph $G_{\boldsymbol{q}}$ of width $m-1$; and let $\boldsymbol{w}_1, \dots, \boldsymbol{w}_M$ be all the bag types for $\Omega_{\boldsymbol{Q}}$ ($M = |\Omega_{\boldsymbol{Q}}|^m$). We construct a tree $T_{H_s} = (U_s, V_s)$ from T similarly to the tree T_H in Section 5.3 except that nodes u^k_{ij} and v^k_{ij} are introduced only if $\boldsymbol{w}_k$ is strongly compatible with N_i (it follows that each edge of T corresponds to a sequence of $4 \cdot 2^m + 1$ edges in T_{H_s}, which does not depend on $\Omega_{\boldsymbol{Q}}$). Then we define a generalised monotone THGP P_s based on a tree hypergraph $H_s = (U_s, V_s, E_s)$ with the underlying tree T_{H_s} and the following hyperedges:

- $e^k_i = [N_i, u^k_{ij_1}, \dots, u^k_{ij_n}]$ if $N_{j_1}, \dots, N_{j_n}$ are the neighbours of N_i in T and $\boldsymbol{w}_k$ is strongly compatible with N_i;
- $f^{k\ell}_{ij} = [v^k_{ij}, v^\ell_{ji}]$ if $\{N_i, N_j\}$ is an edge in T, bag types $\boldsymbol{w}_k$ and $\boldsymbol{w}_\ell$ are strongly compatible with N_i and N_j, respectively, and $(\boldsymbol{w}_k, \boldsymbol{w}_\ell)$ is compatible with (N_i, N_j).

Each vertex $\{u^k_{ij}, v^k_{ij}\}$ in H_s 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 $\boldsymbol{w}_k[z] = \varepsilon$, for all $z \in \boldsymbol{z}$;
- $p_{\mathfrak{t}}$, whenever $\boldsymbol{w}_k[z] = P_{\mathfrak{t}}$ for some $z \in \lambda(N_i)$;

all other vertices are labelled with 0.

We claim that the generalised THGP P_s is of degree polynomial in 2^t and size polynomial in 2^t and $|\boldsymbol{q}|$. First, since there are at most 2^m types strongly compatible with a node, each vertex can belong to at most 2^m hyperedges of the first type and 2^{2m} hyperedges of the second type. So, the degree of P_s does not exceed $2^m + 2^{2m} = 2^{O(t)}$. It then can be easily verified that P_s contains at most $(2^{m+2}+1)\cdot|\boldsymbol{q}|$ vertices and at most $(2^m + 2^{2m})\cdot|\boldsymbol{q}|$ hyperedges, and has labels with at most $|\boldsymbol{q}|$ conjuncts.

We show now that P_s computes $f^{\vee}_{Q}$: for any assignment $\boldsymbol{\alpha}$,

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

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

$$\bigwedge_{S(\boldsymbol{z})\in\boldsymbol{q}\setminus\boldsymbol{q}_\Theta} p_{S(\boldsymbol{z})} \quad \wedge \quad \bigwedge_{\mathsf{t}\in\Theta} p_{\mathsf{t}}. \tag{28}$$

Since $\mathcal{T}$ is explicit, every $\mathsf{t} \in \Theta$ has a role P_{t} that initiates it and a homomorphism h_{t} that induces it: $h_{\mathsf{t}} \colon \boldsymbol{q}_{\mathsf{t}} \to \mathcal{C}_{\mathcal{T}}^{\exists y\, P_{\mathsf{t}}(a,y)}$ with $h_{\mathsf{t}}(z) = aP_{\mathsf{t}}$, for every $z \in \mathsf{t}_{\mathsf{i}}$. By Lemma A.4, we can assign a bag type $\tau(N)$ to each node N in T so that $\tau(N)$ is compatible with N. In fact, as condition (C3) is immediate from the choice of h_{t}, the bag types are *strongly compatible* with their nodes.

Consider now a subset E'_s of hyperedges of H_s defined in the same way as E' in the proof of ($\Rightarrow$) in Theorem 5.12. Note that E'_s are hyperedges of H_s because each $\tau(N)$ is strongly compatible with N and each pair $(\tau(N_i), \tau(N_j))$ is compatible with (N_i, N_j). Again, it is easy to see that E'_s is independent. It remains to show that every vertex of H_s that is not covered by E'_s evaluates to 1 under $\boldsymbol{\alpha}$. Observe first that most of the vertices are covered by E'_s (see the proof of Theorem 5.12 for details), and only vertices of the form $\{u^k_{ij}, v^k_{ij}\}$, for $\boldsymbol{w}_k = \tau(N_i)$, are not covered by E'_s. If $p_{S(\boldsymbol{z})}$ appears in its label such vertex, then $\boldsymbol{w}_k[z] = \varepsilon$, for all $z \in \boldsymbol{z}$, and hence no variable in $S(\boldsymbol{z})$ belongs to any t_{i} for $\mathsf{t} \in \Theta$. It follows that $S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta$, and since (28) is satisfied, $\boldsymbol{\alpha}(p_{S(\boldsymbol{z})}) = 1$. If p_{t} is part of the label, then there is $z \in \lambda(N_i)$ with $\boldsymbol{w}_k[z] = P_{\mathsf{t}}$. By the definition of $\tau(N_i)$, we obtain $\mathsf{t} \in \Theta$, whence, by (28), $\boldsymbol{\alpha}(p_{\mathsf{t}}) = 1$. Therefore, E'_s is independent and covers all zeros under $\boldsymbol{\alpha}$, which means that $P_s(\boldsymbol{\alpha}) = 1$.

($\Leftarrow$) Suppose $P_s(\boldsymbol{\alpha}) = 1$, that is, there is an independent subset E' of the hyperedges in H_s that covers all vertices evaluated to 0 under $\boldsymbol{\alpha}$. Then, in the same way as in the proof of ($\Leftarrow$) in Theorem 5.12, we can define an equivalence relation $\equiv$ on $\boldsymbol{q}$ so that its equivalence classes $\boldsymbol{q}_1, \dots, \boldsymbol{q}_n$ satisfy Lemma A.5. Since all bag types are strongly compatible, condition (C3) in addition implies that (a) all words w_z are of length at most 1 and (b) each tree witness t^i is P_{t^i}-initiated and induced by a homomorphism h_i with $h_i(z) = aP_{\mathsf{t}^i}$, for all $z \in \mathsf{t}^i_{\mathsf{i}}$. Let $\Theta = \{\mathsf{t}^1, \dots, \mathsf{t}^n\}$. As the $\boldsymbol{q}_i$ are disjoint, the set Θ is independent. We show that $\boldsymbol{\alpha}$ satisfies the disjunct of $f^{\vee}_{Q}$ that corresponds to Θ; cf. (28). Observe that, since all zeros are covered by E', the following variables are evaluated to 1 by $\boldsymbol{\alpha}$:

$$p_{S(\boldsymbol{z})}, \quad \text{if } S(\boldsymbol{z}) \in \boldsymbol{q} \text{ and } w_z = \varepsilon, \text{ for all } z \in \boldsymbol{z}; \tag{29}$$

$$p_{\mathsf{t}}, \quad \text{if } w_z = P_{\mathsf{t}}. \tag{30}$$

Now, consider some $S(\boldsymbol{z}) \in \boldsymbol{q} \setminus \boldsymbol{q}_\Theta$: for every variable z in $S(\boldsymbol{z})$, we have $w_z = \varepsilon$, whence, by (29), we get $\boldsymbol{\alpha}(p_{S(\boldsymbol{z})}) = 1$. Next, by (30), we obtain $\boldsymbol{\alpha}(p_{\mathsf{t}}) = 1$ for all $\mathsf{t} \in \Theta$, as required.

To complete the proof of Theorem 5.14, we convert the generalised THGP P_s into a THGP using Proposition A.2. □

A.9 Proof of Theorem 6.6

Theorem 6.6. *Any tree hypergraph H is isomorphic to a subgraph of $\mathcal{H}(\boldsymbol{T}_H)$, and any monotone THGP based on H computes a subfunction of $f^{\wedge}_{\boldsymbol{T}_H}$.*

Proof. Let $H = (U, V, E)$ be a tree hypergraph with $U = \{1, \dots, n\}$, for $n > 1$, and 1 be a leaf of the underlying tree T_H. The *directed* tree obtained from T_H by fixing 1 as the root and orienting the edges away from 1 is denoted by $T^* = (U, V^*)$. By definition, each $e \in E$ induces a convex subtree $T_e = (U_e, V_e)$ of T^*. Observe that, for each subtree T_e, the OMQ $\boldsymbol{T}_H$ has a tree-witness $\mathfrak{t}^e$ such that

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

Thus, H is isomorphic to the subgraph of $\mathcal{H}(\boldsymbol{T}_H)$ with vertices $R_{ij}(z_i, y_{ij})$, for $(i,j) \in V_e$, and hyperedges $\boldsymbol{q}_{\mathfrak{t}^e} \cap \{R_{ij}(z_i, y_{ij}) \mid (i,j) \in V_e\}$, for $e \in E$; in other words, the hypergraph is obtained by eliminating the S_{ij} atoms.

For the second statement, let P be based on a tree hypergraph $H = (U, V, E)$. Given an input $\boldsymbol{\alpha}$ for P, we define an assignment $\boldsymbol{\gamma}$ for the predicates in $\boldsymbol{T}_H = (\mathcal{T}, \boldsymbol{q})$ by taking each $\boldsymbol{\gamma}(R_{ij})$ and $\boldsymbol{\gamma}(S_{ij})$ to be the value of the label of $(i,j) \in V^*$ under $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}(A_e) = 1$ for all $e \in E$ (of course, $\boldsymbol{\gamma}(P_\zeta) = 0$ for all normalisation predicates P_ζ). Observe that, 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

$$\mathcal{C}_{\mathcal{T},\,\mathcal{A}(\boldsymbol{\gamma})} \models \bigwedge_{(r_e,j)\in V_e} R_{r_e j}(a, w_e) \;\wedge\; \bigwedge_{(i,j)\in V_e,\ j\in L_e} S_{ij}(w_e, a) \;\wedge\; \bigwedge_{(i,j)\in V_e,\ i\neq r_e} R_{ij}(w'_e, w_e) \;\wedge\; \bigwedge_{(i,j)\in V_e,\ j\notin L_e} S_{ij}(w_e, w'_e).$$

We show that $P(\boldsymbol{\alpha}) =$ iff $f^{\wedge}_{\boldsymbol{T}_H}(\boldsymbol{\gamma}) = 1$.

($\Rightarrow$) Suppose that $P(\boldsymbol{\alpha}) = 1$. Then there exists an independent $E' \subseteq E$ that covers all zeros under $\boldsymbol{\alpha}$. We show that $\mathcal{T}, \mathcal{A}(\boldsymbol{\gamma}) \models \boldsymbol{q}$ (that is, $f^{\wedge}_{\boldsymbol{T}_H}(\boldsymbol{\gamma}) = 1$). 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 $\mathcal{C}_{\mathcal{T},\,\mathcal{A}(\boldsymbol{\gamma})}$. Consider a pair of atoms $R_{ij}(z_i, y_{ij})$ and $S_{ij}(y_{ij}, z_j)$ in $\boldsymbol{q}$. We have $(i,j) \in V^*$. 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 of e 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 $h(y_{ij}) = w_e$;
- if 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.

($\Leftarrow$) Suppose that $f^{\wedge}_{\boldsymbol{T}_H}(\boldsymbol{\gamma}) = 1$. 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 under $\boldsymbol{\alpha}$. 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. Recall that r_e is 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 $\boldsymbol{q}$ (using the order of variables induced by the tree T^*). Observe that z cannot be of the form z_j, for otherwise $\boldsymbol{q}$ would contain an 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 $(p,j) \in V^*$. 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: let $\{i,j\} \in e \cap e'$. 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 $\{i,j\} \in e$. Therefore, there is an $e \in E'$ such that $\{i,j\} \in e$, so all zeros under $\boldsymbol{\alpha}$ are covered by E'. It follows that $P(\boldsymbol{\alpha}) = 1$. □

A.10 Size of OMQs Constructed in Section 6

OMQ $\boldsymbol{Q}_H$ has an ontology of depth 2 and at most $|E| + |V|$ tree witnesses and is of size $O(|H|^2)$: indeed, the ontology has $|E|$ axioms, each of size $O(|H|)$, and the CQ at most $|V| + |E| \cdot |V|$ atoms.

OMQ $\boldsymbol{S}_H$ has an ontology of depth 1 and exactly $|E|$ tree witnesses and is of size $O(|H|)$: the ontology contains $|E|$ axioms with the total number of atoms in them not exceeding $2|V|$, and the CQ contains exactly $|V|$ atoms.

OMQ $\boldsymbol{T}_H$ has an ontology of depth 2 and a tree-shaped CQ with the same number of leaves as the underlying tree of H, has exactly $|E|$ tree witnesses (by Remark 2, we ignore the tree witnesses generated by normalisation predicates) and is of size $O(|H|^2)$: by construction, the ontology has $|E|$ axioms, each of size $O(|E|)$, and the CQ has $2|V|$ atoms.

A.11 Proofs of Theorems 7.4 and 7.6

Lemma A.6. *Any semi-unbounded fan-in circuit C of AND-depth d is equivalent to a semi-unbounded fan-in circuit C' of AND-depth d with $|C'| \leq 2^d \cdot |C|$ and $C'^{\text{left}}_n \cap C'^{\text{right}}_n = \emptyset$, for all $n \leq d$.*

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. First, take a copy C'' of C^{left}_n and feed its outputs as left inputs to the AND-gates of C of AND-depth n. This operation at most doubles the size of the circuit and ensures the property for the AND-gates of AND-depth n. Apply now the same procedure inductively to the sub-circuits of both C'' and C^{right}_n (which do not intersect). The size of the result increases at most by the factor of 2^{n-1}, and the property for all gates of AND-depth not exceeding n is ensured. □

In the setting of Theorem 7.4, let g_i be a gate in C and T_i the subtree of T rooted in v_i. 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 *entirely* in T_i and cover all zeros in T_i under $\boldsymbol{\alpha}$.

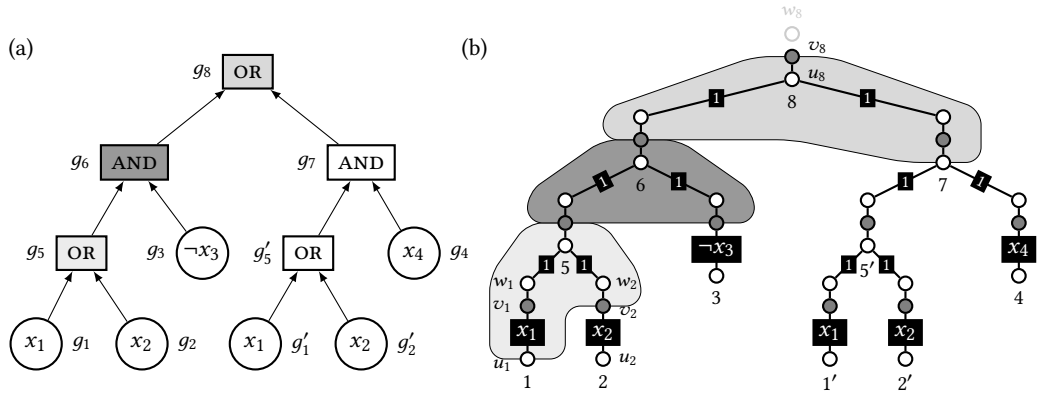


Fig. 24. (a) A formula C. (b) The labelled tree T for C: the nodes in the ith triple are u_i, v_i, w_i and the omitted edge labels are 0s. The vertices of the THGP are the edges of T (with the same labels) and the hyperedges are sets of edges of T (two of them are shown).

Lemma A.7. *For a given input $\boldsymbol{\alpha}$ and any i, the gate g_i outputs* 1 *if and only if 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 $\{u_i, v_i\}$ evaluates to 1 under $\boldsymbol{\alpha}$, and the remainder of T_i can be covered by $[u_j, w_j]$-hyperedges. Conversely, if an input gate g_i outputs 0, then no hyperedge in T_i can cover $\{u_i, v_i\}$.

If $g_i = g_j \wedge g_k$ is an AND-gate and outputs 1, then both g_j and g_k output 1. So we apply the induction hypothesis to cover both subtrees T_j and T_k (which are disjoint by construction) and add to the cover the hyperedge $[v_j, v_k, v_i]$, which is entirely inside T_i. Thus, T_i is covered. Conversely, any covering of zeros in T_i must include the hyperedge $[v_j, v_k, v_i]$, and so the subtrees T_j and T_k are covered. Thus, by the induction hypothesis, g_j and g_k should output 1, and so does g_i.

If $g_i = g_{j_1} \vee \dots \vee g_{j_k}$ is an OR-gate and outputs 1, then one of its inputs, say, g_j, outputs 1. By the induction hypothesis, we cover T_j and add the hyperedge $[v_j, v_i]$, which, possibly together with hyperedges of the form $[u_\ell, w_\ell]$, forms a covering of T_i. Conversely, since $\{u_i, v_i\}$ is labelled with 0, any covering of T_i must include a hyperedge of the form $[v_j, v_i]$, 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 7.6. $\mathsf{NC}^1 = \mathsf{THGP}^d$ *and* $\mathsf{mNC}^1 = \mathsf{mTHGP}^d$, *for any* $d \geq 3$.

Proof. To show $\mathsf{NC}^1 \subseteq \mathsf{THGP}^3$, we use a more direct construction than in the proof of Theorem 7.4. First, we assume that OR-gates have only two inputs and second, even under this assumption, the construction in the proof of Theorem 7.4 results in a hypergraph of degree 4.

Consider a polynomial-size formula C in negation normal form, 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 C has negated variables in place of NOT-gates. We first construct a tree T that contains triples of vertices u_i, v_i, w_i (in this order) partially ordered in the same way as the g_i in C. We then remove vertex w_m and make v_m the root of T. Next, we consider a hypergraph H whose vertices are the edges of T and whose hyperedges comprise the following, see Fig. 24:

- $[u_i, w_i]$, for each $i < m$;
- $[v_j, v_k, v_i]$, for each $g_i = g_j \wedge g_k$;
- $[v_j, u_k, v_i], [u_j, v_k, v_i]$, for each $g_i = g_j \vee g_k$

Finally, we construct a THGP P based on H by labelling its vertices (which are edges of T): if an input gate g_i is a literal $\boldsymbol{l}$, then we label $\{u_i, v_i\}$ with $\boldsymbol{l}$; all other edges are labelled with 0. It is not hard to check that P is of degree 3 and size polynomial in $|C|$ and computes the same function as C. Indeed, one can show by induction that we can cover all zeros in a subtree rooted at v_i by the hyperedges entirely inside this subtree iff g_i outputs 1. (Hyperedges $[u_j, w_j]$ ensure that we can cover all zeros in all subtrees rooted in vertices u_j, which is needed for OR-gates; see Fig. 24)

The inclusion $\mathsf{THGP}^d \subseteq \mathsf{NC}^1$ follows from the proof of $\mathsf{THGP} \subseteq \mathsf{LogCFL/poly}$ in Theorem 7.4. Indeed, if the degree of a given THGP P is at most d, then the disjunction in (16) has at most $d+1$ disjuncts, and so the constructed circuit is of bounded fan-in. Since its depth is $O(\log |P|)$, the size of the circuit is polynomial in $|P|$. □

A.12 Proof of the LogCFL membership in Theorem 9.3

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. (31)

For every tuple $(z \mapsto (a, n), z') \in \mathsf{frontier}$, we have $n \leq |\mathsf{stack}|$. (32)

All tuples $(z \mapsto (a, n), z') \in \mathsf{frontier}$ with $n > 0$ share the same a. (33)

Once $(z \mapsto (a, n), z')$ is added to frontier, (34)
no tuple of the form $(z \mapsto (a', n'), z')$ can ever be added to frontier.

In every successful iteration, either at least one tuple is removed from frontier (35)
or frontier is unchanged but one ϱ is popped from the stack.

If $(z \mapsto (a, n), z')$ is removed from frontier in a successful iteration, then (36)
a tuple of the form $(z' \mapsto (a', n'), z'')$ is added to frontier, for each child z'' of z' in T.

Proposition A.8. *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 (31) and (34), the total number of tuples that may appear in frontier at any point cannot exceed the number of edges in T, which is bounded by $|\boldsymbol{q}|$. By (34) and (35), 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 a **check** operations fails), or it will eventually exit the loop after reaching an empty frontier. □

Proposition A.9. *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 $h\colon \boldsymbol{q} \to C_{\mathcal{T}, \mathcal{A}}$ such that $h(\boldsymbol{x}) = \boldsymbol{a}$. Without loss of generality [6], we may choose h so that the image of h consists of elements aw with $|w| \leq 2|\mathcal{T}| + |\boldsymbol{q}|$. We use h to specify an execution of $\mathsf{BLQuery}((\mathcal{T}, \boldsymbol{q}), \mathcal{A}, \boldsymbol{a})$ that returns true. First, we fix an arbitrary variable z_0 as root, and then we choose the element $h(z_0) = a_0 w_0$. Since h defines a homomorphism of $\boldsymbol{q}(\boldsymbol{a})$ into $C_{\mathcal{T}, \mathcal{A}}$, the call $\mathsf{canMap}(z_0, a_0, \mathsf{top}(\mathsf{stack}))$ 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 nondeterministic choices to satisfy the following invariant:

$$\text{If} \quad (z \mapsto (a, m), z') \in \mathsf{frontier}, \quad \text{then} \quad h(z) = a\,\mathsf{stack}_{\leq m}. \tag{37}$$

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 (37) 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 (37)) 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 $\mathtt{canMap}(z', a', \varepsilon)$ returns true. We 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, $|\mathsf{stack}| > 0$, 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 stack is non-empty, $h(z) = h(z')$ occurs in the tree part. As h is a homomorphism, we have $\mathcal{T} \models P(x, x)$, for all $P(z, z') \in \boldsymbol{q}$, and $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ 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 (37) holds for z, it 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 (37), $h(z) = aw$, 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 (37), since h is a homomorphism and $h(z') = h(z)\varrho$, the call $\mathtt{isGenerated}(\varrho, a, \mathsf{top}(\mathsf{stack}))$ returns true, $\mathcal{T} \models \varrho(x, y) \to P(x, y)$, for all $P(z, z') \in \boldsymbol{q}$, and the call $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ 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 (37) holds for the newly added tuples.

[Case 4.] If none of Case 1, 2 or 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 (33), all tuples in deepest share the same individual a. By (37), for every tuple $(z \mapsto (a, n), z') \in \mathsf{deepest}$, we have $h(z) = aw\varrho$, where $w = \mathsf{stack}_{\leq |\mathsf{stack}|}$; moreover, as Case 3 is not applicable, $h(z') = aw$. Since 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 $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ returns true. So, we add to frontier all tuples $(z' \mapsto (a, |\mathsf{stack}|), z'')$, for children z'' of z' in T. Note that invariant (37) 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 (37). Finally, if $\mathsf{deepest} = \emptyset$, then we do nothing, but the tuples in frontier continue to satisfy (37).

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 nondeterministic choices in the **while** loop without returning false, by Proposition A.8, 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 frontier. Let L be the total number of iterations of the **while** loop. We inductively define a sequence $h_0, h_1, \ldots, h_L$ of partial functions from 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 $0 \leq i < L$:

$$\text{If } i > 0, \text{ then } \mathsf{dom}(h_{i-1}) \subseteq \mathsf{dom}(h_i), \text{ and } h_i(z) = h_{i-1}(z), \text{ for } z \in \mathsf{dom}(h_{i-1}). \tag{38}$$

$$\text{If } (z \mapsto (a, n), z') \in \text{frontier at the end of iteration } i, \text{ then} \tag{39}$$

$$h_i(z) = aw, \text{ where } w = \mathsf{stack}_{\leq n}, \tag{39a}$$

$$\text{and neither } z' \text{ nor any of its descendants belongs to } \mathsf{dom}(h_i). \tag{39b}$$

$$h_i \text{ is a homomorphism } \boldsymbol{q}_i \to \mathcal{C}_{\mathcal{T},\mathcal{A}}, \text{ where } \boldsymbol{q}_i \text{ is the restriction of } \boldsymbol{q} \text{ to } \mathsf{dom}(h_i). \tag{40}$$

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 (38) is vacuously satisfied. Property (39) holds because of the initial values of frontier and stack and 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 frontier). To see why (40) is satisfied, first suppose that $w_0 = \varepsilon$ and so $a_0 w_0 \in \mathsf{ind}(\mathcal{A})$. Then, the call $\mathsf{canMap}(z_0, a_0, \mathsf{top}(\mathsf{stack}))$ returns true. It follows that

$$\begin{array}{l} \text{if } z_0 \text{ is the } j\text{th answer variable, then } a_0 = a_j; \\ a_0 \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}, \text{ for each } A(z_0) \in \boldsymbol{q}, \quad \text{and} \quad (a_0, a_0) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}, \text{ for each } P(z_0, z_0) \in \boldsymbol{q}; \end{array}$$

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

$$\begin{array}{l} z_0 \text{ is not an answer variable of } \boldsymbol{q}; \\ \mathcal{T} \models \exists y\, \varrho(y, x) \to A(x), \text{ for each } A(z_0) \in \boldsymbol{q}, \quad \text{and} \quad \mathcal{T} \models P(x, x), \text{ for each } P(z_0, z_0) \in \boldsymbol{q}; \end{array}$$

hence h_0 homomorphically maps all atoms of $\boldsymbol{q}_0$ into $\mathcal{C}_{\mathcal{T},\mathcal{A}}$. Thus, h_0 satisfies (38)–(40).

Next, we show how to inductively define h_i from h_{i-1} while preserving (38)–(40). 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 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 (38) is by definition. For property (39), consider a tuple $\tau = (v \mapsto (c, m), v')$ that belongs to frontier at the end of iteration i. Suppose first τ was added to frontier during iteration i, in which case τ is of the form $(z' \mapsto (a', 0), z'')$ for some child z'' of z'. Property (39a) is satisfied because $\mathsf{stack}^i_{\leq 0} = \varepsilon$. Since h_{i-1} satisfies (39), z'' (a descendant of z') is not in $\mathsf{dom}(h_{i-1})$, which satisfies (39b). The remaining possibility is that τ was already in frontier at the beginning of iteration i. Since h_{i-1} satisfies (39), 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 (39a) holds for τ. Moreover, as τ was not removed from frontier during iteration i, we have $\tau \neq (z \mapsto (a, 0), z')$, and so, by (34), $v' \neq z'$. Thus, neither v' nor any of its descendants is in $\mathsf{dom}(h_i)$.

For property (40), we first note that since h_i agrees with h_{i-1} on $\mathsf{dom}(h_i)$ and h_{i-1} satisfies (40), it is only necessary to consider the atoms in $\boldsymbol{q}_i \setminus \boldsymbol{q}_{i-1}$. There are three kinds of such atoms:

- if $A(z') \in \boldsymbol{q}_i$, then, since $\mathsf{canMap}(z', a', \varepsilon)$ returns true, we have $h_i(z') = a' \in A^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
- if $P(z', z') \in \boldsymbol{q}_i$, then, since $\mathsf{canMap}(z', a', \varepsilon)$ returns true, $(h_i(z'), h_i(z')) = (a', a') \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
- 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, (40) holds for h_i.

[Option 2.] a tuple $(z \mapsto (a, n), z')$ is removed from frontier, $n = |\mathsf{stack}|$ and a role ϱ is guessed. We set $h_i(z') = h_{i-1}(z)\varrho$. By (39), $h_{i-1}(z)$ is defined. Moreover, the call $\mathtt{isGenerated}(\varrho, a, \mathsf{top}(\mathsf{stack}))$ 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 (38) is immediate from the definition of h_i, and (39b) can be shown exactly as for Option 1. To show (39a), 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 (39), we have $h_{i-1}(z) = a\,\mathsf{stack}^{i-1}_{\leq n}$. Property (39a) 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 (39), we have $h_{i-1}(v) = a\,\mathsf{stack}^{i-1}_{\leq m}$. Property (39a) 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 (40). 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 $\mathtt{canMap}(z', a, \varrho)$ 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 $\mathtt{canMap}(z', a, \varrho)$ 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 (40).

[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 (33), 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 (38) is again immediate, and the argument for (39b) is the same as in Option 1. For property (39a), 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 (39a). 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 (39a) holds for τ.

For property (40), 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 $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ 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 $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ returns true, we have $(h_i(z'), h_i(z')) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$;
- if $P(z', v) \in \boldsymbol{q}_i$ with $v \neq z'$ 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 $(h_i(z'), h_i(v)) = (a\,\mathsf{stack}^i, a\,\mathsf{stack}^i\,\varrho) \in P^{\mathcal{C}_{\mathcal{T},\mathcal{A}}}$.

[Option 4.] A tuple $(z \mapsto (a, n), z')$ is removed from frontier with $n = |\mathsf{stack}| > 0$. 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 (38) and (39b) are satisfied by h_i. For property (39a), let $\tau = (v \mapsto (c, m), v')$ be a tuple in frontier at the end of iteration i. If the tuple is 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 $h_i(z) = a\,\mathsf{stack}^i_{\leq m}$. As $h_i(z') = h_i(z)$, we have $h_i(z') = a\,\mathsf{stack}^i_{\leq m}$, so τ satisfies (39a). If τ is 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 frontier satisfy (39a).

To show (40), we consider the three types of atoms in $\boldsymbol{q}_i \setminus \boldsymbol{q}_{i-1}$:

- if $A(z') \in \boldsymbol{q}_i$ then, since $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ 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 $\mathtt{canMap}(z', a, \mathsf{top}(\mathsf{stack}))$ 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 (31) and (36) and the fact that

$$\mathsf{dom}(h_{i+1}) = \mathsf{dom}(h_i) \cup \{\, z' \mid (z \mapsto (a, n), z') \text{ is removed from frontier during iteration } i \,\}.$$

This completes the proof of Proposition A.9. □

Proposition A.10. BLQuery *can be implemented by an NAuxPDA.*

Proof. It suffices to show that BLQuery runs in nondeterministic logarithmic space and polynomial time (the size of stack does not have to be bounded).

First, we nondeterministically 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 `canMap`, can be made to run in nondeterministic logarithmic space. Then, since the children of a node in T can be identified in NL, we can decide in nondeterministic logarithmic space whether a tuple $(z_0 \mapsto (a_0, |\mathsf{stack}|, z_i)$ should be included in frontier. Moreover, since the input query $\boldsymbol{q}$ is a tree-shaped query with a bounded number of leaves, only a bounded number of tuples can be added to frontier by each such operation. Moreover, it is clear that every tuple can be stored using logarithmic space. More generally, by (31) and (34), one can show that |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.

Observe that iterations of the loop involve a polynomial number of simple operations such as

- remove a tuple from frontier, or add a tuple to 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 subroutines `isGenerated` or `canMap`.

For each of the above operations, it is either easy to see, or has already been explained, that the operation can be performed in nondeterministic logarithmic space.

To complete the proof, observe that, by (35), each iteration of the while loop involves removing a tuple from frontier or popping a role from stack. By (31), every tuple in frontier corresponds to an edge in T, and, by (34), we create at most one tuple per edge. Thus, there are at most $|\boldsymbol{q}|$ iterations involving the removal of a tuple. The total number of roles added to stack is bounded by $2|\mathcal{T}| + |\boldsymbol{q}|$ roles in the initial stack plus $|\boldsymbol{q}|$ 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 does not exceed $2|\mathcal{T}| + 3|\boldsymbol{q}|$. □

A.13 Proof of LogCFL-hardness in Theorem 9.3

Proposition A.11. *The OMQ $(\mathcal{T}_{\boldsymbol{\alpha}}, \boldsymbol{q}')$ can be computed from $\boldsymbol{C}$ by logspace transducers.*

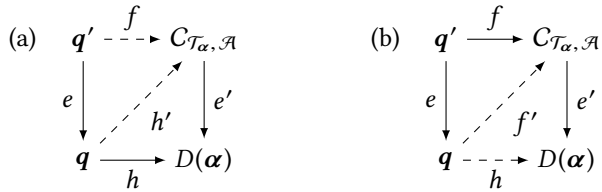
Proof. Consider a circuit $\boldsymbol{C}$ in normal form with $2d+1$ layers of gates, where d is logarithmic in the number of its inputs n. We show that $\mathcal{T}_{\boldsymbol{\alpha}}$ and $\boldsymbol{q}'$ can be constructed using $O(\log(n))$ worktape memory.

To produce the CQ $\boldsymbol{q}'$, we can generate the word w_d symbol-by-symbol 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 $\boldsymbol{\alpha}$. □

Proposition A.12. *$\boldsymbol{C}$ 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 $D(\boldsymbol{\alpha})$ [36], it remains to show that there is a homomorphism h from $\boldsymbol{q}$ to $D(\boldsymbol{\alpha})$ if and only if there exists a homomorphism f from $\boldsymbol{q}'$ to $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$:



($\Rightarrow$) Suppose h is a homomorphism from $\boldsymbol{q}$ to $D(\boldsymbol{\alpha})$. We define a homomorphism $h' : \boldsymbol{q} \to \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ inductively moving from the root z_m of $\boldsymbol{q}$ to its leaves. For the basis of induction, we set $h'(z_m) = a$; note that $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \models G_m(a)$. For the inductive step, suppose that z_j is a child of z_i, $h'(z_i)$ is defined, $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \models G_{i'}(h'(z_i))$ and $h(z_j) = g_{j'}$. In this case, we set $h'(z_j) = h'(z_i)P_{i'j'}$, where $P_{i'j'}$ is the normalisation predicate for the axiom $G_{i'}(x) \to \exists y \left(S(y,x) \wedge G_{j'}(y)\right)$. By the definition of $\mathcal{T}_{\boldsymbol{\alpha}}$, we have $\mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}} \models G_{j'}(h'(z_j))$, 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 $f : \boldsymbol{q}' \to \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}},\mathcal{A}}$ 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

$$e(y_i) = e(y_j) \quad \text{implies} \quad f(y_i) = f(y_j), \qquad \text{for all variables } y_i, y_j. \tag{41}$$

The base case ($|j-i| = 0$) is trivial. For the inductive step, we may assume without loss of generality that $i < j$ and there is no intermediate variable y_k between y_i and y_j 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 $z = e(y_i) = e(y_j)$ has a single child since in this case there are exactly two variables in $\boldsymbol{q}'$ which are mapped to z, and they bound the subtree in $\boldsymbol{q}$ generated by z. If z has two children, this also holds by our assumption on the intermediate variables. By the induction hypothesis, $f(y_{i+1}) = f(y_{j-1}) = aw\varrho$ 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 a homomorphism, it follows that $f(y_i) = f(y_j) = aw$. This concludes the inductive argument.

Next, we define function $f' \colon \boldsymbol{q} \to \mathcal{C}_{\mathcal{T}_{\boldsymbol{\alpha}}, \mathcal{A}}$ by setting $f'(z) = f(y)$, where y is such that $e(y) = z$. By (41), f' is well-defined, and since 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) above. □

ACKNOWLEDGMENTS

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', the grant MK-5379.2018.1 of the President of the Russian Federation, and the Russian Academic Excellence Project 5-100. The authors are grateful to the anonymous referees for their careful reading, valuable comments and constructive suggestions.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. 1995. *Foundations of Databases.* Addison-Wesley.
[2] M. Agrawal, E. Allender, R. Impagliazzo, T. Pitassi, and S. Rudich. 2001. Reducing the complexity of reductions. *Computational Complexity* 10, 2 (2001), 117–138.
[3] M. Agrawal, E. Allender, and S. Rudich. 1998. Reductions in circuit complexity: an isomorphism theorem and a gap theorem. *J. Comput. System Sci.* 57, 2 (1998), 127–143.
[4] N. Alon and R. Boppana. 1987. The monotone circuit complexity of Boolean functions. *Combinatorica* 7, 1 (1987), 1–22.
[5] S. Arora and B. Barak. 2009. *Computational Complexity: A Modern Approach* (1st ed.). Cambridge University Press, New York, NY, USA.
[6] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. 2009. The DL-Lite family and relations. *J. Artif. Intell. Res. (JAIR)* 36 (2009), 1–69.
[7] 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.
[8] J. Avigad. 2003. Eliminating definitions and Skolem functions in first-order logic. *ACM Trans. Comput. Logic* 4, 3 (2003), 402–415.
[9] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. 2011. On rules with existential variables: walking the decidability line. *Artif. Intell.* 175, 9–10 (2011), 1620–1654.
[10] M. Bienvenu, S. Kikot, R. Kontchakov, V. V. Podolskii, V. Ryzhikov, and M. Zakharyaschev. 2017. The complexity of ontology-based data access with OWL 2 QL and bounded treewidth queries. In *Proc. of the 36th ACM SIGMOD-SIGACT-SIGAI Symp. on Principles of Database Systems, PODS 2017.* ACM, 201–216.
[11] M. Bienvenu, S. Kikot, and V. V. Podolskii. 2015. Tree-like queries in OWL 2 QL: succinctness and complexity Results. In *Proc. of the 30th Annual ACM/IEEE Symp. on Logic in Computer Science, LICS 2015.* IEEE Computer Society, 317–328.
[12] 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.
[13] M. Bienvenu, M. Ortiz, and M. Simkus. 2015. Regular path queries in lightweight description logics: complexity and algorithms. *J. Artif. Intell. Res. (JAIR)* 53 (2015), 315–374.
[14] 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.
[15] M. Bienvenu and R. Rosati. 2015. Query-based comparison of OBDA specifications. In *Proc. of the 28th Int. Workshop on Description Logics, DL 2015 (CEUR)*, Vol. 1350. CEUR-WS, 55–66.
[16] M. Bienvenu, B. ten Cate, C. Lutz, and F. Wolter. 2014. Ontology-based data access: a study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.* 39, 4 (2014), 33:1–44.
[17] 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.*
[18] A. Brandstädt, V. B. Le, and J. P. Spinrad. 1999. *Graph Classes: A Survey.* SIAM, Philadelphia, PA, USA.
[19] A. Bretto. 2013. *Hypergraph Theory: An Introduction.* Springer.
[20] A. Calì, G. Gottlob, and T. Lukasiewicz. 2012. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semantics* 14 (2012), 57–83.
[21] A. Calì, G. Gottlob, and A. Pieris. 2012. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.* 193 (2012), 87–128.

[22] 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.
[23] 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. *J. of Autom. Reasoning* 39, 3 (2007), 385–429.
[24] A. Chandra and P. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Conf. Record of the 9th Annual ACM Symp. on Theory of Computing, STOC'77*. ACM, 77–90.
[25] C. Chekuri and A. Rajaraman. 2000. Conjunctive query containment revisited. *Theoretical Computer Science* 239, 2 (2000), 211–229.
[26] A. Chortaras, D. Trivela, and G. Stamou. 2011. Optimized query rewriting for OWL 2 QL. In *Proc. of the 23rd Int. Conf. on Automated Deduction, CADE-23 (LNCS)*, Vol. 6803. Springer, 192–206.
[27] 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 (LNCS)*, Vol. 7494. Springer, 68–80.
[28] 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 (LNCS)*, Vol. 8797. Springer, 164–179.
[29] S. A. Cook. 1971. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18, 1 (1971), 4–18.
[30] 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.
[31] C. Flament. 1978. Hypergraphes arborés. *Discrete Mathematics* 21, 3 (1978), 223–227.
[32] J. Flum and M. Grohe. 2006. *Parameterized Complexity Theory*. Springer.
[33] 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.
[34] 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. *Artif. Intell.* 213 (2014), 42–59.
[35] G. Gottlob, N. Leone, and F. Scarcello. 1999. Computing LOGCFL certificates. In *Proc. of the 26th Int. Colloquium on Automata, Languages & Programming, ICALP-99 (LNCS)*, Vol. 1644. Springer, 361–371.
[36] G. Gottlob, N. Leone, and F. Scarcello. 2001. The complexity of acyclic conjunctive queries. *J. ACM* 48, 3 (2001), 431–498.
[37] G. Gottlob, M. Manna, and A. Pieris. 2015. Polynomial rewritings for linear existential rules. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence, IJCAI 2015*. AAAI, 2992–2998.
[38] G. Gottlob, G. Orsi, and A. Pieris. 2011. Ontological queries: rewriting and optimization. In *Proc. of the 27th Int. Conf. on Data Engineering, ICDE 2011*. IEEE Computer Society, 2–13.
[39] 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 & Reasoning, KR 2012*. AAAI, 254–263.
[40] S. A. Greibach. 1973. The Hardest Context-Free Language. *SIAM J. Comput.* 2, 4 (1973), 304–310.
[41] M. Grigni and M. Sipser. 1992. Monotone complexity. In *Proc. of the London Mathematical Society Symp. on Boolean Function Complexity*. Cambridge University Press, 57–75.
[42] M. Grohe, T. Schwentick, and L. Segoufin. 2001. When is the evaluation of conjunctive queries tractable?. In *Proc. of the 33rd Annual ACM Symp. on Theory of Computing, STOC 2001*. ACM, 657–666.
[43] V. Gutiérrez-Basulto, Y. Ibáñez-García, R. Kontchakov, and E. V. Kostylev. 2015. Queries with negation and inequalities over lightweight ontologies. *J. Web Semantics* 35 (2015), 184–202.
[44] 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.
[45] 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.
[46] N. Immerman. 1988. Nondeterministic space is closed under complementation. *SIAM J. Comput.* 17, 5 (1988), 935–938.
[47] D. S. Johnson. 1990. A Catalog of Complexity Classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*. 67–161.
[48] D. S. Johnson and A. C. Klug. 1982. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proc. of the ACM Symp. on Principles of Database Systems, PODS*. ACM, 164–169.
[49] S. Jukna. 2012. *Boolean Function Complexity — Advances and Frontiers*. Algorithms and combinatorics, Vol. 27. Springer.
[50] 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.
[51] M. Karchmer and A. Wigderson. 1988. Monotone circuits for connectivity require super-logarithmic depth. In *Proc. of the 20th Annual ACM Symp. on Theory of Computing, STOC '88*. ACM, 539–550.

[52] E. Kharlamov, D. Bilidas, D. Hovland, E. Jiménez-Ruiz, D. Lanti, H. Lie, M. Rezk, M. Skjæveland, A. Soylu, G. Xiao, D. Zheleznyakov, M. Giese, Y. Ioannidis, Y. Kotidis, M. Koubarakis, and A. Waaler. 2017. Ontology based data access in Statoil. *J. Web Semantics* 44 (2017), 3–36.
[53] E. Kharlamov, T. Mailis, G. Mehdi, C. Neuenstadt, Ö. Özçep, M. Roshchin, N. Solomakhina, A. Soylu, C. Svingos, S. Brandt, M. Giese, Y. Ioannidis, S. Lamparter, R. Möller, Y. Kotidis, and A. Waaler. 2017. Semantic access to streaming and static data at Siemens. *J. Web Semantics* 44 (2017), 54–74.
[54] S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev. 2012. Exponential lower bounds and separation for query rewriting. In *Proc. of the 39th Int. Colloquium on Automata, Languages & Programming, ICALP 2012 (LNCS)*, Vol. 7392. Springer, 263–274.
[55] S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev. 2014. On the succinctness of query rewriting over shallow ontologies. In *Proc. of the Joint Meeting of the 23rd EACSL Annual Conf. on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symp. on Logic in Computer Science (LICS), CSL-LICS'14*. ACM, 57:1–57:10.
[56] 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.
[57] S. Kikot, R. Kontchakov, and M. Zakharyaschev. 2012. Conjunctive query Answering with OWL 2 QL. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation & Reasoning, KR 2012*. AAAI, 275–285.
[58] M. König, M. Leclère, and M.-L. Mugnier. 2015. Query rewriting for existential rules with compiled preorder. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence, IJCAI 2015*. AAAI, 3106–3112.
[59] M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. 2015. Sound, complete and minimal UCQ-rewriting for existential rules. *Semantic Web* 6, 5 (2015), 451–475.
[60] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. 2010. The combined approach to query answering in DL-Lite. In *Proc. of the 12th Int. Conf. on Principles of Knowledge Representation & Reasoning, KR 2010*. AAAI, 247–257.
[61] 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 (LNCS)*, Vol. 8796. Springer, 552–567.
[62] 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.
[63] 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 (LNCS)*, Vol. 9366. Springer, 217–234.
[64] L. Libkin. 2004. *Elements of Finite Model Theory*. Springer.
[65] 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.
[66] 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.
[67] C. Lutz, I. Seylan, D. Toman, and F. Wolter. 2013. The combined approach to OBDA: taming role hierarchies using filters. In *Proc. of the 12th Int. Semantic Web Conf., ISWC 2013, Part I*. 314–330.
[68] C. Lutz, D. Toman, and F. Wolter. 2009. Conjunctive query answering in the description logic EL using a relational database system. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence, IJCAI 2009*. 2070–2075.
[69] 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 (LNCS)*, Vol. 8796. Springer, 568–583.
[70] H. Pérez-Urbina, B. Motik, and I. Horrocks. 2009. A comparison of query rewriting techniques for DL-lite. In *Proc. of the 22nd Int. Workshop on Description Logics, DL 2009 (CEUR)*, Vol. 477. CEUR-WS.
[71] 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)*, Vol. 943. CEUR-WS.
[72] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. 2008. Linking data to ontologies. *J. Data Semantics* X (2008), 133–173.
[73] R. Raz and A. Wigderson. 1992. Monotone circuits for matching require linear depth. *J. ACM* 39, 3 (1992), 736–744.
[74] A. Razborov. 1985. Lower bounds for the monotone complexity of some Boolean functions. *Dokl. Akad. Nauk SSSR* 281, 4 (1985), 798–801.
[75] A. A. Razborov. 1991. Lower bounds for deterministic and nondeterministic branching programs. In *Proc. of the 8th Int. Symp. on Fundamentals of Computation Theory, FCT'91 (LNCS)*, Vol. 529. Springer, 47–60.
[76] 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 & Reasoning, KR 2012*. AAAI, 308–318.
[77] 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 (LNCS)*, Vol. 8218. Springer, 558–573.
[78] R. Rosati. 2007. The limits of querying ontologies. In *Proc. of the 11th Int. Conf. on Database Theory, ICDT 2007 (LNCS)*, Vol. 4353. Springer, 164–178.

[79] R. Rosati. 2012. Prexto: query rewriting under extensional constraints in DL-Lite. In *Proc. of the 9th Extended Semantic Web Conf., EWSC 2012 (LNCS)*, Vol. 7295. Springer, 360–374.
[80] 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 & Reasoning, KR 2010*. AAAI, 290–300.
[81] 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 (LNCS)*, Vol. 8796. Springer, 535–551.
[82] I. H. Sudborough. 1978. On the tape complexity of deterministic context-free languages. *J. ACM* 25, 3 (1978), 405–414.
[83] R. Szelepcsényi. 1988. The method of forced enumeration for nondeterministic automata. *Acta Informatica* 26, 3 (1988), 279–284.
[84] M. Thomazo. 2013. Compact rewritings for existential rules. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence, IJCAI 2013*. IJCAI/AAAI, 1125–1131.
[85] 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.
[86] H. Venkateswaran. 1991. Properties that characterize LOGCFL. *J. Comput. System Sci.* 43, 2 (1991), 380–404.
[87] H. Vollmer. 1999. *Introduction to Circuit Complexity: A Uniform Approach*. Springer.
[88] M. Yannakakis. 1981. Algorithms for acyclic database schemes. In *Proc. of the 7th Int. Conf. on Very Large Data Bases, VLDB'81*. IEEE Computer Society, 82–94.
[89] 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. *J. Artif. Intell. Res. (JAIR)* 54 (2015), 309–367.

Received April 2016; revised August 2017; accepted March 2018