Logical Methods in Computer Science
Volume 15, Issue 2, 2019, pp. 15:1–15:46
https://lmcs.episciences.org/

Submitted Jun. 06, 2018
Published May 23, 2019

REWRITABILITY IN MONADIC DISJUNCTIVE DATALOG, MMSNP, AND EXPRESSIVE DESCRIPTION LOGICS

CRISTINA FEIER [a,*], ANTTI KUUSISTO [b,*], AND CARSTEN LUTZ [b,*]

[a] University of Bremen, Department of Computer Science, Germany

[b] Tampere University, Mathematics, Finland

Abstract. We study rewritability of monadic disjunctive Datalog programs, (the complements of) MMSNP sentences, and ontology-mediated queries (OMQs) based on expressive description logics of the $\mathcal{ALC}$ family and on conjunctive queries. We show that rewritability into FO and into monadic Datalog (MDLog) are decidable, and that rewritability into Datalog is decidable when the original query satisfies a certain condition related to equality. We establish 2NExpTime-completeness for all studied problems except rewritability into MDLog for which there remains a gap between 2NExpTime and 3ExpTime. We also analyze the shape of rewritings, which in the case of MMSNP correspond to obstructions, and give a new construction of canonical Datalog programs that is more elementary than existing ones and also applies to non-Boolean queries.

1. Introduction

In data access with ontologies, the premier aim is to answer queries over incomplete and heterogeneous data while taking advantage of the domain knowledge provided by an ontology [BtCLW14, CDL$^+$09, BO15]. Since traditional database systems are often unaware of ontologies, it is common to rewrite the emerging ontology-mediated queries (OMQs) into more standard database query languages. For example, the DL-Lite family of description logics (DLs) was designed as an ontology language specifically so that any OMQ $Q = (\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is a DL-Lite ontology, Σ a data signature, and q a conjunctive query, can be rewritten into an equivalent first-order (FO) query that can then be executed using a standard SQL database system [CGL$^+$07, ACKZ09]. In more expressive ontology languages, it is not guaranteed that for every OMQ, there is an equivalent FO query. For example, this is the case for DLs of the $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ families and for DLs of the expressive $\mathcal{ALC}$ family; please see [BHLS17] for a general introduction to DLs. In many members of the $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ families, however, rewritability into monadic Datalog (MDLog) is guaranteed, thus enabling the use of Datalog engines for query answering. In $\mathcal{ALC}$ and above, not

Key words and phrases: MDDLog, MMSNP, OMQ, FO-Rewritability, Monadic Datalog-Rewritability.

* The current paper is the extended version of an invited conference abstract [FKL17]. It contains detailed proofs of all results and new material regarding dichotomies and deciding PTime query evaluation.

* The author was supported by ERC Consolidator Grant 647289 CODA.

DOI:10.23638/LMCS-15(2:15)2019

even Datalog-rewritability is generally ensured. Since ontologies emerging from practical applications tend to be structurally simple, though, there is reason to hope that (FO-, MDLog-, and Datalog-) rewritings do exist in many practically relevant cases even when the ontology is formulated in an expressive language. This has in fact been experimentally confirmed for FO-rewritability in the $\mathcal{EL}$ family of DLs [HLISW15], and it has led to the implementation of rewriting tools that, although incomplete, are able to compute rewritings in many practical cases [PUMH10, KNG14, TSCS15].

Fundamental problems that emerge from this situation are to understand the exact limits of rewritability and to provide (complete) algorithms that decide the rewritability of a given OMQ and that compute a rewriting when it exists. These problems have been adressed in [BLW13, HLISW15, BHLW16, LS17] for DLs from the $\mathcal{EL}$ and Horn-$\mathcal{ALC}$ families. For DLs from the $\mathcal{ALC}$ family, first results were obtained in [BtCLW14] where a connection between OMQs and constraint satisfaction problems (CSPs) was established and then used to transfer decidability results from CSPs to OMQs. In fact, rewritability is an important topic in CSP (where it is called definability) as it constitutes a central tool for analyzing the complexity of CSPs [FV98, LLT07, ELT07, DL08]. In particular, it is known that deciding the rewritability of (the complement of) a given CSP into FO and into Datalog is NP-complete [LLT07, Bar16, CL17] and rewritability into MDLog is NP-hard and in ExpTime [CL17]. In [BtCLW14], these results were used to show that FO- and Datalog-rewritability of OMQs $(\mathcal{T}, \Sigma, q)$ is decidable and in fact NExpTime-complete when $\mathcal{T}$ is formulated in $\mathcal{ALC}$ or a moderate extension thereof and q is an atomic query (AQ), that is, a monadic query of the simple form $A(x)$. For MDLog-rewritability, one can show NExpTime-hardness and containment in 2ExpTime.

The aim of this paper is to study the above questions for OMQs where the ontology is formulated in an expressive DL from the $\mathcal{ALC}$ family and where the actual query is a conjunctive query (CQ) or a union of conjunctive queries (UCQ). As observed in [BtCLW14], transitioning in OMQs from AQs to UCQs corresponds to the transition from CSP to its logical generalization MMSNP introduced by Feder and Vardi [FV98] and studied, for example, in [MS07, Mad09, Mad10, BCF12]. More precisely, while the OMQ language $(\mathcal{ALC}, \text{AQ})$ that consists of all OMQs $(\mathcal{T}, \Sigma, q)$ where $\mathcal{T}$ is formulated in $\mathcal{ALC}$ and q is an AQ has the same expressive power as the complement of CSP (with multiple templates and a single constant), the OMQ language $(\mathcal{ALC}, \text{UCQ})$ has the same expressive power as the complement of MMSNP (with free variables)—which in turn is essentially a notational variant of monadic disjunctive Datalog (MDDLog). It should be noted, however, that while all these formalisms are equivalent in expressive power, they differ significantly in succinctness [BtCLW14]; in particular, the best known translation of OMQs from $(\mathcal{ALC}, \text{UCQ})$ into MMSNP/MDDLog involves a double exponential blowup. In contrast to the CSP case, FO-, MDLog-, and Datalog-rewritability of (complemented) MMSNP sentences was not known to be decidable. In this paper, we establish decidability of FO- and MDLog-rewritability in $(\mathcal{ALC}, \text{UCQ})$ and related OMQ languages, in MDDLog, and in complemented MMSNP. We show that FO-rewritability is 2NExpTime-complete in all three cases, and that MDLog-rewritability is in 3ExpTime; a 2NExpTime lower bound was established in [BL16]. Let us discuss our results on the complexity of FO-rewritability from three different perspectives. From the OMQ perspective, the transition from AQs to UCQs results in an increase of complexity from NExpTime to 2NExpTime. From the monadic Datalog perspective, adding disjunction (transitioning from monadic Datalog to MDDLog) results in a moderate increase of complexity from 2ExpTime [BtCCV15] to 2NExpTime. And from the CSP

perspective, the transition from CSPs to MMSNP results in a rather dramatic complexity jump from NP to 2NExpTime.

For Datalog-rewritability, we obtain only partial results. In particular, we show that Datalog-rewritability is decidable and 2NExpTime-complete for MDDLog programs that, in a certain technical sense made precise in the paper, *have equality*. For the general case, we only obtain a potentially incomplete procedure. It is well possible that the procedure is in fact complete, but proving this remains an open issue for now. These results also apply to analogously defined classes of MMSNP sentences and OMQs that have equality.

While we mainly focus on deciding whether a rewriting exists rather than actually computing it, we also analyze the shape that rewritings can take. Since the shape turns out to be rather restricted, this is important information for algorithms (complete or incomplete) that seek to compute rewritings. In the CSP/MMSNP world, this corresponds to analyzing obstruction sets for MMSNP, in the style of CSP obstructions [Nes08, BKL08, Ats08] and not to be confused with colored forbidden patterns sometimes used to characterize MMSNP [MS07]. More precisely, we show that an OMQ $(\mathcal{T}, \Sigma, q)$ from $(\mathcal{ALC}, \text{UCQ})$ is FO-rewritable if and only if it is rewritable into a UCQ in which each CQ has treewidth $(1, \max\{2, n_q\})$, where n_q is the size of q;[1] similarly, the complement of an MMSNP sentence φ is FO-definable if and only if it admits a finite set of finite obstructions of treewidth $(1, k)$ where k is the diameter of φ (the maximum number of variables in a negated conjunction in its body, in Feder and Vardi's terminology). We also show that an OMQ $(\mathcal{T}, \Sigma, q)$ is MDLog-rewritable if and only if it is rewritable into an MDLog program of diameter $\max\{2, n_q\}$ where the diameter of an MDLog program is the maximum number of variables in a rule; equivalently, the complemented of an MMSNP sentence φ is MDLog-definable if and only if it admits a (potentially infinite) set of finite obstructions of treewidth $(1, k)$ where k is the diameter of φ. For the case of rewriting into Datalog, we give a new and direct construction of canonical Datalog-rewritings of MMSNP sentences. It has been observed in [FV98] that for every CSP and all ℓ, k, it is possible to construct a canonical Datalog program Π of width ℓ and diameter k (the width is the maximum arity of IDB relations) in the sense that if any such program is a rewriting of the CSP, then so is Π; moreover, even when there is no (ℓ, k)-Datalog rewriting, then Π is the best possible approximation of such a rewriting. The existence of canonical Datalog-rewritings for (complemented) MMSNP sentences was already known from [BD13]. However, the construction given there is quite complex, proceeding via an infinite template that is obtained by applying an intricate construction due to Cherlin, Shelah, and Shi [CSS99], and resulting in canonical programs that are rather difficult to understand and to analyze. In contrast, our construction is elementary and essentially parallels the CSP case; it also applies to MMSNP formulas with free variables, where the canonical program takes a rather special form that involves *parameters*, similar in spirit to the parameters to least fixed-point operators in FO(LFP) [BBV16].

Our main technical tool is the translation of MMSNP sentences into CSPs exhibited by Feder and Vardi [FV98]; actually, the target of the translation is a generalized CSP, meaning that there are multiple templates. The translation is not equivalence preserving and involves a double exponential blowup, but it was designed so as to preserve complexity up to polynomial time reductions. Here, we are primarily interested in the semantic relationship between the original MMSNP sentence and the constructed CSP. It turns out that the

[1] What we mean here is that q has a tree decomposition in which every bag has at most $\max\{2, n_q\}$ elements and in which neighboring bags overlap in at most one element.

translation does not quite preserve rewritability. In particular, when the original MMSNP sentence has a rewriting, then the natural way of constructing from it a rewriting for the CSP is sound only on instances of high girth. However, FO- and MDLog-rewritings of CSPs that are sound on high girth (and unconditionally complete) can be converted into rewritings that are unconditionally sound (and complete). The same is true for Datalog-rewritings when the CSP is derived from an MMSNP sentence that has equality, but it remains open whether it is true for Datalog-rewritings of unrestricted CSPs.

With our translations in place, we can also make relevant observations regarding the (data) complexity of query evaluation in MMSNP, in MDDLog, and of OMQs. This is especially interesting in the light of the recently obtained breakthrough in CSPs that there is a dichotomy between PTime and NP in the complexity of CSPs [Bul17, Zhu17] and that it is decidable and NP-complete whether a CSP is in PTime [CL17]. We show that this implies a dichotomy between PTime and coNP for MDDLog and for all OMQ languages mentioned above. We also prove that PTime query evaluation is decidable and 2NExpTime-complete in the mentioned query languages, and that the same is true for MMSNP.

The structure of this paper is as follows. In Section 2, we introduce the essentials of disjunctive Datalog and its relevant fragments as well as CSP and MMSNP; in fact, we shall always work with Boolean MDDLog rather than with complemented MMSNP. In Section 3, we summarize the main properties of Feder and Vardi's translation of MMSNP into CSP. We use this in Section 4 to show that FO- and MDLog-rewritability of Boolean MDDLog programs and of the complement of an MMSNP sentences is decidable, also establishing the announced complexity results. In Section 5, we analyze the shape of FO- and MDLog-rewritings and of obstructions for MMSNP sentences. We also establish an MMSNP analogue of an essential combinatorial lemma for CSPs which says that it is possible to replace a structure by a structure of high high girth while preserving certain homomorphisms; the MMSNP analogue achieves high 'decomposition girth' (defined in the paper) and preserves the truth of certain MMSNP sentences. In Section 6, we study Datalog-rewritability of MDDLog programs that have equality and construct canonical Datalog programs. Section 7 is concerned with lifting our results from the Boolean case to the general case, concerning the complexity of deciding rewritability, the shape of rewritings, and the construction of canonical Datalog programs. In this section, Datalog programs with parameters play a central role. In Section 8, we introduce OMQs and further lift our results to that setting, finally arriving at our goal to study fundamental rewritability questions for OMQ languages based on (unions of) conjunctive queries. Section 9 is then concerned with dichotomies and the complexity of deciding PTime query evaluation. We conclude in Section 10.

2. Preliminaries

We introduce disjunction Datalog, CSP, and MMSNP. To avoid overloading this section, the introduction of ontology languages and ontology-mediated queries is deferred to Section 8.

A *schema* is a finite collection $\mathbf{S} = (S_1, \dots, S_k)$ of relation symbols with associated arity. An $\mathbf{S}$-*fact* is an expression of the form $S(a_1, \dots, a_n)$ where $S \in \mathbf{S}$ is an n-ary relation symbol, and $a_1, \dots, a_n$ are elements of some fixed, countably infinite set $\mathtt{const}$ of *constants*. For an n-ary relation symbol S, $\mathtt{pos}(S)$ is $\{1, \dots, n\}$. An $\mathbf{S}$-*instance* I is a finite set of $\mathbf{S}$-facts. The *active domain* $\mathsf{dom}(I)$ of I is the set of all constants that occur in the facts in I. We use the symbols $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ to denote tuples of constants and, slightly abusing notation, write $\mathbf{a} \subseteq \mathtt{dom}(I)$ to mean that $\mathbf{a}$ is a tuple of constants from $\mathtt{dom}(I)$ when we do not want to be

precise about the length of the tuple. For an instance I and a schema $\mathbf{S}$, we write $I|_{\mathbf{S}}$ to denote the restriction of I to the relation symbols in $\mathbf{S}$.

A *tree decomposition* of an instance I is a pair $(T, (B_v)_{v\in V})$, where $T = (V, E)$ is an undirected tree and $(B_v)_{v\in V}$ is a family of subsets of $\mathsf{dom}(I)$ such that the following conditions are satisfied:

(1) for all $a \in \mathsf{dom}(I)$, $\{v \in V \mid a \in B_v\}$ is nonempty and connected in T;
(2) for every fact $R(a_1, \dots a_r)$ in I, there is a $v \in V$ such that $a_1, \dots, a_r \in B_v$.

Unlike in the traditional setup [FG06], we are interested in two parameters of tree decompositions instead of only one. We call $(T, (B_v)_{v\in V})$ an (ℓ, k)-*tree decomposition* if for all distinct $v, v' \in V$, $|B_v \cap B_{v'}| \leq \ell$ and $|B_v| \leq k$. An instance I *has treewidth* (ℓ, k) if it admits an (ℓ, k)-tree decomposition.

We now define the notion of girth. A finite structure I has a *cycle* of length $n > 0$ if there are distinct facts $R_0(\mathbf{a}_0), \dots, R_{n-1}(\mathbf{a}_{n-1}) \in I$ and positions $p_0, p'_0 \in \mathtt{pos}(R_0), \dots, p_{n-1}, p'_{n-1} \in \mathtt{pos}(R_{n-1})$ such that

- $p_i \neq p'_i$ for $0 \leq i < n$ and
- the constant in the p'_i-th position of $\mathbf{a}_i$ is identical to the constant in the p_{i+1}-th position of $\mathbf{a}_{i+1}$ for $0 \leq i < n$ and with $p_n := p_0$ and $p'_n := p'_0$.

The *girth* of I is the length of the shortest cycle in it and ω if I has no cycle (in which case we say that I is a *tree*).

A *constraint satisfaction problem (CSP)* is defined by an instance T over a schema $\mathbf{S}_E$, called *template*.[2] The problem associated with T, denoted $\mathrm{CSP}(T)$, is to decide whether an input instance I over $\mathbf{S}_E$ admits a homomorphism to T, denoted $I \to T$. We use $\mathrm{coCSP}(T)$ to denote the complement problem, that is, deciding whether $I \not\to T$. A *generalized CSP* is defined by a set of templates S over the same schema $\mathbf{S}_E$ and asks for a homomorphism from the input I to at least one of the templates $T \in S$, denoted $I \to S$. Note that a (generalized) CSP can be viewed as a Boolean query over $\mathbf{S}_E$-instances.

An *MMSNP sentence* θ over schema $\mathbf{S}_E$ has the form $\exists X_1 \cdots \exists X_n \forall x_1 \cdots \forall x_m \varphi$ with $X_1, \dots, X_n$ monadic second-order variables, $x_1, \dots, x_m$ first-order variables, and φ a conjunction of quantifier-free formulas of the form

$$\beta_1 \vee \cdots \vee \beta_n \leftarrow \alpha_1 \wedge \cdots \wedge \alpha_m$$

with $n, m \geq 0$ and where each α_i takes the form $X_i(x_j)$ or $R(\mathbf{x})$ with $R \in \mathbf{S}_E$ and each β_i takes the form $X_i(x_j)$. The *diameter* of θ is the maximum number of variables in some implication in φ. This presentation is syntactically different from, but semantically equivalent to the original definition from [FV98], which does not use the implication symbol and instead restricts the allowed polarities of atoms. Both forms can be interconverted in polynomial time, see [BtCLW14]. The semantics of MMSNP is the standard semantics of second-order logic. More information can be found, e.g., in [BCF12, BD13]. Note that, just like CSPs, MMSNP sentences can be viewed as Boolean queries.

A *conjunctive query (CQ)* takes the form $q(\mathbf{x}) = \exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ where φ is a conjunction of relational atoms and $\mathbf{x}$, $\mathbf{y}$ denote tuples of variables; the equality relation may be used. Whenever convenient, we shall confuse $q(\mathbf{x})$ with the set of atoms in φ. The variables in $\mathbf{x}$ are the *answer variables* in $q(\mathbf{x})$. The *arity* of $q(\mathbf{x})$ is the number of its answer variables

[2] Adopting Datalog terminology, we generally call the schema in which the data is formulated the *EDB schema* and denote it with $\mathbf{S}_E$.

and $q(\mathbf{x})$ is *Boolean* if it has arity zero. We say that $q(\mathbf{x})$ is *over* $\mathbf{S}_E$ if φ uses only relation symbols from $\mathbf{S}_E$. An *answer* to q on an $\mathbf{S}_E$-instance I is a tuple of constants $\mathbf{a}$ such that $I \models q(\mathbf{a})$ in the standard sense of first-order logic. It is well-known that $I \models q(\mathbf{a})$ if and only if there is a homomorphism from φ viewed as an instance to I that takes $\mathbf{x}$ to $\mathbf{a}$. A Boolean CQ q is *true* on an instance I, denoted $I \models q$, if the empty tuple is an answer to q on I. A CQ q is a *contraction* of a CQ q' if it can be obtained from q' by identifying variables. A *union of conjunctive queries (UCQ)* is a disjunction of CQs with the same answer variables. The semantics of UCQs is defined in the expected way.

A *disjunctive Datalog rule* ρ has the form

$$S_1(\mathbf{x}_1) \vee \cdots \vee S_m(\mathbf{x}_m) \leftarrow R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$$

with $n > 0$ and $m \geq 0$. We refer to $S_1(\mathbf{x}_1) \vee \cdots \vee S_m(\mathbf{x}_m)$ as the *head* of ρ, and to $R_1(\mathbf{y}_1) \wedge \cdots \wedge R_n(\mathbf{y}_n)$ as the *body*. Every variable that occurs in the head of ρ is required to also occur in the body of ρ. A *disjunctive Datalog (DDLog) program* Π is a finite set of disjunctive Datalog rules with a selected *goal relation* $\texttt{goal}$ that does not occur in rule bodies and appears only in non-disjunctive *goal rules* $\texttt{goal}(\mathbf{x}) \leftarrow R_1(\mathbf{x}_1) \wedge \cdots \wedge R_n(\mathbf{x}_n)$. The *arity of* Π is the arity of the $\texttt{goal}$ relation; we say that Π is *Boolean* if it has arity zero. Relation symbols that occur in the head of at least one rule of Π are *intensional (IDB) relations*, and all remaining relation symbols in Π are *extensional (EDB) relations*. Note that, by definition, $\texttt{goal}$ is an IDB relation. When all relations in Π are from schema $\mathbf{S}_E$, then we say that Π is *over EDB schema* $\mathbf{S}_E$. The *IDB schema* of Π is the set of all IDB relations in Π.

We will sometimes use body atoms of the form $\texttt{true}(x)$ that are vacuously true for all elements of the active domain. This is just syntactic sugar since any rule with body atom $\texttt{true}(x)$ can equivalently be replaced by a set of rules obtained by replacing $\texttt{true}(x)$ in all possible ways with an atom $R(x_1, \ldots, x_n)$ where R is a relation symbol from $\mathbf{S}_E$ and where $x_i = x$ for some i and all other x_i are fresh variables.

A DDLog program is called *monadic* or an *MDDLog program* if all its IDB relations with the possible exception of $\texttt{goal}$ have arity at most one. The *size* of a DDLog program Π is the number of symbols needed to write it (where relation symbols and variable names count one), its *width* is the maximum arity of non-$\texttt{goal}$ IDB relations used in it, and its *diameter* is the maximum number of variables that occur in a rule in Π.

A *Datalog rule* is a disjunctive Datalog rule in which the rule head contains exactly one disjunct. Datalog (DLog) programs and monadic Datalog (MDLog) programs are then defined in the expected way. We call a Datalog program an (ℓ, k)-*Datalog program* if its width is ℓ and its diameter is k.

For Π an n-ary DDLog program over schema $\mathbf{S}_E$, an $\mathbf{S}_E$-instance I, and $a_1, \ldots, a_n \in \mathsf{dom}(I)$, we write $I \models \Pi(a_1, \ldots, a_n)$ if $\Pi \cup I \models \texttt{goal}(a_1, \ldots, a_n)$ where the variables in all rules of Π are universally quantified and thus Π is a set of first-order (FO) sentences; please see [AHV95, EGM97] for more information on the semantics of (disjunctive) Datalog. A query $q(\mathbf{x})$ over $\mathbf{S}_E$ of arity n is

- *sound for* Π if for all $\mathbf{S}_E$-instances I and $\mathbf{a} \subseteq \mathsf{dom}(I)$, $I \models q(\mathbf{a})$ implies $I \models \Pi(\mathbf{a})$;
- *complete for* Π if for all $\mathbf{S}_E$-instances I and $\mathbf{a} \subseteq \mathsf{dom}(I)$, $I \models \Pi(\mathbf{a})$ implies $I \models q(\mathbf{a})$;
- a *rewriting of* Π if it is sound for Π and complete for Π.

Note that Boolean programs are also covered by the above definitions. To additionally specify the syntactic shape of $q(\mathbf{x})$, we speak of a UCQ-rewriting, an MDLog-rewriting, and so on. An *FO-rewriting* takes the form of an FO-query that uses only relation symbols from

the relevant EDB schema and possibly equality, but neither constants nor function symbols. We say that an MDDLog program Π is *FO-rewritable* if there is an FO-rewriting of Π, and likewise for *UCQ-rewritability* and for *MDLog-rewritability*. Since a generalized CSP defined by a set of templates S can be viewed as a Boolean query, we can also speak of a query to be sound and complete for respectively a rewriting of coCSP(S). The definitions are as expected, paralleling the ones above.

It was shown in [BtCLW14] that the complement of an MMSNP sentence can be translated into an equivalent Boolean MDDLog program in polynomial time and vice versa; moreover, the transformations preserve diameter and all other parameters relevant for this paper. From now on, we will thus not explicitly distinguish between Boolean MDDLog and (complemented) MMSNP.

3. MDDLog, Simple MDDLog and CSP

Feder and Vardi show how to translate an MMSNP sentence into (the complemen of) a generalized CSP that has the same complexity up to polynomial time reductions [FV98]. The resulting CSP has a different schema than the original MMSNP sentence and is thus not equivalent to it. We are going to make use of this translation to reduce rewritability problems for MDDLog to the corresponding problems for CSPs. Consequently, our main interest is in the precise semantic relationship between the MMSNP sentence and the constructed CSP, rather than in their complexity. In this section, we sum up the properties of the results obtained in [FV98] that are relevant for us. These properties are all we need in later sections, that is, we do not need to go further into the details of the translation. For the reader's convenience and information, we still describe the translations in full detail in Appendix A; these are based on the presentation given in [BL16], which is more detailed than the original presentation in [FV98].

Let $\mathbf{S}_E$ be a schema. A schema $\mathbf{S}'_E$ is a *k-aggregation schema* for $\mathbf{S}_E$ if its relations have the form $R_{q(\mathbf{x})}$ where $q(\mathbf{x})$ is a CQ over $\mathbf{S}_E$ without quantified variables and the arity of $R_{q(\mathbf{x})}$ is identical to the length of $\mathbf{x}$, which is at most k. The generalized CSP to be constructed later makes use of a schema of this form. What is important at this point is that there are natural translations of instances between the two schemas. To make this precise, let I be an $\mathbf{S}_E$-instance. The *corresponding $\mathbf{S}'_E$-instance* I' consists of all facts $R_{q(\mathbf{x})}(\mathbf{a})$ such that $I \models q(\mathbf{a})$. Conversely, let I' be an $\mathbf{S}'_E$-instance. The *corresponding $\mathbf{S}_E$-instance* I consists of all facts $S(\mathbf{b})$ such that $R_{q(\mathbf{x})}(\mathbf{a}) \in I'$ and $S(\mathbf{b})$ is a conjunct of $q(\mathbf{a})$.

Example 3.1. Let $\mathbf{S}_E = \{r\}$ with r a binary relation symbol,

$$q(x_1, x_2, x_3) = r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1),$$

and let $\mathbf{S}'_E$ consist of $R_{q(\mathbf{x})}$ where $\mathbf{x} = (x_1, x_2, x_3)$ for brevity. Take the $\mathbf{S}'_E$-instance I' defined by

$$R_{q(\mathbf{x})}(a, a', c'), R_{q(\mathbf{x})}(b, b', a'), R_{q(\mathbf{x})}(c, c', b').$$

The corresponding $\mathbf{S}_E$-instance I is

$$r(a, a'), r(a', c'), r(c', a), r(b, b'), r(b', a'), r(a', b), r(c, c'), r(c', b), r(b', c).$$

Note that when we transition from $\mathbf{S}_E$ back to $\mathbf{S}'_E$ and take the $\mathbf{S}'_E$-instance I'' corresponding to I, we do *not* obtain I', but rather a strict superset that contains additional facts such as $R_{q(\mathbf{x})}(c', b', a')$. This is illustrated in Figure 1 which shows the instances I, I', and a subset of I'' that contains all facts from I plus one additional fact.

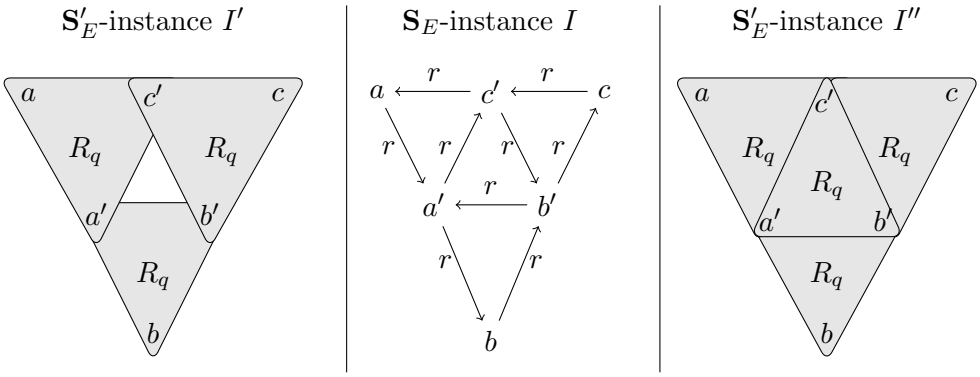

Figure 1: Translating an $\mathbf{S}'_E$-instance into an $\mathbf{S}_E$-instance and back.

The translation in [FV98] consists of two steps. We describe them here using Boolean MDDLog instead of (complemented) MMSNP. The first step is to transform the given Boolean MDDLog program Π into a Boolean MDDLog program Π_S over a suitable aggregation schema $\mathbf{S}'_E$ such that Π_S is of a restricted syntactic form called *simple*. In the second step, one transforms Π_S into a generalized CSP whose complement is equivalent to Π_S.

We start with summing up the important aspects of the first step. A Boolean MDDLog program Π_S is *simple* if it satisfies the following conditions:

(1) every rule in Π_S contains at most one EDB atom and this atom contains all variables of the rule body, each variable exactly once;
(2) rules without an EDB atom contain at most a single variable.

Now, the first step achieves the following.

Theorem 3.2 [FV98]**.** *Given a Boolean MDDLog program Π over EDB schema $\mathbf{S}_E$ of diameter k, one can construct a simple Boolean MDDLog program Π_S over a k-aggregation EDB schema $\mathbf{S}'_E$ for $\mathbf{S}_E$ and IDB schema $\mathbf{S}'_I$ such that*

(1) *If I is an $\mathbf{S}_E$-instance and I' the corresponding $\mathbf{S}'_E$-instance, then $I \models \Pi$ iff $I' \models \Pi_S$;*
(2) *If I' is an $\mathbf{S}'_E$-instance and I the corresponding $\mathbf{S}_E$-instance, then*
   (a) *$I' \models \Pi_S$ implies $I \models \Pi$;*
   (b) *$I \models \Pi$ implies $I' \models \Pi_S$ if the girth of I' exceeds k.*

If Π is of size n, then the size of Π_S and the cardinality of $\mathbf{S}'_E \cup \mathbf{S}'_I$ are bounded by $2^{p(k \cdot \log n)}$, where p is a polynomial. The construction takes time polynomial in the size of Π_S.

The translation underlying Theorem 3.2 consists of three steps itself: first saturate Π by adding all rules that can be obtained as a *contraction* of a rule in Π, that is, by identifying variables in the rule body and head in a consistent way. Then rewrite Π in an equivalence-preserving way so that all rule bodies are biconnected, introducing fresh unary and nullary IDB relations as needed. And finally replace the conjunction $q(\mathbf{x})$ of all EDB atoms in each rule body with a single EDB atom $R_{q(\mathbf{x})}(\mathbf{x})$, additionally taking care of interactions between the new EDB relations that arise e.g. when we have two relations $R_{q(\mathbf{x})}$ and $R_{p(\mathbf{x})}$ such that $q(\mathbf{x})$ is contained in $p(\mathbf{x})$ in the sense of query containment. Details are in Appendix A.1.

The following theorem summarizes the second step of the translation of Boolean MDDLog into a generalized CSP.

Theorem 3.3 [FV98]**.** *Let* Π *be a simple Boolean MDDLog program over EDB schema* $\mathbf{S}_E$ *and with IDB schema* $\mathbf{S}_I$*, m the maximum arity of relations in* $\mathbf{S}_E$*. Then there exists a set of templates* S_Π *over* $\mathbf{S}_E$ *such that*

(1) Π *is equivalent to coCSP*(S_Π)*;*
(2) $|S_\Pi| \leq 2^{|\mathbf{S}_I|}$ *and* $|T| \leq |\mathbf{S}_E| \cdot 2^{m|\mathbf{S}_I|}$ *for each* $T \in S_\Pi$*;*

The construction takes time polynomial in $\sum_{T \in S_\Pi} |T|$.

We again sketch the idea underlying the proof of the theorem. The desired set of templates S_Π contains one template for every 0-type, that is, for every set of nullary IDB relations in Π that does not contain `goal`() and that satisfies all rules in Π which use only nullary IDBs. Each template contains one constant c_M for every 1-type M, that is, for every set M of unary IDBs that agrees on nullary IDBs with the 0-type for which the template was constructed and that satisfy all rules in Π which use only IDB relations that are at most unary. One then interprets all EDB relations in a maximal way so that all rules in Π are satisfied. The fact that Π is simple implies that no choices arise, that is, there is only one maximal interpretation of each EDB relation and the interpretations of different such relations do not interact. Details are given in Appendix A.2.

4. FO- and MDLog-Rewritability of Boolean MDDLog Programs

We exploit the translations described in the previous section and the known results that FO-rewritability of CSPs and MDLog-rewritability of coCSPs are decidable to obtain analogous results for Boolean MDDLog, and thus also for MMSNP. In the case of FO-rewritability, we obtain tight 2NExpTime complexity bounds. For MDLog-rewritability, the exact complexity remains open (as in the CSP case), between 2NExpTime and 3ExpTime.

We start with observing that FO-rewritability and MDLog-rewritability are more closely related than one might think at first glance. Recall that, by Rossman's homomorphism preservation theorem [Ros08], a first-order formula is preserved under homomorphisms on finite structures if and only if it is equivalent to a UCQ. While every MDLog-rewriting can be viewed as an infinitary UCQ-rewriting, Rossman's result implies that FO-rewritability of a Boolean MDDLog program coincides with (finitary) UCQ-rewritability. The latter is true also in the non-Boolean case.

Proposition 4.1. *Let* Π *be an MDDLog program. Then* Π *is FO-rewritable iff* Π *is UCQ-rewritable.*

Proof. It is well known and easy to show that truth of disjunctive Datalog programs is preserved under homomorphisms. Thus, the proposition immediately follows from Rossman's theorem in the Boolean case. For the non-Boolean case, we observe that Rossman establishes his result also in the presence of constants. Let Π be an MDDLog program and $\varphi(\mathbf{x})$ a rewriting of Π. We can apply Rossman's result to $\varphi(\mathbf{a})$, where $\mathbf{a}$ is a tuple of constants of the same length as $\mathbf{x}$, obtaining a UCQ $q(\mathbf{a})$ equivalent to $\varphi(\mathbf{a})$. Let $q(\mathbf{x})$ be obtained from $q(\mathbf{a})$ by replacing the constants in $\mathbf{a}$ with the variables from $\mathbf{x}$. It can be verified that $q(\mathbf{x})$ is a rewriting of Π. □

For utilizing the translation of Boolean MDDLog programs to generalized CSPs in the intended way, the interesting aspect is to deal with the translation of a Boolean MDDLog program Π into a simple program Π_S stated in Theorem 3.2, since it is not equivalence

preserving. The following proposition relates rewritings of Π to rewritings of Π_S. It also applies to Datalog-rewritings, which we will make use of in Section 6.

Lemma 4.2. *Let Π be a Boolean MDDLog program of diameter k, Π_S as in Theorem 3.2, and $\mathcal{Q} \in \{\mathit{UCQ}, \mathit{MDLog}, \mathit{DLog}\}$. Then*

(1) *every $\mathcal{Q}$-rewriting of Π_S can effectively be converted into a $\mathcal{Q}$-rewriting of Π;*
(2) *every $\mathcal{Q}$-rewriting of Π can effectively be converted into a $\mathcal{Q}$-rewriting of Π_S that is (i) sound on instances of girth exceeding k and (ii) complete.*

Proof. Let $\mathbf{S}_E$ and $\mathbf{S}'_E$ be the EDB schema of Π and of Π_S, respectively. We start with the case $\mathcal{Q} = \text{UCQ}$.

For Point 1, let q_{Π_S} be a UCQ-rewriting of Π_S. Let q_Π be the UCQ obtained from q_{Π_S} by replacing every atom $R_{q(\mathbf{x})}(\mathbf{y})$ with $q[\mathbf{y}/\mathbf{x}]$, that is, with the result of replacing the variables $\mathbf{x}$ in $q(\mathbf{x})$ with the variables $\mathbf{y}$ (which may lead to identifications). We show that q_Π is as required. Let I be an $\mathbf{S}_E$-instance and I' the corresponding $\mathbf{S}'_E$-instance. Then we have $I \models \Pi$ iff $I' \models \Pi_S$ (by Point 1 of Theorem 3.2) iff $I' \models q_{\Pi_S}$ (by choice of q_{Π_S}) iff $I \models q_\Pi$ (by construction of I' and of q_Π). Let us expand on the latter.

First assume that $I' \models q_{\Pi_S}$. Then there is a CQ q in q_{Π_S} and a homomorphism h from q to I'. By construction, q_Π contains a CQ q' that is obtained from q by replacing every atom $R_{q(\mathbf{x})}(\mathbf{y}) \in q$ with $q[\mathbf{y}/\mathbf{x}]$. Clearly, for every atom $R_{q(\mathbf{x})}(\mathbf{y}) \in q$, we must have $R_{q(x)}(h(\mathbf{y})) \in I'$. The construction of I' yields $q(h(\mathbf{y})) \subseteq I$. Consequently, h is also a homomorphism from q' to I. Conversely, assume that there is a CQ q' in q_Π and a homomorphism h from q' to I. Then there is a CQ q in q_{Π_S} from which q' was obtained by the described replacement operation. For every atom $R_{q(\mathbf{x})}(\mathbf{y}) \in q$, we must have $q(h(\mathbf{y})) \subseteq I$. We obtain $R_{q(\mathbf{x})}(h(\mathbf{y})) \in q$ and thus h is a homomorphism from q to I'.

For Point 2, let q_Π be a UCQ-rewriting of Π. The UCQ q_{Π_S} consists of all CQs that can be obtained as follows:

(1) choose a CQ $\exists \mathbf{x}\, q(\mathbf{x})$ from q_Π, a contraction $\exists \mathbf{x}'\, q'(\mathbf{x}')$ of $\exists \mathbf{x}\, q(\mathbf{x})$, and a partition $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$ of $q'(\mathbf{x}')$;
(2) for each $i \in \{1, \dots, n\}$, choose a relation $R_{p(\mathbf{z})}$ from $\mathbf{S}'_E$ and a tuple $\mathbf{y}$ of $|\mathbf{z}|$ variables (repeated occurrences allowed) that are either from $\mathbf{x}_i$ or do not occur in $\mathbf{x}'$ such that $q_i(\mathbf{x}_i) \subseteq p[\mathbf{y}/\mathbf{z}]$; then replace $q_i(\mathbf{x}_i)$ in $\exists \mathbf{x}'\, q'(\mathbf{x}')$ with the single atom $R_{p(\mathbf{z})}(\mathbf{y})$.

To establish that q_{Π_S} is as desired, we show that for every $\mathbf{S}'_E$-instance I'

(I) $I' \models q_{\Pi_S}$ implies $I' \models \Pi_S$ if I' is of girth exceeding k (soundness) and
(II) $I' \models \Pi_S$ implies $I' \models q_{\Pi_S}$ (completeness).

Let I be the $\mathbf{S}_E$-instance corresponding to I'.

For Point (I), we observe that $I' \models q_{\Pi_S}$ implies $I \models q_\Pi$ (by construction of q_{Π_S} and of I') implies $I \models \Pi$ (by choice of q_Π) implies $I' \models \Pi_S$ (by Point 2b of Theorem 3.2 and if I' is of girth exceeding k). Let us zoom into the first implication. Assume that $I' \models q_{\Pi_S}$. Then there is a CQ $\exists \mathbf{u}\, p_0(\mathbf{u})$ in q_{Π_S} and a homomorphism h from $p_0(\mathbf{u})$ to I'. There must be some CQ $\exists \mathbf{x}\, q(\mathbf{x})$ in q_Π from which $\exists \mathbf{u}\, p_0(\mathbf{u})$ has been constructed in Steps 1 and 2 above. Let $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$ be as in this construction. It suffices to show that h is a homomorphism from $q_i(\mathbf{x}_i)$ to I, for each i. Thus fix a $q_i(\mathbf{x}_i)$. Then there is a relation $R_{p(\mathbf{z})} \in \mathbf{S}'_E$ and a tuple $\mathbf{y}$ of variables that are either from $\mathbf{x}_i$ or do not occur in $\mathbf{x}'$ such that $q_i(\mathbf{x}_i) \subseteq p[\mathbf{y}/\mathbf{z}]$ and $R_{p(\mathbf{z})}(\mathbf{y}) \in p_0(\mathbf{u})$. Thus $R_{p(\mathbf{z})}(h(\mathbf{y})) \in I'$. By construction of I', this yields $q_i(h(\mathbf{x}_i)) \subseteq I$ and thus we are done.

For Point (II), we have that $I' \models \Pi_S$ implies $I \models \Pi$ (by Point 2a of Theorem 3.2) implies $I \models q_\Pi$ (by choice of q_Π). It thus remains to show that $I \models q_\Pi$ implies $I' \models q_{\Pi_S}$. Thus assume that there is a CQ $\exists \mathbf{x}\, q(\mathbf{x})$ in q_Π and a homomorphism h from $q(\mathbf{x})$ to I. We use $\exists \mathbf{x}\, q(\mathbf{x})$ and h to guide the choices in Step 1 and Step 2 of the construction of CQs in q_{Π_S} to exhibit a CQ p_0 in q_{Π_S} such that $p_0 \to I'$.

We start with Step 1. As $\exists \mathbf{x}'\, q'(\mathbf{x}')$, we use the contraction of $\exists \mathbf{x}\, q(\mathbf{x})$ obtained by identifying variables x and y whenever $h(x) = h(y)$. Thus, h is an injective homomorphism from $q'(\mathbf{x}')$ to I. We next need to choose a partition of $q'(\mathbf{x}')$. For every fact $R(\mathbf{a}) \in I$, choose a fact $R_{p(\mathbf{x}_0)}(\mathbf{b}) \in I'$ that $R(\mathbf{a})$ was obtained from during the construction of I and denote this fact with $\mu(R(\mathbf{a}))$. Now let $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$ be the partition of $q'(\mathbf{x}')$ obtained by grouping together two atoms $R_1(\mathbf{y}_1)$ and $R_2(\mathbf{y}_2)$ if and only if $\mu(R_1(h(\mathbf{y}_1))) = \mu(R_2(h(\mathbf{y}_2)))$. Let $\mu(q_i)$ denote the (unique) value of μ for all the atoms in $q_i(\mathbf{x}_i)$.

Step 2 deals with each query $q_i(\mathbf{x}_i)$ separately. We choose the relation $R_{p(\mathbf{z})}$ from $\mu(q_i) = R_{p(\mathbf{z})}(\mathbf{b})$, which clearly is in $\mathbf{S}'_E$. We choose the tuple $\mathbf{y}$ of variables based on the tuple of individuals $\mathbf{b}$. Let $\mathbf{b} = b_1, \dots, b_n$. Then the ℓ-th variable in $\mathbf{y}$ is y if $h(y) = b_\ell$ (which is well-defined since h is injective) and a fresh variable if there is no such y. This finishes the guiding process and thus gives rise to a query $p_0(\mathbf{u})$ in q_{Π_S}.

It remains to argue that h can be extended to a homomorphism h' from $p(\mathbf{u})$ to I'. Take a $q_i(\mathbf{x}_i)$ and consider the corresponding atom $R_{p(\mathbf{z})}(\mathbf{y})$ in p_0. Then all the facts in $q_i(h(\mathbf{x})) \subseteq I$ were obtained from the fact $\mu(q_i) = R_{p(\mathbf{z})}(\mathbf{b}) \in I'$ during the construction of I. By construction of $\mathbf{y}$ from $\mathbf{b}$, we can extend h to the fresh variables in $\mathbf{y}$ so that $h(\mathbf{y}) = \mathbf{b}$ and thus $R_{p(\mathbf{z})}(h(\mathbf{y})) \in I'$. Doing this for all q_i yields the desired h'.

Now for the cases $\mathcal{Q} \in \{\text{MDLog}, \text{DLog}\}$. We treat these cases in one since our construction preserves the width of Datalog-rewritings. In fact, this construction is very similar to the case $\mathcal{Q} = \text{UCQ}$, so we only give a sketch.

For Point 1, let Γ_{Π_S} be a Datalog-rewriting of Π_S. We construct a Datalog program Γ_Π of the same width over EDB schema $\mathbf{S}_E$ by modifying the EDB part of each rule body in the same way in which we had modified the UCQ-rewriting q_{Π_S} in the case $\mathcal{Q} = \text{UCQ}$: replace every EDB-atom $R_{q(\mathbf{x})}(\mathbf{y})$ with $q[\mathbf{y}/\mathbf{x}]$. We then have $I \models \Pi$ iff $I' \models \Pi_S$ (by Point 1 of Theorem 3.2) iff $I' \models \Gamma_{\Pi_S}$ (by choice of Γ_{Π_S}) iff $I \models \Gamma_\Pi$. The latter is by construction of I' and of Γ_Π. To prove it in more detail, it suffices to show that for every extension J of I to the IDB relations in Γ_{Π_S} with corresponding extension J' of I', and every rule body q in Γ_{Π_S} which was translated into a rule body q' in Γ_Π, we have $q \to J$ iff $q' \to J'$. The arguments needed are as in the case $\mathcal{Q} = \text{UCQ}$.

The proof of Point 2 can be adapted from UCQs to Datalog in an analogous way. Let Γ_Π be a Datalog-rewriting of Π. We construct a Datalog program Γ_{Π_S} of the same width over EDB schema $\mathbf{S}_{E'}$. The rules in Γ_{Π_S} are obtained by taking a rule

$$P_0(\mathbf{x}_0) \leftarrow P_1(\mathbf{x}_1) \wedge \dots \wedge P_n(\mathbf{x}_n) \wedge q(\mathbf{y})$$

from Γ_Π, where the P_i are IDB and $q(\mathbf{y})$ is a CQ over schema $\mathbf{S}_E$, converting $q(\mathbf{y})$ into a CQ $q'(\mathbf{y}')$ over schema $\mathbf{S}'_E$ in two steps, in the same way in which a CQ over $\mathbf{S}_E$ was converted into a CQ over $\mathbf{S}'_E$ in the case $\mathcal{Q} = \text{UCQ}$, and then including in Γ_{Π_S} the rule

$$P_0(\mathbf{x}_0) \leftarrow P_1(\mathbf{x}_1) \wedge \dots \wedge P_n(\mathbf{x}_n) \wedge q'(\mathbf{y}').$$

The crucial step in the correctness proof is to show that $I \models \Gamma_\Pi$ implies $I' \models \Gamma_{\Pi_S}$ for any $\mathbf{S}'_E$-instance I' and corresponding $\mathbf{S}_E$-instance I. The arguments are again the same as in

the case $\mathcal{Q} = \text{UCQ}$, the main difference being that we need to consider extensions of I and I' to IDB relations from Γ_Π instead of working with I and I' themselves. □

Point 2 of Lemma 4.2 only yields a rewriting of Π_S on $\mathbf{S}'_E$-instances of high girth. We next show that, for $\mathcal{Q} \in \{\text{UCQ}, \text{MDLog}\}$, the existence of a $\mathcal{Q}$-rewriting on instances of high girth implies the existence of a $\mathcal{Q}$-rewriting that works on instances of unrestricted girth. Whether the same is true for $\mathcal{Q} = \text{Datalog}$ remains as an open problem. We need the following well-known lemma that goes back to Erdös and was adapted to CSPs by Feder and Vardi. Informally, it says that every instance can be 'exploded' into an instance of high girth that behaves similarly regarding homomorphisms.

Lemma 4.3. *For every instance I and $g, s \geq 0$, there is an instance I' (over the same schema) such that $I' \to I$, I' has girth exceeding g, and for every instance T of size at most s, we have $I \to T$ iff $I' \to T$.*

Feder and Vardi additionally show that I' can be constructed by a randomized polynomial time reduction that was later derandomized by Kun [Kun13], but here we do not rely on such computational properties. Every CQ q can be viewed as an instance I_q by using the variables as constants and the atoms as facts. It thus makes sense to speak about tree decompositions of CQs and about their treewidth, and it is clear what we mean by saying that a CQ is a tree (that is, has girth ω).

Lemma 4.4. *Let S be a set of templates over schema $\mathbf{S}_E$, $g \geq 0$, and $\mathcal{Q} \in \{\text{UCQ}, \text{MDLog}\}$. If coCSP($S$) is $\mathcal{Q}$-rewritable on instances of girth exceeding g, then it is $\mathcal{Q}$-rewritable.*

Proof. We start with $\mathcal{Q} = \text{UCQ}$. Let q_g be a UCQ that defines coCSP(S) on instances of girth exceeding g, and let q be the UCQ that consists of all contractions of a CQ in q_g that are a tree CQ. We show that q defines coCSP(S) on unrestricted $\mathbf{S}_E$-instances.

Let I be an $\mathbf{S}_E$-instance. First assume that $I \not\to S$. By Lemma 4.3, there is an $\mathbf{S}_E$-instance I' of girth exceeding g and also exceeding the number of variables in each CQ in q_g and satisfying $I' \to I$ and $I' \not\to S$. Thus $I' \models q_g$, that is, there is a CQ q' in q_g and a homomorphism h from q' to I'. Let q'' be the contraction of q' obtained by identifying variables x and y if $h(x) = h(y)$. Thus, h is an injective homomorphism from q'' to I'. Since the girth of I' exceeds the number of variables in q'', q'' must be a tree. Consequently, q'' is a CQ in q and we have $I' \models q$. From $I' \to I$, we obtain $I \models q$.

Now assume that $I \models q$. Then, there is a tree CQ q' in q such that $q' \to I$. When we view q' as an $\mathbf{S}_E$-instance $I_{q'}$, then clearly $I_{q'} \models q_g$ and $I_{q'}$ has girth exceeding k. Thus, $q' \not\to S$, and from $q' \to I$ we obtain $I \not\to S$.

Now for the case $\mathcal{Q} = \text{MDLog}$. Let Γ_g be an MDLog program that defines coCSP(S) on instances of girth exceeding g. Let Γ be the program obtained from Γ_g by replacing every rule $P(x) \leftarrow q(\mathbf{x})$ with all rules $P(x) \leftarrow q'(\mathbf{x}')$ such that $q'(\mathbf{x}')$ is a tree CQ that is a contraction of $q(\mathbf{x})$. We show that Γ is an MDLog-definition of coCSP(S) on instances of unrestricted girth.

Let I be an $\mathbf{S}_E$-instance. First assume that $I \not\to S$. By Lemma 4.3, there is an $\mathbf{S}_E$-instance I' whose girth exceeds g and also exceeds the diameter of Γ_g and that satisfies $I' \to I$ and $I' \not\to S$. The latter yields $I' \models \Gamma_g$. It remains to show that this implies $I' \models \Gamma$ since with $I' \to I$, this yields $I \models \Gamma$ as required.

To show that $I' \models \Gamma$ follows from $I' \models \Gamma_g$, it suffices to show that all IDB facts derived by Γ_g starting from I' are also derived by Γ. Thus let J' be an extension of I' to the IDBs in Γ_g. It is enough to show that when a single application of a rule from Γ_g in J' yields

an IDB atom $P(a)$, then Γ can derive the same atom. The former is the case only if Γ_g contains a rule $P(x) \leftarrow q(\mathbf{x})$ such that there is a homomorphism h from $q(\mathbf{x})$ to J' with $h(x) = a$. Let $q'(\mathbf{x}')$ be the contraction of $q(\mathbf{x})$ obtained by identifying variables x and y when $h(x) = h(y)$. Since the girth of I' exceeds the diameter of Γ_g, $q'(\mathbf{x}')$ is a tree. Thus, Γ contains the rule $P(x) \leftarrow q'(\mathbf{x}')$ and the application of this rule in J' enabled by h yields $P(a)$. We have thus shown $I' \models \Gamma$ and are done.

Now assume that $I \models \Gamma_g$. Then there is a proof tree for $\texttt{goal}()$ from I and Γ_g, see [AHV95] for details. From that tree, we can read off an $\mathbf{S}_E$-instance O such that $O \to I$, $O \models \Gamma_g$, and, since Γ_g is monadic and only comprises rules with tree-shaped bodies, O is a tree. Thus, O has girth exceeding g and from $O \models \Gamma_g$ we get $O \not\to S$. But with $O \to I$, this yields $I \not\to S$ as required. □

Putting together Theorems 3.2 and 3.3, Proposition 4.1, and Lemmas 4.2 and 4.4, we obtain the following reductions of rewritability of Boolean MDDLog programs to CSP rewritability.

Proposition 4.5. *Every Boolean MDDLog program* Π *can be converted into a set of templates* S_Π *such that*

(1) Π *is* $\mathcal{Q}$*-rewritable iff coCSP*(S_Π) *is* $\mathcal{Q}$*-rewritable for every* $\mathcal{Q} \in \{FO, UCQ, MDLog\}$*;*
(2) *every* $\mathcal{Q}$*-rewriting of* Π *can be effectively translated into a* $\mathcal{Q}$*-rewriting of coCSP*(S_Π) *and vice versa, for every* $\mathcal{Q} \in \{UCQ, MDLog\}$*.*
(3) $|S_\Pi| \leq 2^{2^{p(n)}}$ *and* $|T| \leq 2^{2^{p(n)}}$ *for each* $T \in S_\Pi$*,* n *the size of* Π *and* p *a polynomial.*

The construction takes time polynomial in $\sum_{T \in S_\Pi} |T|$*.*

FO-rewritability of CSPs (and their complements) is NP-complete [LLT07] and it was observed in [BtCLW14] that the upper bound lifts to generalized CSPs. MDLog-rewritability of coCSPs is NP-hard and in ExpTime [CL17]. We show in Appendix B that also this upper bound lifts to generalized coCSPs. Together with Proposition 4.5, this yields the upper bounds in the following theorem. The lower bounds are from [BL16].

Theorem 4.6. *For Boolean MDDLog programs and the complement of MMSNP sentences,*

(1) *FO-rewritability (equivalently: UCQ-rewritability) is 2*NExpTime*-complete;*
(2) *MDLog-rewritability is in 3*ExpTime *(and 2*NExpTime*-hard).*

5. Shape of Rewritings, Obstructions, Explosion

In the FO case, it is possible to extract from the approach in the previous section an algorithm that computes actual rewritings, if they exist. However, that algorithm is hardly practical. An important first step towards the design of more practical algorithms that compute rewritings (in an exact or in an approximative way) is to analyze the shape of rewritings. In fact, both FO- and MDLog-rewritings of coCSPs are known to be of a rather restricted shape, far from exploiting the full expressive power of the target languages. In this section, we establish corresponding results for Boolean MDDLog. This topic is closely related to the theory of obstructions, so we also establish connections between the rewritability of MMSNP sentences and natural obstruction sets. Finally, we observe an MMSNP counterpart of Lemma 4.3, the fundamental 'explosion' lemma for CSPs.

The following summarizes our results regarding the shape of rewritings.

Theorem 5.1. *Let* Π *be a Boolean MDDLog program of diameter* k*. Then*

(1) *if* Π *is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth* $(1, k)$*;*
(2) *if* Π *is MDLog-rewritable, then it has an MDLog-rewriting of diameter* k*.*

Proof. We analyze the proof of Lemma 4.2 and use known results from CSP. In fact, any FO-rewritable coCSP has a UCQ-rewriting that consists of tree CQs [NT00], and thus the same holds for simple Boolean MDDLog programs. If we convert such a rewriting of Π_S into a rewriting of Π as in the proof of Lemma 4.2, we obtain a UCQ-rewriting in which each CQ has treewidth $(1, k)$. For Point 2 of Theorem 5.1, one uses the proof of Lemma 4.2 and the known fact that every MDLog-rewritable CSP has an MDLog-rewriting in which every rule body comprises at most one EDB atom, see e.g. the proof of Theorem 19 in [FV98]. □

In a sense, the concrete bound k in Points 1 and 2 of Theorem 5.1 is quite remarkable. Point 2 says, for example, that when eliminating disjunctions from a Boolean MDDLog program, it is never necessary to increase the diameter!

We now consider obstructions. An *obstruction set* $\mathcal{O}$ for a CSP template T over schema $\mathbf{S}_E$ is a set of instances over the same schema such that for any $\mathbf{S}_E$-instance I, we have $I \not\to T$ iff $O \to I$ for some $O \in \mathcal{O}$. The elements of $\mathcal{O}$ are called *obstructions*. A lot is known about CSP obstructions. For example, T is FO-rewritable if and only if it has a finite obstruction set [Ats08] if and only if it has a finite obstruction set that consists of finite trees [NT00], and T is MDLog-rewritable if and only if it has a (potentially infinite) obstruction set that consists of finite trees [FV98]. Here we consider obstruction sets for MMSNP, defined in the obvious way: an *obstruction set* $\mathcal{O}$ for an MMSNP sentence θ over schema $\mathbf{S}_E$ is a set of instances over the same schema such that for any $\mathbf{S}_E$-instance I, we have $I \not\models \theta$ iff $O \to I$ for some $O \in \mathcal{O}$. This should not be confused with colored forbidden patterns used to characterize MMSNP in [MS07]. The following characterizes FO-rewritability of MMSNP sentences in terms of obstruction sets.

Corollary 5.2. *For every MMSNP sentence* θ*, the following are equivalent:*
(1) θ *is FO-rewritable;*
(2) θ *has a finite obstruction set;*
(3) θ *has a finite set of finite obstructions of treewidth* $(1, k)$*.*

Corollary 5.2 follows from Point 1 of Theorem 5.1 and the straightfoward observations that an MMSNP sentence θ is FO-rewritable iff $\neg\theta$ is (which is equivalent to a Boolean MDDLog program) and that every finite obstruction set $\mathcal{O}$ for θ gives rise to a UCQ-rewriting $\bigvee \mathcal{O}$ of $\neg\theta$ and vice versa. We now turn to MDLog-rewritability.

Proposition 5.3. *Let* θ *be an MMSNP sentence of diameter* k*. Then* $\neg\theta$ *is MDLog-rewritable iff* θ *has a set of obstructions (equivalently: finite obstructions) that are of treewidth* $(1, k)$*.*

Proof. The "only if" direction is a consequence of Point 2 of Theorem 5.1 and the fact that, for any Boolean monadic Datalog program $\Pi \equiv \neg\theta$ of diameter k over EDB schema $\mathbf{S}_E$, a proof tree for $\texttt{goal}()$ from an $\mathbf{S}_E$-instance I and Π gives rise to a finite $\mathbf{S}_E$-instance J of treewidth $(1, k)$ with $J \to I$. The desired obstruction set for $\neg\theta$ is then the set of all these J. The "if" direction is a consequence of Theorem 5 in [BD13]. □

We remark that the results in [BD13] almost give Proposition 5.3, but do not seem to deliver any concrete bound on the parameter k of the treewidth of obstruction sets.

We close with observing an MMSNP counterpart of the 'explosion' Lemma 4.3, first giving a preliminary. Let I be an instance over some schema $\mathbf{S}_E$. A $(1, k)$*-decomposition* of I

is a pair $(V, (I_v)_{v\in V})$ where V is a set of indices and $(I_v)_{v\in V}$ is a partition of I such that for all distinct $v, v' \in V$, $|\mathsf{dom}(I_v) \cap \mathsf{dom}(I'_v)| \leq 1$ and $|\mathsf{dom}(I_v)| \leq k$. Thus, a $(1,k)$-decomposition $D = (V, (I_v)_{v\in V})$ decomposes I into parts of size at most k and with little overlap. These parts can be viewed as the facts of an instance I_D over an aggregation schema $\mathbf{S}'_E$ defined by the relations $R_{q_v(\mathbf{x})}$ where $q_v(\mathbf{x})$ is I_v viewed as a CQ, that is,

$$I_D = \{R_{q_v(x)}(\mathsf{dom}(I_v)) \mid v \in V\}$$

where we assume some fixed (but otherwise irrelevant) order on the elements of each $\mathsf{dom}(I_v)$. Now, we say that I has $(1,k)$-*decomposition girth* g if g is the supremum of the girths of I_D, for all $(1,k)$-decompositions D of I. It can be shown that I has $(1,k)$-decomposition girth ω if and only if it has treewidth $(1,k)$.

Here comes the announced MMSNP counterpart of Lemma 4.3.

Lemma 5.4. *For every instance I and $g \geq s > 0$, and every MDDLog program Π of diameter at most s, there is an instance J (over the same schema) such that $J \to I$, J has $(1,s)$-decomposition girth exceeding g, and $I \models \Pi$ iff $J \models \Pi$.*

Proof. Let Π be a Boolean MDDLog program of diameter $k \leq s$ over EDB schema $\mathbf{S}_E$. By Theorems 3.2 and 3.3, there is a k-aggregation schema $\mathbf{S}'_E$ and a set of templates S_Π over $\mathbf{S}'_E$ such that:

(1) for any $\mathbf{S}_E$-instance I with corresponding $\mathbf{S}'_E$-instance I', $I \models \Pi$ iff $I' \not\to S_\Pi$;
(2) for any $\mathbf{S}'_E$-instance I' whose girth exceeds k with corresponding $\mathbf{S}_E$-instance I, $I' \not\to S_\Pi$ iff $I \models \Pi$.

Let I and I' be an $\mathbf{S}_E$-instance and its corresponding $\mathbf{S}'_E$-instance. Furthermore, let J' be the $\mathbf{S}'_E$-instance obtained from I' by applying Lemma 4.3 with $s = \max\{|T| \mid T \in S_\Pi\}$ and g as given. Then $J' \to I'$, J' has girth exceeding g, and $J' \to S_\Pi$ iff $I' \to S_\Pi$ iff $I \not\models \Pi$. Let J be the $\mathbf{S}_E$-instance corresponding to J'. As J' has girth exceeding k, Point (2) above yields $J \models \Pi$ iff $J' \not\to S_\Pi$. In summary, we thus obtain $I \models \Pi$ iff $J \models \Pi$.

It thus remains to show that J has $(1,s)$-decomposition girth exceeding g and that $J \to I$. The former is witnessed by the $(1,k)$-decomposition $D = (V, (I_v)_{v\in V})$ of J obtained by using as V the facts of J' and as I_v the set of facts obtained from fact v during the construction of J.

As the last step, we argue that $J \to I$ follows from $J' \to I'$, and that in fact any homomorphism h from J' to I' is also a homomorphism from J to I. Thus let h be such a homomorphism. For any fact $R(a_{i_1}, \ldots, a_{i_\ell})$ in J, there is a fact $R_{q(x_1,\ldots,x_n)}(a_1, \ldots, a_m)$ in J' such that $R(x_{i_1}, \ldots, x_{i_\ell}) \in q_i(x_1, \ldots, x_n)$. We have $R_{q(x_1,\ldots,x_n)}(h(a_1), \ldots, h(a_m)) \in I'$. By definition of I', this means $R(h(a_{i_1}), \ldots, h(a_{i_\ell})) \in I$ and we are done. □

We believe that Lemma 5.4 can be useful in many contexts, saving a detour via CSPs. For example, it enables an alternative proof of Theorem 5.1. We illustrate this for Point 1. We can start with a UCQ-rewriting q of an MDDLog program Π of diameter k and show that the UCQ q_t that consists of all CQs of treewidth $(1,k)$ that are a contraction of a CQ in q must also be a rewriting of Π: take an instance I that makes Π true, use Lemma 5.4 to transform I to an I' of girth exceeding k and also exceeding the size of any CQ in q such that $I' \models \Pi$ and $I' \to I$, observe that $I' \models q$ and that a homomorphism from any CQ p in q to I' gives rise to a homomorphism from a CQ p' in q_t to I', and derive $p' \to I$ from $I' \to I$.

6. Datalog-Rewritability of Boolean MDDLog Programs and Canonical Datalog Programs

We consider rewriting Boolean MDDLog programs into Datalog programs, making two contributions. First, we show that Datalog-rewritability is decidable for programs that have equality, a condition that is defined in detail below. For programs that do not have equality, the same construction yields a procedure that is sound, but whose completeness remains an open problem. And second, we give a new and direct construction of canonical Datalog-rewritings of Boolean MDDLog programs (equivalently: the complements of MMSNP sentences), bypassing the construction of infinite templates [BD13] which involves the application of a non-trivial construction due to Cherlin, Shelah, and Shi [CSS99]. This construction is potentially useful even though it is yet unknown whether Datalog-rewritability of MDDLog programs Π is decidable (for programs that do not have equality): when Π is not rewritable, then the canonical Datalog-rewriting is the best possible approximation of Π in terms of a Datalog program (of given width and diameter).

6.1. Datalog-Rewritability of Boolean MDDLog Programs.

A CSP template T *has equality* if its EDB schema includes the distinguished binary relation $\mathtt{eq}$ and T interprets $\mathtt{eq}$ as the relation $\{(a,a) \mid a \in \mathsf{dom}(T)\}$. Thus, $\mathtt{eq}$ is an extremely natural kind of constraint: a fact $\mathtt{eq}(a,b)$ in the input instance means that a and b must be mapped to the same template element; spoken from the perspective of constraint satisfaction, they are variables that must receive the same value.

In accordance with the above, we say that an MDDLog program Π *has equality* if its EDB schema includes the distinguished binary relation $\mathtt{eq}$, Π contains the rules

$$P(x) \wedge \mathtt{eq}(x,y) \to P(y) \quad \text{and} \quad P(y) \wedge \mathtt{eq}(x,y) \to P(x)$$

for each IDB relation P, and these are the only rules that mention $\mathtt{eq}$. Thus, a fact $\mathtt{eq}(a,b)$ in the input instance says that the same IDB relations can be derived by Π for a and for b. It can be verified that when an MDDLog program that has equality is converted into a generalized CSP based on a set of templates S_Π according to Theorems 3.2 and 3.3 (using the concrete constructions in the appendix), then all templates in S_Π have equality.

We aim to show decidability of the Datalog-rewritability of MDDLog programs that have equality following the strategy that we have used for rewritability into FO and into MDLog in Section 4. We thus need a counterpart of Lemma 4.4, that is, we have to show that for all templates T that have equality, Datalog-rewritability of coCSP(T) on instances of high girth implies unrestricted Datalog-rewritability. It is here that having equality is an advantage. In particular, every input instance for coCSP(T) can be made high girth preserving (non-)homomorphisms to T by introducing additional $\mathtt{eq}$-facts. This is similar in spirit to the explosion Lemma 4.3, but the construction is much simpler than in the proof of that lemma. We next make it explicit.

Let I be an $\mathbf{S}_E$-instance and let $g \geq 0$. We use $\mathtt{pos}(I)$ to denote the set of pairs $(R(\mathbf{a}), i)$ such that $R(\mathbf{a}) \in I$ and $i \in \mathtt{pos}(R)$. In what follows, for any tuple of constants $\mathbf{a}$, we use a_i to denote its i-th component. Reserve fresh constants as follows:

- a constant b_p, for all $p = (R(\mathbf{a}), i) \in \mathtt{pos}(I)$;
- g constants $b_{p,p',1}, \ldots, b_{p,p',g}$, for all $p, p' = (R(\mathbf{a}), i), (R'(\mathbf{a}'), i') \in \mathtt{pos}(I)$ with $a_i = a'_{i'}$.

Define an instance I^g that consists of the following facts:

(1) for every $R(\mathbf{a}) \in I$ with R of arity n, the fact $R(b_{p_1}, \ldots, b_{p_n})$ where $p_i = (R(\mathbf{a}), i)$;

(2) for all distinct $p, p' = (R(\mathbf{a}), i), (R'(\mathbf{a}'), i') \in \mathtt{pos}(I)$ with $a_i = a'_{i'}$, the facts

$$\mathtt{eq}(b_p, b_{p,p',1}), \mathtt{eq}(b_{p,p',1}, b_{p,p',2}), \ldots, \mathtt{eq}(b_{p,p',g-1}, b_{p,p',g}), \mathtt{eq}(b_{p,p',g}, b_{p'}).$$

Observe that I^g has girth exceeding g. Moreover, it satisfies the following crucial property.

Lemma 6.1. *For every CSP template T over $\mathbf{S}_E$ that has equality, $I^g \to T$ iff $I \to T$.*

Proof. Let T be a template over $\mathbf{S}_E$ that has equality. We have to show that there is a homomorphism h from I to T iff there is a homomorphism h_g from I^g to T. In fact, h_g can be obtained from h by setting $h_g(b_p) = h_g(b_{p,p',j}) = h(a_i)$ when $p = (R(\mathbf{a}), i)$; conversely, h can be obtained from h_g by setting $h(a_i) = h_g(b_p)$ when $p = (R(\mathbf{a}), i)$—the latter is well-defined by construction of I^g and since $\mathtt{eq}$ is interpreted as the reflexive relation in T. □

We are now ready to establish the announced counterpart of Lemma 4.4.

Lemma 6.2. *Let S be a set of templates over schema $\mathbf{S}_E$ that have equality, and let $g \geq 0$. If coCSP(S) is DLog-rewritable on instances of girth exceeding g, then it is DLog-rewritable.*

Proof. Assume that coCSP(S) is DLog-rewritable on instances of girth exceeding g and let Γ be a concrete rewriting. We construct a Datalog program Γ' such that for any $\mathbf{S}_E$-instance I, $I \models \Gamma'$ iff $I^g \models \Gamma$. Clearly, Γ' is then a rewriting of coCSP(S) on instances of unrestricted girth.

We aim to construct Γ' such that it mimics the execution of Γ on I^g, despite being executed on I. One challenge is that the domains of I and I^g are not identical. In Γ', the IDB relations of Γ need to be adapted to reflect this change of domain, and so do the rules. Let m be the maximum arity of any relation in $\mathbf{S}_E$. Every IDB relation P of Γ gives rise to a *set* of IDB relations in Γ'. In fact, every position of P can be replaced either with

(1) ℓ positions, for some $\ell \leq m$, reflecting the case that the position is filled with a constant b_p where $p = (R(\mathbf{a}), i)$ with R ℓ-ary; or with
(2) $\ell + \ell'$ positions, for some $\ell, \ell' \leq m$, reflecting the case that the position is filled with a constant $b_{p,p',j}$ where $p = (R(\mathbf{a}), i)$ and $p' = (R'(\mathbf{a}'), i')$, with R ℓ-ary and R' ℓ'-ary.

In Case 1, the ℓ positions store the constants in $\mathbf{a}$. The symbol R and the number i from p also need to be stored, which is done as an annotation to the IDB relation. In Case 2, the first ℓ positions store the constants in $\mathbf{a}$ while the latter ℓ' positions store the constants in $\mathbf{a}'$; we additionally need to store the symbols R and R', the numbers i and i' from p and p', and the number j, which is again done by annotation of the IDB relation.

Let us make this formal. The IDB relations of Γ' take the form P^μ where P is an IDB relation of Γ and μ is a function from $\mathtt{pos}(P)$ to

$$\Omega := (\mathbf{S}_E \times \{1, \ldots, m\}) \cup (\mathbf{S}_E \times \{1, \ldots, m\} \times \mathbf{S}_E \times \{1, \ldots, m\} \times \{1, \ldots, g\})$$

such that if $\mu(\ell) = (R, i)$, then $i \in \mathtt{pos}(R)$ and if $\mu(\ell) = (R, i, R', i', j)$, then $i \in \mathtt{pos}(R)$ and $i' \in \mathtt{pos}(R')$. The arity of P^μ is $\sum_{\ell=1..\mathtt{pos}(P)} q_\ell$ where q_ℓ is the arity of R if $\mu(\ell) = (R, i)$ and q_ℓ is the sum of the arities of R and R' if $\mu(\ell) = (R, i, R', i', j)$. In the construction of Γ', we manipulate the rules of Γ to account for this change in the IDB schema. We can assume w.l.o.g. that Γ is closed under contractions of rules. Let

$$\begin{aligned} P_0(\mathbf{x}_0) \quad \leftarrow \quad & P_1(\mathbf{x}_1) \wedge \cdots \wedge P_{\ell_1}(\mathbf{x}_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & \mathtt{eq}(z_{1,1}, z_{1,2}) \wedge \cdots \wedge \mathtt{eq}(z_{\ell_3,1}, z_{\ell_3,2}) \end{aligned}$$

be a rule in Γ where $P_0, \dots, P_{\ell_1}$ are IDB and $R_1, \dots, R_{\ell_2}$ are EDB (possibly the distinguished $\mathtt{eq}$ relation), such that

($*$) every variable occurs at most once in $R_1(\mathbf{y}_1) \wedge \dots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2})$.

Note that it might be possible to write a single rule from Γ in the above form in more than one way because $\mathtt{eq}$-atoms can be placed in the second line or in the third line; we then consider all possible ways. Informally, this choice corresponds to the decision whether the $\mathtt{eq}$-atom is mapped to an $\mathtt{eq}$-fact in I^g that comes from an $\mathtt{eq}$-fact in I (Point 1 of the definition of I^g) or to a freh $\mathtt{eq}$-fact (Point 2 of the definition of I^g). Also note that rules that do not satisfy ($*$) can be ignored since they never apply in I^g.

Let $\mathbf{x}$ be the variables in the rule, and let $\delta : \mathbf{x} \to \Omega$ be such that the following conditions are satisfied:

(1) for each $R_i(\mathbf{y}_i)$ with $\mathbf{y}_i = y_1 \cdots y_k$, we have $\delta(y_j) = (R_i, j)$ for all j;
(2) for each $\mathtt{eq}(z_{i,1}, z_{i,2})$, one of the following is true for some R, i, R', i', j:
   (a) $\delta(z_{i,1}) = (R, i)$ and $\delta(z_{i,2}) = (R, i, R', i', 1)$;
   (b) $\delta(z_{i,1}) = (R, i, R', i', g)$ and $\delta(z_{i,2}) = (R', i')$;
   (c) $\delta(z_{i,1}) = (R, i, R', i', j)$ and $\delta(z_{i,2}) = (R, i, R', i', j \pm 1)$.

With each variable x in $\mathbf{x}$, we associate a tuple $\mathbf{u}_x$ of distinct variables. If $\delta(x)$ is of the form (R, i), then the length of $\mathbf{u}_x$ matches the arity of R and $\mathbf{u}_x$ is called a *variable block*. If $\delta(x)$ is of the form (R, i, R', i', j), then the length of $\mathbf{u}_x$ is the sum of the arities n and n' of R and R'; the first n variables in $\mathbf{u}_x$ are then also called a *variable block*, and so are the last n' variables. Variable blocks will either be disjoint or identical. Identities are minimized such that the following conditions are satisfied:

(I1) if x occurs in some $\mathbf{y}_i$, then $\mathbf{u}_x = \mathbf{y}_i$;
(I2) if Case 2a applies to $\mathtt{eq}(z_{i,1}, z_{i,2})$, then $\mathbf{u}_{z_{i,1}}$ is identical to the first variable block in $\mathbf{u}_{z_{i,2}}$;
(I3) if Case 2b applies to $\mathtt{eq}(z_{i,1}, z_{i,2})$, then $\mathbf{u}_{z_{i,2}}$ is identical to the second variable block in $\mathbf{u}_{z_{i,1}}$;
(I4) if Case 2c applies to $\mathtt{eq}(z_{i,1}, z_{i,2})$, then the first variable blocks of $\mathbf{u}_{z_{i,1}}$ and $\mathbf{u}_{z_{i,2}}$ are identical, and so are the second variable blocks.

Regarding (I1), note that x cannot occur in more than one $\mathbf{y}_i$ because of ($*$), thus the condition can always be satisfied 'without conflicts'. Then include in Γ' the rule

$$\begin{aligned} P_0^{\mu_0}(\mathbf{x}'_0) \quad \leftarrow \quad & P_1^{\mu_1}(\mathbf{x}'_1) \wedge \dots \wedge P_{\ell_1}^{\mu_{\ell_1}}(\mathbf{x}'_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \dots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & W \end{aligned}$$

such that

(R1) if the k-th component in $\mathbf{x}_0$ is x, then $\mu_i(k) = \delta(x)$;
(R2) $\mathbf{x}'_i$ is obtained from $\mathbf{x}_i$ by replacing each variable x with $\mathbf{u}_x$;
(R3) W contains the following atoms:
- for each variable $x \in \mathbf{x}$ with $\delta(x)$ of the form (R, i), an atom $R(\mathbf{w})$ where the i-th component of $\mathbf{w}$ is x and all other variables are distinct and fresh;
- for each variable $x \in \mathbf{x}$ with $\delta(x)$ of the form (R, i, R', i', j), atoms $R(\mathbf{w}), R'(\mathbf{w}')$ where the i-th component of $\mathbf{w}$ and the i'-th component of $\mathbf{w}'$ is x and all other variables are distinct and fresh.

As an example, consider the following rule in Γ:

$$\mathtt{goal}() \leftarrow P(x_1, x_2) \wedge R(x_1, x_2) \wedge \mathtt{eq}(x_2, x_3)$$

where R is EDB and P IDB, and let $\delta(x_1) = (R, 1)$, $\delta(x_2) = (R, 2)$, and $\delta(x_3) = (R, 2, R, 1, 1)$. Note that Case 2a applies to $\mathtt{eq}(x_2, x_3)$. We have $\mathbf{u}_{x_1} = \mathbf{u}_{x_2} = x_1x_2$ and $\mathbf{u}_{x_3} = x_1x_2u_1u_2$ and thus obtain the following rule in Γ':

$$\begin{array}{rcl} \mathtt{goal}() & \leftarrow & P^{\mu}(x_1, x_2, x_1, x_2, u_1, u_2) \wedge R(x_1, x_2) \wedge \\ & & R(x_1, w_1) \wedge R(w_2, x_2) \wedge R(w_3, x_3) \wedge R(x_3, w_4) \end{array}$$

where the last line corresponds to W above, and where $\mu(1) = (R, 1)$ and $\mu(2) = (R, 2, R, 1, 1)$.

We have have to show that $I \models \Gamma'$ iff $I^g \models \Gamma$ for any $\mathbf{S}_E$-instance I. There is a correspondence between extensions of I^g to the IDB relations in Γ and extensions of I to the IDB relations in Γ'. More precisely, a fact $P^{\mu}(\mathbf{a})$ in an extension of I represents a fact $P(\mathbf{b})$ in an extension of I^g as follows (and vice versa): for each $i \in \mathtt{pos}(P)$, let $\mathbf{a}_i$ be the subtuple of $\mathbf{a}$ that starts at position $\sum_{\ell=1..i-1} q_\ell$ and is of length q_i (where, as before, q_ℓ is the arity of R if $\mu(\ell) = (R, i)$ and q_ℓ is the sum of the arities of R and R' if $\mu(\ell) = (R, i, R', i', j)$); the i-th constant in $\mathbf{b}$ is $b_{R(\mathbf{a}_i),j}$ if $\mu(i) = (R, j)$ and $b_{R(\mathbf{c}),j,R'(\mathbf{c}'),j',\ell}$ if $\mu(i) = (R, j, R', j', \ell)$ and $\mathbf{a}_i = \mathbf{c}\mathbf{c}'$.

One essentially has to show that every application of a rule from Γ' in an extension of I can be reproduced by an application of a rule from Γ in the corresponding extension of I^g, and vice versa. We only sketch the details. First let J^g be an extension of I^g to the IDB relations in Γ and let $P(\mathbf{y}) \leftarrow q(\mathbf{x})$ be a rule in Γ applicable in J^g, and h a homomorphism from $q(\mathbf{x})$ to J^g such that $P(h(\mathbf{y})) \notin J^g$. Since Γ is closed under contractions of rules, we can assume that h is injective. Let

$$\begin{array}{rcl} q(\mathbf{x}) & = & P_1(\mathbf{x}_1) \wedge \cdots \wedge P_{\ell_1}(\mathbf{x}_{\ell_1}) \wedge \\ & & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & & \mathtt{eq}(z_{1,1}, z_{1,2}) \wedge \cdots \wedge \mathtt{eq}(z_{\ell_3,1}, z_{\ell_3,2}) \end{array}$$

such that all P_i are IDB, all R_i EDB, and an equality atom $\mathtt{eq}(x, y)$ is included in the third line if and only if at least one of $h(x)$ and $h(y)$ is not of the form b_p. Consequently, for all variables x that occur in the second line, $h(x)$ is of the form b_p. One can now verify that Condition $(*)$ is satisfied. Assume that this is not the case. The first case is that that there are distinct atoms $R_i(\mathbf{y}_i)$ and $R_j(\mathbf{y}_j)$ that share a variable x. In I^g, every constant of the form b_p occurs in exactly one fact that only contains constants of the form b_p. Thus, h must take $R_i(\mathbf{y}_i)$ and $R_j(\mathbf{y}_j)$ to the same fact in J^g. Since h is injective, $R_i(\mathbf{y}_i)$ and $R_j(\mathbf{y}_j)$ must be identical which is a contradiction. The second case is that there is an atom $R_i(\mathbf{y}_i)$ in which a variable occurs more than once. This is in contradiction to h being a homomorphism to J^g.

Now define a map $\delta : \mathbf{x} \to \Omega$ by putting $\delta(x) = p$ if $h(x) = b_p$ and $\delta(x) = (p, p', i)$ if $h(x) = b_{p,p',i}$. It can be verified that the two conditions required of δ are satisfied. We thus obtain a corresponding rule in Γ'. It can be verified that applying this rule in the extension J of I corresponding to J^g adds the fact that corresponds to $P(h(\mathbf{y}))$.

Conversely, let J be an extension of I to the IDB relations in Γ' and let

$$\begin{array}{rcl} P_0^{\mu_0}(\mathbf{x}'_0) & \leftarrow & P_1^{\mu_1}(\mathbf{x}'_1) \wedge \cdots \wedge P_{\ell_1}^{\mu_{\ell_1}}(\mathbf{x}'_{\ell_1}) \wedge \\ & & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & & W \end{array}$$

be a rule in Γ' and h a homomorphism from the rule body to J such that $P^{\mu_0}(h(\mathbf{x}'_0))$ is not in J. This rule was derived from a rule

$$\begin{aligned} P_0(\mathbf{x}_0) \quad \leftarrow \quad & P_1(\mathbf{x}_1) \wedge \cdots \wedge P_{\ell_1}(\mathbf{x}_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & \mathsf{eq}(z_{1,1}, z_{1,2}) \wedge \cdots \wedge \mathsf{eq}(z_{\ell_3,1}, z_{\ell_3,2}) \end{aligned}$$

in Γ and a map $\delta : \mathbf{x} \to \Omega$, $\mathbf{x}$ the variables in the latter rule. We define a map h' from $\mathbf{x}$ to $\mathsf{dom}(J^g)$, where J^g is the extension of I^g that corresponds to J. Let $x \in \mathbf{x}$. If $\delta(x) = (R, i)$ and $h(\mathbf{u}_x) = \mathbf{a}$, then set $h'(x) = b_{R(\mathbf{a}),i}$. If $\delta(x) = (R, i, R', i', j)$ and $h(\mathbf{u}_x) = \mathbf{a}\mathbf{a}'$, then set $h'(x) = b_{R(\mathbf{a}),i,R'(\mathbf{a}'),i',j}$. We argue that h' is a homomorphism from the body of the latter rule to J^g. There are three cases:

- Consider an atom $P_i(\mathbf{x}_i)$. Let $\mathbf{x}_i = x_1 \cdots x_n$. Then there is a corresponding atom $P_i^{\mu_i}(\mathbf{x}_i)$ in the former rule and thus $P_i^{\mu_i}(h(\mathbf{x}_i)) \in J$. For each $j \in \mathsf{pos}(P_i)$, let $\mathbf{a}_j$ be the subtuple of $h(\mathbf{x}_i)$ that starts at position $\sum_{\ell=1..j-1} q_\ell$ and is of length q_j. Define the tuple $\mathbf{b}$ by letting the j-th constant be $b_{R(\mathbf{a}_j),\ell}$ if $\mu(j) = (R, \ell)$ and $b_{R(\mathbf{c}),\ell,R'(\mathbf{c}'),\ell',k}$ if $\mu(j) = (R, \ell, R', \ell', k)$ and $\mathbf{a}_j = \mathbf{c}\mathbf{c}'$. By (R3), all constants in $\mathbf{b}$ occur in the domain of J^g. Moreover, $P_i(\mathbf{b}) \in J^g$. It thus remains to observe that $h'(\mathbf{x}_i) = \mathbf{b}$, which follows from (R1) and (R2) and the definition of h'.
- Consider an atom $R_i(\mathbf{y}_i)$. Let $\mathbf{y}_i = y_1 \cdots y_n$. Then the atom $R_i(\mathbf{y}_i)$ must also be in the former rule and thus $R_i(h(\mathbf{y}_i)) \in J$, yielding $R(b_{R_i(h(\mathbf{y}_i)),1}, \ldots, b_{R_i(h(\mathbf{y}_i)),n}) \in J^g$. By Condition 1 imposed on δ, we have $\delta(y_j) = (R_i, j)$ for each j. Moreover, by (I1) we must have $\mathbf{u}_{y_j} = \mathbf{y}_i$ for each j. Thus, the definition of h' yields $h'(\mathbf{y}_i) = b_{R_i(h(\mathbf{y}_i)),1} \cdots b_{R_i(h(\mathbf{y}_i)),n}$ and we are done.
- Consider an atom $\mathsf{eq}(z_{i,1}, z_{i,2})$. We know that one of the Cases 2a to 2d apply to $\mathsf{eq}(z_{i,1}, z_{i,2})$. We only treat the first case explicitly. Thus assume that $\delta(z_{i,1}) = (R, j)$ and $\delta(z_{i,2}) = (R, j, R', j', 1)$. By definition, $h'(z_{i,1}) = b_{R(h(\mathbf{u}_{z_{i,1}})),j}$ and $h'(z_{i,2}) = b_{R(\mathbf{c}),j,R'(\mathbf{c}'),j',1}$ where $h(\mathbf{u}_{z_{i,1}}) = \mathbf{c}\mathbf{c}'$. By (I2), $\mathbf{u}_{z_{i,1}}$ is identical to the first variable block in $\mathbf{u}_{z_{i,2}}$ and thus $h(\mathbf{u}_{z_{i,1}}) = \mathbf{c}$. By definition if I^g, J^g contains $\mathsf{eq}(b_{R(\mathbf{c}),j}, b_{R(\mathbf{c}),j,R'(\mathbf{c}'),j',1})$ and we are done.

It can now be verified that the application of the latter rule adds to J^g the fact that corresponds to $P^{\mu_0}(h(\mathbf{x}'_0))$. □

DLog-rewritability of CSPs is NP-complete [Bar16, CL17] and it was observed in [BtCLW14] that this result lifts to generalized CSPs. It thus follows from Theorems 3.2 and 3.3 and Lemma 6.2 that DLog-rewritability of Boolean MDDLog programs that have equality is decidable in 2NExpTime. It is straightforward to verify that the 2NExpTime lower bound for Datalog-rewritability of MDDLog programs from [BL16] applies also to programs that have equality.

Theorem 6.3. *For Boolean MDDLog programs that have equality, Datalog-rewritability is* 2NExpTime*-complete.*

Regarding MDDLog programs that do not have equality, the above yields a sound but possibly incomplete algorithm for deciding DLog-rewritability. Let us make this more precise. For an MDDLog program Π that does not have equality, we use $\Pi^=$ to denote the extension of Π with the fresh EDB relation eq and the above rules. If Π has equality, then $\Pi^=$ simply denotes Π.

Lemma 6.4. *For MDDLog programs Π, DLog-rewitability of $\Pi^=$ implies DLog-rewritability of Π.*

Lemma 6.4 follows from the trivial observation that any DLog-rewriting of $\Pi^=$ can be converted into a DLog-rewriting of Π by dropping all rules that use the relation eq. It is an interesting open question whether the converse of Lemma 6.4 holds. Due to Lemma 6.4, a sound but possibly incomplete algorithm for unrestricted MDDLog programs Π can thus be formulated as follows: first replace Π with $\Pi^=$ and then decide DLog-rewritability as per Theorem 6.3. We speculate that this algorithm is actually complete. In particular, for CSPs it is known that adding equality does preserve Datalog-rewritability [LZ07], and completeness of our algorithm is equivalent to an analogous result holding for MDDLog.

6.2. **Canonical Datalog-Rewritings.** For constructing actual DLog-rewritings instead of only deciding their existence, *canonical Datalog programs* play an important role. Feder and Vardi show that for every CSP template T and all $\ell, k > 0$, one can construct an (ℓ, k)-Datalog program that is canonical for T in the sense that if there is any (ℓ, k)-Datalog program which is equivalent to the complement of T, then the canonical one is [FV98]. In this section, we show that there are similarly simple canonical Datalog programs for Boolean MDDLog. Note that the existence of canonical Datalog programs for MMSNP (and thus for Boolean MDDLog) is already known from [BD13]. However, the construction given there is more general and rather complex, proceeding via an infinite template and exploiting that it is ω-categorial. This makes it hard to analyze the exact structure and size of the resulting canonical programs. Here, we define canonical Datalog programs for Boolean MDDLog programs in a more elementary way. In contrast to the previous subsection, we do not assume that equality is available.

Let Π be a Boolean MDDLog program over EDB schema $\mathbf{S}_E$ and with IDB relations from $\mathbf{S}_I$. Further let $0 \leq \ell < k$. We aim to construct a canonical (ℓ, k)-DLog program for Π. The most important properties of this program is that it is sound for Π and complete for Π on $\mathbf{S}_E$-instances of treewidth (ℓ, k). We first convert Π into a DDLog program Π' that is equivalent to Π on instances of treewidth (ℓ, k) and then construct the canonical program for Π' rather than for Π. Unlike Π, the new program Π' is not monadic. Informally, the canonical program simulates Π on $\mathbf{S}_E$-instances of treewidth (ℓ, k) proceeding in a bag-by-bag fashion. This is enabled by the additional non-monadic IDB relations introduced in Π' which represent information that needs to be passed from bag to bag. We remark that the construction of Π' is vaguely similar in spirit to the first step of converting an MDDLog program into simple form, c.f. Appendix A.1. To describe it, we need a preliminary.

With every MDDLog rule $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ where $q(\mathbf{x})$ is of treewidth (ℓ, k) and every (ℓ, k)-tree decomposition $(T, (B_v)_{v \in V})$ of $q(\mathbf{x})$, we associate a set of *rewritten rules* constructed as follows. Choose a root v_0 of the undirected tree T, thus inducing a direction. We write $v \prec v'$ if v' is a successor of v in T and use $\mathbf{x}_{v'}$ to denote $B_v \cap B_{v'}$. For all $v \in V \setminus \{v_0\}$ such that $|\mathbf{x}_v| = m$, introduce a fresh m-ary IDB relation Q_v; note that $m \leq \ell$. Now, the set of rewritten rules contains one rule for each $v \in V$. For $v \neq v_0$, the rule is

$$p_v(\mathbf{y}_v) \vee Q_v(\mathbf{x}_v) \leftarrow q(\mathbf{x})|_{B_v} \wedge \bigwedge_{v \prec v'} Q_{v'}(\mathbf{x}_{v'})$$

where $p_v(\mathbf{y}_v)$ is the sub-disjunction of $p(\mathbf{y})$ that contains all disjuncts $P(\mathbf{z})$ with $\mathbf{z} \subseteq B_v$ and $q(\mathbf{x})|_{B_v}$ is the restriction of q to the atoms that contain only variables from B_v. For v_0, we include the same rule, but use only $p_v(\mathbf{y}_v)$ as the head. The set of rewritten rules associated with $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ is obtained by taking the union of the rewritten rules associated with $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ and any $(T, (B_v)_{v \in V})$.

The DDLog program Π' is constructed from Π as follows:

(1) first extend Π with all contractions of rules in Π;
(2) then delete all rules with $q(\mathbf{x})$ not of treewidth (ℓ, k) and replace every rule $p(\mathbf{y}) \leftarrow q(\mathbf{x})$ with $q(\mathbf{x})$ of treewidth (ℓ, k) with the rewritten rules associated with it.

To clarify the relation between Π and Π', we remark that it is possible to verify the following conditions; a detailed proof is omitted since these conditions are not going to be used in what follows:

(I) Π' is sound for Π, that is, for all $\mathbf{S}_E$-instances I, $I \models \Pi'$ implies $I \models \Pi$;
(II) Π' is complete for Π on $\mathbf{S}_E$-instances of treewidth (ℓ, k), that is, for all such instances I, $I \models \Pi$ implies $I \models \Pi'$.

Note that Π' is not complete for Π on instances of unrestricted treewidth. For example, if Π consists of only a goal rule whose rule body is a $k+1$-clique (without reflexive loops), then Π' returns false on the instance that consists of the same clique.

Example 6.5. Assume that Π contains the rule

$$P_1(x) \vee P_2(z) \leftarrow R(x, y_1) \wedge S(x, y_2) \wedge R(y_1, z) \wedge R(y_2, z)$$

and consider the $(2, 3)$-tree decomposition of the rule body that consists of two nodes v, v', v' successor of v, with $B_v = \{x, y_1, y_2\}$ and $B_{v'} = \{y_1, y_2, z\}$. In Π', the rule is split into two rules

$$P_2(z) \vee Q_{v'}(y_1, y_2) \leftarrow R(y_1, z) \wedge R(y_2, z)$$
$$P_1(z) \leftarrow R(x, y_1) \wedge S(x, y_2) \wedge Q_{v'}(y_1, y_2).$$

Informally, these rules are supposed to cover homomorphisms from the body of the original rule to an $\mathbf{S}'_E$-instance of treewidth (ℓ, k) such that the variables in $B_{v'}$ are mapped to constants from some bag and variables from B_v to constants from a neighboring bag. The IDB relation $Q_{v'}$ memorizes that we have already seen part of the rule body.

Let $\mathbf{S}'_I$ denote the additional IDB relations in Π'. We now construct the canonical (ℓ, k)-DLog program Γ^c for Π. Fix constants $a_1, \ldots, a_\ell$. For $\ell' \leq \ell$, we use $\mathfrak{I}_{\ell'}$ to denote the set of all $\mathbf{S}_I \cup \mathbf{S}'_I$-instances with domain $\mathbf{a}_{\ell'} := a_1, \ldots, a_{\ell'}$. The program uses ℓ'-ary IDB relations P_M, for all $\ell' \leq \ell$ and all $M \subseteq \mathfrak{I}_{\ell'}$. It contains all rules $q(\mathbf{x}) \to P_M(\mathbf{y})$, $M \subseteq \mathfrak{I}_{\ell'}$, that satisfy the following conditions:

(1) $q(\mathbf{x})$ is over schema $\mathbf{S}_E \cup \{P_M \mid M \subseteq \mathfrak{I}_{\ell'},\ \ell' \leq \ell\}$ and contains at most k variables;
(2) for every extension J of the $\mathbf{S}_E$-instance $I_q|_{\mathbf{S}_E}$ with $\mathbf{S}_I \cup \mathbf{S}'_I$-facts such that
   (a) J satisfies all rules of Π' and does not contain $\texttt{goal}()$ and
   (b) for each $P_N(\mathbf{z}) \in q$, $N \subseteq \mathfrak{I}_{\ell''}$, there is an $L \in N$ such that $L[\mathbf{z}/\mathbf{a}_{\ell''}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{z}}$ there is an $L \in M$ such that $L[\mathbf{y}/\mathbf{a}_{\ell'}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{y}}$

where I_q is q viewed as an instance, $L[\mathbf{x}/\mathbf{a}]$ denotes the result of replacing the constants in $\mathbf{a}$ with the variables in $\mathbf{x}$ (possibly resulting in identifications), and $J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{x}}$ denotes the simultaneous restriction of J to schema $\mathbf{S}_I \cup \mathbf{S}'_I$ and constants $\mathbf{x}$.[3] We also include in Γ^c all rules of the form $P_\emptyset(\mathbf{x}) \to \texttt{goal}()$, $P_\emptyset$ of any arity from 0 to ℓ.

The intuition behind the construction of Γ^c is as follows. When starting with an input $\mathbf{S}_E$-instance I of treewidth (ℓ, k) and then chasing with Γ^c, that is, exhaustively applying these rules in an unspecified order, then the resulting instance I' represents all extensions J

[3] We could additionally demand that M is minimal so that Condition 2 is satisfied, but this is not strictly required.

of I to the relations in $\mathbf{S}_I \cup \mathbf{S}'_I$ that satisfy all rules in Π' and do not contain $\texttt{goal}()$. A fact $P_M(\mathbf{a}) \in I'$, $M \subseteq \mathfrak{I}_{\ell''}$, means that for every such J there is an $L \in M$ such that J contains the facts in $L[\mathbf{a}/\mathbf{a}_{\ell''}]$. Thus, the set M in the index of P_M should be read disjunctively. Note that $P_\emptyset(\mathbf{a}) \in I'$ then indicates that every extension of I that satisfies all rules in Π' must contain $\texttt{goal}()$. The bodies of rules in Γ^c are large enough to cover the restriction of I to the constants from any single bag. This suffices only because we have transitioned from Π to Π' before constructing Γ^c.

The following are central properties of canonical DLog programs.

Lemma 6.6.

(1) Γ^c *is sound for* Π*;*
(2) Γ^c *is complete for* Π *on instances of treewidth* (ℓ, k)*.*

Proof. For Point 1, let I be an $\mathbf{S}_E$-instance with $I \models \Gamma^c$. It suffices to show that $I \models \Pi'$. Let $I = I_1, I_2, \dots$ be the sequence of $\mathbf{S}_E \cup \mathbf{S}_I \cup \mathbf{S}'_I$-instances obtained by chasing I with Γ^c. We first note that the following can be proved by induction on i (and using the definition of Γ_c):

Claim. If $P_M(\mathbf{b}) \in I_i$, $M \subseteq \mathfrak{I}_{\ell'}$, then for every extension J of I to the relations in $\mathbf{S}_I \cup \mathbf{S}'_I$ that satisfies all rules of Π' and does not contain $\texttt{goal}()$, there is an $L \in M$ such that $L[\mathbf{b}/\mathbf{a}_{\ell'}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{b}}$.

Since $I \models \Gamma^c$, there are $i > 0$ and $\mathbf{b} \subseteq \mathsf{dom}(I)$ such that $P_\emptyset(\mathbf{b}) \in I_i$. By the claim, there is thus no extension J of I to the relations in $\mathbf{S}_I \cup \mathbf{S}'_I$ that satisfies all rules of Π and does not contain $\texttt{goal}()$. Consequently, $I \models \Pi'$.

For Point 2, assume that $I \not\models \Gamma^c$ and let $(T, (B_v)_{v \in V})$ be an (ℓ, k)-tree decomposition of I, $T = (V, E)$. Then there is an extension J of I to the IDB relations in Γ^c such that all rules in Γ^c are satisfied and J contains no atom of the form $P_\emptyset(\mathbf{b})$.

We use J to construct an extension J' of I to the relations in $\mathbf{S}_I \cup \mathbf{S}'_I$. Choose a root v_0 of T, thus inducing a direction on the undirected tree T. For all $v \in V$ and successors v' of v, choose an ordering $\mathbf{c}_{v,v'}$ of the constants in $B_v \cap B_{v'}$ and let $\ell_{v,v'}$ denote the number of these constants. Let $P_{M_1}(\mathbf{c}_{v,v'}), \dots, P_{M_r}(\mathbf{c}_{v,v'})$ be all facts of this form in J. By construction of Γ^c, there must be at least one such fact, and the fact $P_{M_1 \cap \dots \cap M_r}(\mathbf{c}_{v,v'})$ must also be in J. Thus, we can associate with v, v' a unique minimal set $M_{v,v'}$ so that $P_{M_{v,v'}}(\mathbf{c}_{v,v'}) \in J$.

The construction of J' proceeds top down over T. At all points, we maintain the invariant that

$(*)$ for all nodes $v \in V$ and successors v' of v, there is an $L \in M_{v,v'}$ such that $L[\mathbf{c}_{v,v'}/\mathbf{a}_{\ell_{v,v'}}] = J'|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{c}_{v,v'}}$.

The construction of J' starts at the root v_0 of T. There must be an extension J_{v_0} of $I|_{B_{v_0}}$ with $S_I \cup S'_I$-facts such that

(i) J_{v_0} satisfies all rules of Π and does not contain $\texttt{goal}()$
(ii) for each $P_M(\mathbf{b}) \in J|_{B_{v_0}}$, $M \subseteq \mathfrak{I}_{\ell'}$, there is an $L \in M$ such that $L[\mathbf{b}/\mathbf{a}_{\ell'}] = J_{v_0}|_{S_i \cup S'_I, \mathbf{b}}$

as, otherwise, a rule of Γ^c would create an atom of the form $P_\emptyset(\mathbf{c})$ in J. Start with putting $J' = I \cup J_{v_0}$. Note that for each successor v of v_0, $(*)$ is satisfied because of Point (ii) and since $P_{M_{v_0,v}}(a_{v_0,v}) \in J|_{B_{v_0}}$.

We proceed top-down over T. Assume that v' is a successor of v and B_v has already been treated. There must be an extension $J_{v'}$ of $I|_{B_{v'}}$ with $S_I \cup S'_I$-facts such that

(i) $J_{v'}$ satisfies all rules of Π and does not contain $\texttt{goal}()$,
(ii) $J_{v'}|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{c}_{v,v'}} = J'|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{c}_{v,v'}}$, and
(iii) for each $P_M(\mathbf{b}) \in J|_{B_{v'}}$, $M \subseteq \mathfrak{I}_{\ell'}$, there is an $L \in M$ such that $L[\mathbf{b}/\mathbf{a}_{\ell'}] = J_{v'}|_{S_i \cup S'_I, \mathbf{b}}$

as, otherwise, because of $(*)$ a rule of Γ^c would create an atom of the form $P_M(\mathbf{c}_{v,v'})$ in J with $M \subsetneq M_{v,v'}$, in contradiction to $M_{v,v'}$ being minimal with $P_{M_{v,v'}}(\mathbf{c}_{v,v'}) \in J$. Put $J' = J' \cup J_{v'}$. It can again be verified that $(*)$ is satisfied.

By construction, the instance J' does not contain $\texttt{goal}()$ and $(T, (B_v)_{v \in V})$ is also a tree decomposition of J', that is, each EDB atom *and each IDB atom* of J' falls within some bag B_v. We aim to show that J' satisfies all rules of Π, thus $I \not\models \Pi$ as required.

Let Π_0 be the result of closing Π under contractions of rules and recall that Π' is obtained from Π_0 by dropping and rewriting rules. Let ρ be a rule in Π and let h be a homomorphism from its body to J'. We have to show that one of the disjuncts in the head of ρ is satisfied under h. Π_0 contains the rule ρ_0 obtained from ρ by identifying all variables x, y such that $h(x) = h(y)$. It clearly suffices to show that one of the disjuncts in the head of ρ_0 is satisfied under h. Note that h is an injective homomorphism from the body $q(\mathbf{x})$ of ρ_0 to J' which implies that $q(\mathbf{x})$ is of treewidth (ℓ, k). Moreover, we can read off an (ℓ, k)-tree decomposition $(T', (B'_v)_{v \in V'})$ of $q(\mathbf{x})$ from h and $(T, (B_v)_{v \in V})$.

In Π', ρ_0 and $(T', (B'_v)_{v \in V'})$ are rewritten into rules $\rho_1, \dots, \rho_m$ such that no ρ_i uses a fresh IDB relation from the head of any ρ_j with $j \geq i$ (that is, an IDB relation that does not occur in Π_0, of arity at most ℓ). Let ρ_i be $q_i(\mathbf{x}_i) \to R_{i,1}(\mathbf{x}_{i,1}) \vee \dots \vee R_{i,n_i}(\mathbf{x}_{i,n_i}) \vee Q_i(\mathbf{z}_i)$ where $R_{i,1}(\mathbf{x}_{i,1}), \dots, R_{i,n_i}(\mathbf{x}_{i,n_i})$ are disjuncts that also occur in the head of ρ_0 and Q_i is a fresh IDB relation introduced by the rewriting in the case that $i < m$ and $\texttt{false}$ if $i = m$ (by which we mean: there is no $Q_i(\mathbf{z}_i)$ disjunct in the latter case). One can show by induction on i that for $1 \leq i \leq m$,

(1) $R_{j,m}(h(\mathbf{x}_{j,t})) \in J'$ for some $j \leq i$ and $t \in \{1, \dots, n_j\}$ or
(2) $Q_i(h(\mathbf{z}_i)) \in J'$.

To see this, assume that Point 1 is not satisfied for some i. Then Point 2 holds for all $j < i$. By choice of ρ_i, there is a $v \in V$ such that $h(\mathbf{x}_i) \subseteq B_v$. Thus h is a homomorphism from $q_i(\mathbf{x}_i)$ to J_v, and consequently there is a disjunct $R(\mathbf{z})$ in the head of ρ_i such that $R(h(\mathbf{z})) \in J_v \subseteq J'$. This implies that one of Points 1 or 2 is satisfied for i.

Note that Point 2 cannot hold for $i = m$ because the Q_m disjunct is not present in ρ_m. Thus there is an $i \leq m$ such that $R_{i,j}(h(\mathbf{x}_{i,j})) \in J'$ for some j. Since $R_{i,j}(\mathbf{x}_{i,j})$ occurs in the head of ρ_0, we are done. □

We are now ready to show that the canonical program is indeed canonical, as detailed by the following theorem. For two Boolean DLog programs Π_1, Π_2 over the same EDB schema $\mathbf{S}_E$, we write $\Pi_1 \subseteq \Pi_2$ if for every $\mathbf{S}_E$-instance I, $I \models \Pi_1$ implies $I \models \Pi_2$.

Theorem 6.7. *Let Π be a Boolean MDDLog program, $0 \leq \ell \leq k$, and Γ^c the canonical (ℓ, k)-DLog program for Π. Then*

(1) *$\Gamma \subseteq \Gamma^c$ for every (ℓ, k)-DLog program Γ that is sound for Π;*
(2) *Π is (ℓ, k)-DLog-rewritable iff Γ^c is a DLog-rewriting of Π.*

Proof. Let $\mathbf{S}_E$ be the EDB schema of Π.

For Point 1, let Γ be an (ℓ, k)-DLog program that is sound for Π and let I be an $\mathbf{S}_E$-instance with $I \models \Gamma$. From the proof tree for $\texttt{goal}()$ from I and Γ, we can construct an $\mathbf{S}_E$-instance J of treewidth (ℓ, k) such that $J \models \Gamma$ and $J \to I$. It suffices to show that

$J \models \Gamma_c$, which is easy: from $J \models \Gamma$, we obtain $J \models \Pi$ and Point 2 of Lemma 6.6 yields $J \models \Gamma^c$.

The "if" direction of Point 2 is trivial. For the "only if" direction, assume that Π is (ℓ, k)-DLog-rewritable and let Γ be a concrete rewriting. We have to show that Γ^c is sound and complete for Π. The former is Point 1 of Lemma 6.6. For the latter, we get $\Pi \subseteq \Gamma$ since Γ is a rewriting of Π and $\Gamma \subseteq \Gamma^c$ from Point 1, thus $\Pi \subseteq \Gamma^c$ as required. □

Note that by Point 1 of Theorem 6.7, the canonical (ℓ, k)-DLog program for an MDDLog program Π is interesting even if Π is not rewritable into an (ℓ, k)-DLog program as it is the strongest sound (ℓ, k)-DLog approximation of Π.

7. Non-Boolean MDDLog Programs

We lift the results about the complexity of rewritability, about canonical DLog programs, and about the shape of rewritings and obstructions from the case of Boolean MDDLog programs to the non-Boolean case. For all of this, a certain extension of (ℓ, k)-Datalog programs with parameters plays a central role. We thus begin by introducing these extended programs.

7.1. Deciding Rewritability.

An *(ℓ, k)-Datalog program with n parameters* is an n-ary $(\ell + n, k + n)$-Datalog program in which all IDBs have arity at least n and where in every rule, all IDB atoms agree on the variables used in the last n positions (both in rule bodies and heads and including the `goal` IDB). The last n positions of IDBs are called *parameter positions*. To visually separate the parameter positions from the non-distinguished positions, we use "|" as a delimiter to replace the usual comma, writing e.g.

$$P(x_1, x_2 \,|\, y_1, y_1, y_2) \leftarrow Q(y_1 \,|\, y_1, y_1, y_2) \wedge R(x_1, y_1, y_2, x_2)$$

where P, Q are IDB, R is EDB, and there are three parameter positions. Note that, by definition, all variable positions in `goal` atoms are parameter positions.

Example 7.1. The following is an MDLog program with one parameter that returns all constants which are on an R-cycle, R a binary EDB relation:

$$\begin{aligned} P(y \,|\, x) &\leftarrow R(y \,|\, x) \\ P(z \,|\, x) &\leftarrow P(y \,|\, x) \wedge R(z, y) \\ \texttt{goal}(x) &\leftarrow P(x \,|\, x) \end{aligned}$$

Parameters in Datalog programs play a similar role as parameters to least fixed-point operators in FO(LFP), see for example [BBV16] and references therein. The program in Example 7.1 is not definable in MDLog without parameters, which shows that adding parameters increases expressive power. Although (ℓ, k)-DLog programs with n parameters are $(\ell + n, k + n)$-DLog programs, one should think of them as a mild generalization of (ℓ, k)-programs.

A DLog program is an *ℓ-DLog program* if it is an (ℓ, k)-DLog program for some k. To lift decidability and complexity results from the Boolean to the non-Boolean case, we show that rewritability of an n-ary MDDLog program into ℓ-DLog with n parameters can be reduced to rewritability of a Boolean MDDLog program into ℓ-DLog (without parameters). We believe that Datalog with parameters is a natural rewriting target for non-Boolean MDDLog programs since, in a sense, the n parameters reflect the special role of the constants

from the input instance that are returned as an answer. Note that the case $\ell = 0$ is about UCQ-rewritability (and thus FO-rewritability) because 0-DLog programs (with and without parameters) are an alternative presentation of UCQs. The reduction proceeds in two steps, described by subsequent Lemmas 7.3 and 7.4.

Example 7.2. The following MDDLog program is rewritable into the MDLog program with parameters from Example 7.1, but not into an MDLog program without parameters:

$$\begin{aligned} P_0(x) \vee P_1(y) &\leftarrow R(x,y) \\ \texttt{goal}(x) &\leftarrow P_0(x) \\ P_1(y) &\leftarrow P_1(x) \wedge R(x,y) \\ \texttt{goal}(x) &\leftarrow P_1(x). \end{aligned}$$

The following lemma shows that, by introducing constants, we can reduce the rewritability of non-Boolean MDDLog programs into Datalog with parameters to the rewritability of Boolean MDDLog programs with constants into Datalog with constants. Note that the presence of constants in an (ℓ, k)-DLog program is not reflected in the values of ℓ and k. We will show in a second step that the rewritability of Boolean MDDLog programs with constants into Datalog with constants can be reduced to the rewritability of Boolean MDDLog programs without constants into Datalog without constants.

The *diameter* of an (ℓ, k)-DLog program with n parameters is k and the *diameter* of a DLog program with constants is defined as for DLog programs without constants, that is, only variables contribute to the diameter, but constants do not. The *rule size* of an MDDLog program is the maximum number of variable *occurrences* in a rule body.

Lemma 7.3. *Given an n-ary MDDLog program Π, one can construct Boolean MDDLog programs with constants $\Pi_1, \dots, \Pi_m$ over the same EDB schema such that for all ℓ, k,*

(1) *Π is rewritable into an (ℓ, k)-DLog program with n parameters iff each of $\Pi_1, \dots, \Pi_m$ is rewritable into an (ℓ, k)-DLog program with constants;*
(2) *$m \leq n^n$ and the size (resp. diameter, rule size) of each program Π_i is bounded by the size (resp. diameter, rule size) of Π.*

The construction takes time polynomial in the size of $|\Pi_1 \cup \dots \cup \Pi_m|$.

Proof. Let Π be an n-ary MDDLog program over EDB schema $\mathbf{S}_E$. Fix a set C of n constants. For each $\mathbf{c} \in C^n$, we construct from Π a Boolean MDDLog program $\Pi_{\mathbf{c}}$ such that for any $\ell < k$, Π is (ℓ, k)-DLog rewritable iff all programs $\Pi_{\mathbf{c}}$ are.

Let $\mathbf{c} \in C^n$. Given two n-tuples of terms (constants or variables) $\mathbf{s}$ and $\mathbf{t}$, we write $\mathbf{s} \preceq \mathbf{t}$ if $t_i = t_j$ implies $s_i = s_j$ for $1 \leq i < j \leq n$. We write $\mathbf{s} \approx \mathbf{t}$ when $\mathbf{s} \preceq \mathbf{t} \preceq \mathbf{s}$. The program $\Pi_{\mathbf{c}}$ is obtained from Π as follows:

- replace every rule $\texttt{goal}(\mathbf{x}) \leftarrow q(\mathbf{x}, \mathbf{y})$ with $\mathbf{c} \preceq \mathbf{x}$ by $\texttt{goal}() \leftarrow q(\mathbf{c}, \mathbf{y})$;
- drop every rule $\texttt{goal}(\mathbf{x}) \leftarrow q(\mathbf{x}, \mathbf{y})$ with $\mathbf{c} \not\preceq \mathbf{x}$.

Note that the non-goal rules in $\Pi_{\mathbf{c}}$ are identical to those in Π. By converting proof trees for Π into proof trees for $\Pi_{\mathbf{c}}$ and vice versa, one can show the following.

Claim. For all $\mathbf{S}_E$-instances I and $\mathbf{a} \subseteq \mathsf{dom}(I)^n$ with $\mathbf{a} \approx \mathbf{c}$, $I \models \Pi(\mathbf{a})$ iff $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$.

We show that Π is rewritable into an (ℓ, k)-DLog program with n parameters iff all of the constructed programs $\Pi_{\mathbf{c}}$ are rewritable into an (ℓ, k)-DLog program with constants.

Let Γ be an (ℓ, k)-DLog program with n parameters that is a rewriting of Π. For each $\mathbf{c} \in C^n$, let $\Gamma_{\mathbf{c}}$ be the Boolean (ℓ, k)-DLog program with constants obtained from Γ as follows:

- replace every rule $P(\mathbf{x} \,|\, \mathbf{y}) \leftarrow q(\mathbf{z} \,|\, \mathbf{y})$ with $\mathbf{c} \preceq \mathbf{y}$ (and where P might be $\mathtt{goal}$) by $P(\mathbf{x_c}) \leftarrow q(\mathbf{z_c})$, where $\mathbf{v_c}$ is the result of replacing in $\mathbf{v}$ each variable y_i with c_i;
- drop every rule $P(\mathbf{x} \,|\, \mathbf{y}) \leftarrow q(\mathbf{z} \,|\, \mathbf{y})$ with $\mathbf{c} \not\preceq \mathbf{y}$.

By translating proof trees, it can be shown that $(*)$ $I \models \Gamma(\mathbf{c})$ iff $I \models \Gamma_{\mathbf{c}}$. It is now easy to show that $\Gamma_{\mathbf{c}}$ is a rewriting of $\Pi_{\mathbf{c}}$: for every $\mathbf{S}_E$-instance I, $I \models \Pi_{\mathbf{c}}$ iff $I \models \Pi(\mathbf{c})$ (by the claim) iff $I \models \Gamma(\mathbf{c})$ (since Γ is a rewriting of Π) iff $I \models \Gamma_{\mathbf{c}}$ (by $(*)$).

Conversely, for all $\mathbf{c} \in C^n$ let $\Gamma_{\mathbf{c}}$ be a Boolean (ℓ, k)-DLog program with constants that is a rewriting of $\Pi_{\mathbf{c}}$. We construct an (ℓ, k)-DLog program with n parameters Γ as follows. For each $\mathbf{c} \in C^n$, fix a tuple $\mathbf{v}$ of fresh variables such that $\mathbf{v} \approx \mathbf{c}$. Let $\Gamma^v_{\mathbf{c}}$ be the (ℓ, k)-DLog program with n parameters obtained from $\Gamma_{\mathbf{c}}$ as follows:

(i) replace each c_i with v_i;
(ii) replace each non-goal IDB atom $P(\mathbf{x})$ with the atom $P^{\mathbf{c}}(\mathbf{x} \,|\, \mathbf{v})$ (both in rule bodies and heads), $P^{\mathbf{c}}$ a fresh IDB relation;
(iii) replace $\mathtt{goal}()$ with $\mathtt{goal}(\mathbf{v})$.

Then Γ is defined as the union of all programs $\Gamma^v_{\mathbf{c}}$. We first argue that for every $\mathbf{c} \in C^n$, $\mathbf{S}_E$-instance I, and $\mathbf{a} \subseteq \mathtt{dom}(I)^n$ with $\mathbf{a} \approx \mathbf{c}$,

(1) $I \models \Gamma_{\mathbf{c}}$ implies $I[\mathbf{a}/\mathbf{c}] \models \Gamma^v_{\mathbf{c}}(\mathbf{a})$ and
(2) $I \models \Gamma^v_{\mathbf{c}'}(\mathbf{a})$, with $\mathbf{a} \preceq \mathbf{c}'$, implies $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$.

Point 1 can be proved by showing that, from a proof tree of $\mathtt{goal}()$ from I and $\Gamma_{\mathbf{c}}$, one can construct a proof tree of $\mathtt{goal}(\mathbf{a})$ from $I[\mathbf{a}/\mathbf{c}]$ and $\Gamma^v_{\mathbf{c}}$. For Point 2, assume $I \models \Gamma^v_{\mathbf{c}'}(\mathbf{a})$ with $\mathbf{a} \preceq \mathbf{c}'$. Then $I[\mathbf{c}/\mathbf{a}] \models (\Gamma_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}']$ can again be shown by manipulating proof trees. It can be verified that, by construction, $(\Pi_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}'] \subseteq \Pi_{\mathbf{c}}$. Consequently and since $\Gamma_{\mathbf{c}'}$ is a rewriting of $\Pi_{\mathbf{c}'}$, $J \models (\Gamma_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}']$ implies $J \models \Gamma_{\mathbf{c}}$ for all J, that is, $(\Gamma_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}']$ is contained in $\Gamma_{\mathbf{c}}$ in the sense of query containment. Thus in particular $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$, as required.

It remains to show that Γ is a rewriting for Π. First assume that $I \models \Pi(\mathbf{a})$. Choose some $\mathbf{c} \in C^n$ with $\mathbf{a} \approx \mathbf{c}$. Then $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$ by the claim and thus $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$ since $\Gamma_{\mathbf{c}}$ is a rewriting of $\Pi_{\mathbf{c}}$. Point 1 above yields $I[\mathbf{c}/\mathbf{a}][\mathbf{a}/\mathbf{c}] = I \models \Gamma(\mathbf{a})$.

Now assume that $I \models \Gamma(\mathbf{a})$. Then by construction of Γ, there is a $\mathbf{c}' \in C^n$ such that $\mathbf{a} \preceq \mathbf{c}'$ and $I \models \Gamma^v_{\mathbf{c}'}(\mathbf{a})$. To see this, note in particular that the different programs $\Gamma^v_{\mathbf{c}}$ do not share any IDBs and thus do not interact in Γ. Choose a $\mathbf{c} \in C^n$ with $\mathbf{a} \approx \mathbf{c}$. From Point 2 above, we obtain $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$ which yields $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$. This implies $I \models \Pi(\mathbf{a})$ by the claim. □

We next show that constants can be eliminated from Boolean programs.

Lemma 7.4. *Given a Boolean MDDLog program Π_c with constants over EDB schema $\mathbf{S}_E$, one can construct a Boolean MDDLog program Π over an EDB schema $\mathbf{S}'_E$ such that*

(1) *Π_c is rewritable into ℓ-DLog with constants iff Π is rewritable into ℓ-DLog, for any ℓ;*
(2) *If Π_c is of size n and diameter k, then the size of Π is $2^{p(k \cdot \log n)}$; moreover, the diameter of Π is bounded by the rule size of Π_c.*

The construction takes time polynomial in the size of $|\Pi|$.

Proof. Let Π_c be a Boolean MDDLog program over EDB schema $\mathbf{S}_E$ that contains constants $c_1, \ldots, c_n$. The program Π will be over EDB schema $\mathbf{S}'_E = \mathbf{S}_E \cup \{R_1, \ldots, R_n\}$ where

$R_1, \dots, R_n$ are fresh monadic relation symbols. Π contains all rules that can be obtained from a rule ρ in Π by choosing a partial function δ that maps terms (variables or constants) in ρ to elements of $\{1, \dots, n\}$ such that $\delta(c_i) = i$ for each constant c_i and then, for each term t with $\delta(t) = i$,

(1) replacing each occurrence of t in the body of ρ with a fresh variable x and adding $R_i(x)$, and
(2) replacing each occurrence of t in the head of ρ with one of the fresh variables introduced for t in Step 1.

Additionally, Π contains the rule $\texttt{goal}() \leftarrow R_i(x), R_j(x)$, for $1 \leq i < j \leq n$.

Note that the rewriting presented above, which we call *dejoining* since it introduces different variables for each occurence of a term t in a rule body, can be applied not only to MDDLog programs, but also to MDLog programs. Before we proceed, we make a basic observation about dejoining and its connection to a certain quotient construction. Let Π be an MDDLog program or an MDLog program, with constants $c_1, \dots, c_n$, and let Π_d be the result of dejoining Π. Let I be an $\mathbf{S}'_E$-instance such that R_i, R_j are disjoint whenever $i \neq j$ and which does not contain the constants $c_1, \dots, c_n$. The *quotient* of I is the $\mathbf{S}_E$-instance I' obtained from I by replacing every $d \in \mathsf{dom}(I)$ with $R_i(d) \in I$ by the constant c_i (which also results in the identification of elements in the active domain) and removing all atoms involving one of the R_i relations. By converting proof trees of $\texttt{goal}()$ from Π into proof trees of $\texttt{goal}()$ from Π_c and vice versa, one can show the following.

Claim. $I \models \Pi$ iff $I' \models \Pi_d$.

We now show that Π_c is rewritable into ℓ-DLog iff Π is.

First let Γ_c be an ℓ-DLog rewriting of Π_c. Let Γ be obtained from Γ_c by dejoining all rules and adding the rule $\texttt{goal}() \leftarrow R_i(x), R_j(x)$ for $1 \leq i < j \leq n$. Clearly, Γ is an ℓ-DLog program. We argue that Γ is a rewriting of Π. Let I be an $\mathbf{S}'_E$-instance. W.l.o.g., we can assume that I does not contain $c_1, \dots, c_n$. If R_i, R_j are not disjoint for some $i \neq j$, then $I \models \Pi$ and $I \models \Gamma$. Otherwise, let I' be the quotient of I. We have $I \models \Pi$ iff $I' \models \Pi_c$ (by the claim) iff $I' \models \Gamma_c$ (Γ_c is rewriting of Π_c) iff $I \models \Gamma$ (again by the claim).

Let Γ be an ℓ-DLog rewriting of Π. Let Γ_c be the program constructed from Γ by removing all rules that contain atoms of the form $R_i(x)$ and $R_j(x)$ with $i \neq j$ and replacing all variables x that occur in a rule body in atoms of the form $R_i(x)$ with c_i and removing all R_i-atoms from such rules. Clearly, Γ_c is an ℓ-DLog program (with constants $c_1, \dots, c_n$). We argue that Γ_c is a rewriting of Π_c. Let I be an $\mathbf{S}_E$-instance that w.l.o.g. does not contain $c_1, \dots, c_n$ and let $I' = I \cup \{R_1(c_1), \dots, R_n(c_n)\}$. Note that I is the quotient of I'. Then $I \models \Pi_c$ iff $I' \models \Pi$ (by the claim) iff $I' \models \Gamma$ (Γ is rewriting of Π) iff $I \models \Gamma_c$ (by construction of Γ_c). □

We are now ready to lift the complexity results from Theorems 4.6 and 6.3 to the non-Boolean case, by putting them together with Lemmas 7.3 and 7.4.

Theorem 7.5. *For n-ary MDDLog programs,*

(1) *FO-rewritability (equivalently: UCQ-rewritability) is 2*NExpTime*-complete;*
(2) *rewritability into MDLog with n parameters is in 3*ExpTime *(and 2*NExpTime*-hard);*
(3) *DLog-rewritability is 2*NExpTime*-complete for programs that have equality.*

Proof. We remind that the upper bounds for rewritability of Boolean MDDLog programs stated in Theorems 4.6 and 6.3 are obtained by a (generalized) CSP and then deciding

the rewritability of (the complement of) that CSP. One can trace the blowups stated in Lemmas 7.3 and 7.4 as well as in Theorems 3.2 and 3.3 to verify that the constructed CSP does not become significantly larger in the non-Boolean case, that is, it still satisfy the bounds stated in Point 3 of Proposition 3.2. Thus, we obtain the same upper bounds as in the Boolean case. Regarding Point 1, we additionally recall that Proposition 4.1 also covers non-Boolean MDDLog programs and thus it suffices to consider UCQ-rewritability. Regarding Point 3, we note that it can be verified that the constructions in the proofs of Lemmas 7.3 and 7.4 preserve the property of having equality. □

In view of Point 2, we remark (once more) that for non-Boolean MDDLog programs Π, MDLog with parameters is in a sense a more natural target for rewriting than MDLog without parameters. The intuitive reason is that positions in the answer to Π can be thought of as constants, and constants correspond to parameters. To make this a bit more precise, consider the grounding Π' of Π obtained by replacing, in every goal rule, each variable that occurs in the head by a constant. In contrast to the standard database setup (and in contrast to the proof of Lemma 7.3), we mean here constants that are interpreted according to the standard FO semantics, that is, different constants can denote the same element of an instance. When looking for an MDLog-rewriting of Π', it is clearly very natural to admit the constants from Π' also in the rewriting. Now, one can verify that any such rewriting can be translated in a straightforward way into a rewriting of Π into MDLog with parameters, and vice versa.

We further note that MDLog with parameters enjoys similarly nice properties as standard MDLog. For example, containment is decidable. This follows from [RK13, BKR15] where generalizations of MDLog with parameters are studied, the actual parameters being represented by constants.

We also remark that Theorem 7.5 remains true when we admit constants in MDDLog programs. In fact, the proof of Lemma 7.3 goes through also when the original MDDLog program contains constants, and both the original and the newly introduced constants can then be removed by Lemma 7.4.

7.2. **Canonical Datalog-Rewritings.** We now turn our attention to canonical DLog-rewritings for non-Boolean MDDLog programs. Let Π be an n-ary MDDLog program. We associate with Π a *canonical (ℓ, k)-DLog program with n parameters*, for any $\ell < k$. The construction is a refinement of the one from the Boolean case.

We start with some preliminaries. An *n-marked instance* is an instance I endowed with n (not necessarily distinct) distinguished elements $\mathbf{c} = c_1, \ldots, c_n$. An *$(\ell, k)$-tree decomposition with n parameters* of an n-marked instance $(I, \mathbf{c})$ is an $(\ell+m, k+m)$-tree decomposition of I, m the number of distinct constants in $\mathbf{c}$, in which every bag B_v contains all constants from $\mathbf{c}$. An n-marked instance *has treewidth (ℓ, k) with n parameters* if it admits an (ℓ, k)-tree decomposition with n parameters.

We first convert Π into a DDLog program Π' that is equivalent to Π on instances of bounded treewidth. The construction is identical to the Boolean case (first variable identification, then rewriting) except that

(1) we use treewidth $(\ell + n, k + n)$ in place of treewidth (ℓ, k); consequently, the arity of the freshly introduced IDB relations may also be up to $\ell + n$;

(2) for `goal` rules, all head variables must occur in the root bag of the tree decomposition (they can then be treated in the same way as a Boolean `goal` rule despite the n-ary head relation).

It can be verified that Π' is sound for Π and that it is complete for Π on n-marked instances of treewidth (ℓ, k) with n parameters in the sense that, for all such instances $(I, \mathbf{c})$, $I \models \Pi[\mathbf{c}]$ implies $I \models \Pi'[\mathbf{c}]$. Π' is not guaranteed to be complete for answers other than $\mathbf{c}$ because of the way we treat goal rules in Point 2 above, for example when Π contains a rule of the form $\texttt{goal}(x, y) \leftarrow A(x) \wedge B(y)$.

Let $\mathbf{S}'_I$ denote the additional IDB relations in the resulting program Π'. We now construct the canonical (ℓ, k)-DLog program with n parameters Γ^c. Fix constants $a_1, \dots, a_\ell$, $b_1, \dots, b_n$ and let $\mathfrak{I}_{\ell'+n}$ denote the set of all $\mathbf{S}_I \cup \mathbf{S}_{I'}$-instances with domain $\mathbf{a}_{\ell',n} := a_1, \dots, a_{\ell'}$, $b_1, \dots, b_n$. The program uses $\ell' + n$-ary IDB relations P_M, for all $\ell' \leq \ell$ and all $M \subseteq \mathfrak{I}_{\ell',n}$. It contains all rules $q(\mathbf{x}) \to P_M(\mathbf{y} \,|\, \mathbf{x}_p)$, $M \subseteq \mathfrak{I}_{\ell',n}$, that satisfy the following conditions:

(1) $q(\mathbf{x})$ contains at most $k + n$ variables;
(2) in every extension J of the $\mathbf{S}_E$-instance $I_q|_{\mathbf{S}_E}$ with $\mathbf{S}_I \cup \mathbf{S}'_I$-facts such that
   (a) J satisfies all rules of Π' and does not contain $\texttt{goal}(\mathbf{x}_p)$ and
   (b) for each $P_N(\mathbf{z} \,|\, \mathbf{x}_p) \in q$, $N \subseteq \mathfrak{I}_{\ell'',n}$, there is an $L \in N$ such that $L[\mathbf{z}\mathbf{x}_p/\mathbf{a}_{\ell'',n}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{z}$

   there is an $L \in M$ such that $L[\mathbf{y}\mathbf{x}_p/\mathbf{a}_{\ell',n}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I}, \mathbf{y}$

We also include all rules of the form $P_\emptyset(\mathbf{y} \,|\, \mathbf{x}_p) \to \texttt{goal}(\mathbf{x}_p)$. This finishes the construction of Γ^c. It is straightforward to verify that Γ^c is sound for Π. It is complete in the same sense as Π'.

Lemma 7.6. *Γ^c is sound for Π. It is complete for Π on n-marked instances of treewidth (ℓ, k) with n parameters in the sense that for any such instance $(I, \mathbf{c})$, $I \models \Pi(\mathbf{c})$ implies $I \models \Gamma^c(\mathbf{c})$.*

The proof of Lemma 7.6 is similar to that of Lemma 6.6, details are omitted. In analogy with Theorem 6.7, we can then obtain the following result about canonical DLog programs.

Theorem 7.7. *Let Π be an n-ary MDDLog program, $0 < \ell \leq k$, and Γ^c the canonical (ℓ, k)-DLog program with n parameters associated with Π. Then*

(1) *$\Gamma \subseteq \Gamma^c$ for every (ℓ, k)-DLog program Γ that is sound for Π;*
(2) *Π is rewritable into (ℓ, k)-DLog with n parameters iff Γ^c is a rewriting of Π.*

Note that, as a consequence of Theorem 7.7, an n-ary MDDLog program Π is DLog-rewritable (in the standard sense, without parameters) iff the canonical (ℓ, k)-DLog program with n parameters is a rewriting, for some ℓ, k. In a sense, this exactly parallels the behaviour of canonical DLog programs in the Boolean case. As an important consequence, the reductions presented in Lemmas 7.3 and 7.4 show that if DLog-rewritability of Boolean programs turns out to be decidable (without assuming equality), then the same is true for DLog-rewritability of non-Boolean programs. Theorems 4.6 and 6.3

7.3. **Shape of Rewritings and Obstructions.** We now analyze the shape of rewritings of non-Boolean MDDLog programs. An *(ℓ, k)-tree decomposition with n parameters* of an n-ary CQ q is an $(\ell + n, k + n)$-tree decomposition of q in which every bag B_v contains all answer variables of q. The treewidth with n parameters of an n-ary CQ is now defined in the expected way.

Theorem 7.8. *Let Π be an n-ary MDDLog program of diameter k. Then*

(1) *if Π is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth $(1, k)$ with n parameters;*
(2) *if Π is rewritable into MDLog with n parameters, then it has an MDLog-rewriting with n parameters of diameter k.*

Note that Theorem 7.8 is immediate from Theorem 5.1 and Lemmas 7.3 and 7.4 when k denotes the rule size of Π instead of its diameter. To get the improved version, one needs to carefully trace the construction of rewritings, starting with rewritings for the CSPs ultimately constructed and then through the proofs of Lemmas 4.2, 7.4, and 7.3. In particular, the constructions in Lemmas 4.2 and 7.4 interplay in a subtle way that can be exploited to improve the bound. Details are given in Appendix C.

As in the Boolean case, rewritings are closely related to obstructions. We define obstruction sets for MMSNP formulas with free variables and summarize the results that we obtain for them. A *set of marked obstructions* $\mathcal{O}$ for an MMSNP formula θ with n free variables over schema $\mathbf{S}_E$ is a set of n-marked instances over the same schema such that for any $\mathbf{S}_E$-instance I, we have $I \not\models \theta[\mathbf{a}]$ iff for some $(O, \mathbf{c}) \in \mathcal{O}$, there is a homomorphism h from O to I with $h(\mathbf{c}) = \mathbf{a}$. We obtain the following corollary from Point 1 of Theorem 7.8 in exactly the same way in which Corollary 5.2 is obtained from Point 1 of Theorem 5.1.

Corollary 7.9. *For every MMSNP formula θ with n free variables, the following are equivalent:*

(1) *θ is FO-rewritable;*
(2) *θ has a finite marked obstruction set;*
(3) *θ has a finite set of finite marked obstructions of treewidth $(1, k)$ with n parameters.*

It is interesting to note that this result can be viewed as a generalization of the characterization of obstruction sets for CSP templates with constants in terms of 'c-acyclicity' in [AtCKT11]; our parameters correspond to constants in that paper. We now turn to MDLog-rewritability.

Proposition 7.10. *Let θ be an MMSNP formula of diameter k with n free variables. Then $\neg\theta$ is rewritable into an MDLog program with n parameters iff θ has a set of marked obstructions (equivalently: finite marked obstructions) that are of treewidth $(1, k)$ with n parameters.*

Proof. The "only if" direction is a consequence of Point 2 of Theorem 7.8 and the fact that, for any MDLog program $\Pi \equiv \neg\theta$ with n parameters of diameter k over EDB schema $\mathbf{S}_E$, a proof tree for $\mathtt{goal}(\mathbf{c})$ from an $\mathbf{S}_E$-instance I and Π gives rise to a finite n-marked $\mathbf{S}_E$-instance $(J, \mathbf{c})$ of treewidth $(1, k)$ with n parameters that satisfies $J \to I$. The "if" direction is a consequence of the fact that the canonical $(1, k)$-DLog program with parameters associated with $\neg\theta$ viewed as an MDDLog program is complete on inputs of treewidth $(1, k, n)$ with n parameters in the sense of Lemma 7.6. □

As an illustration, it might be interesting to reconsider Example 7.2. The unary MDDLog program shown there is the negation of a unary MMSNP formula that has as a set of marked obstructions the set of all R-cycles on which one element is the marked element. Each of these obstructions has treewidth $(1, 2)$ with one parameter, but not treewidth $(1, 2)$ in the strict sense.

8. Ontology-Mediated Queries

While the results on disjunctive Datalog and on MMSNP obtained in the previous sections are interesting in their own right, our premier aim is to study fundamental question of rewritability in the context of ontology-mediated queries (OMQs). Such questions have received a lot of interest in the OMQ context, see for example [BtCLW14, BHLW16, LS17] and references therein. In particular, we settle an open question from [BtCLW14] by showing that in the OMQ language $(\mathcal{ALCI}, \mathrm{CQ})$, introduced in detail below, FO-rewritability is decidable and 2NExpTime-complete. In what follows, we first introduce several prominent description logics to serve as ontology languages and, based on that, ontology-mediated queries. We then show how the results from the previous sections can be used to obtain results about ontology-mediated queries.

8.1. **Preliminaries.** In description logics, ontologies are defined by so-called TBoxes. A TBox, in turn, is a set of inclusions (that is, logical implications) between concepts (that is, logical formulas), and possibly also additional kinds of statements. Each description logic is determined by the constructors that are available to build up concepts and by the statements that are allowed in TBoxes. Here, we introduce the widely known description logics $\mathcal{ALC}$, $\mathcal{ALCI}$, and $\mathcal{SHI}$, listed in the order of increasing expressive power. We refer the reader to [BHLS17] for a more thorough introduction to DLs.

An *$\mathcal{ALCI}$-concept* is formed according to the syntax rule

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \exists r^{-}.C \mid \forall r.C \mid \forall r^{-}.C$$

where A ranges over a fixed countably infinite set of *concept names* and r over a fixed countably infinite set of *role names*. An *$\mathcal{ALC}$-concept* is an $\mathcal{ALCI}$-concept in which the constructors $\exists r^{-}.C$ and $\forall r^{-}.C$ are not used. An $\mathcal{ALC}$-TBox (resp. $\mathcal{ALCI}$-TBox) is a finite set of concept inclusions $C \sqsubseteq D$, C and D $\mathcal{ALC}$-concepts (resp. $\mathcal{ALCI}$-concepts). While $\mathcal{ALCI}$ extends $\mathcal{ALC}$ with additional concept constructors, $\mathcal{SHI}$ extends $\mathcal{ALCI}$ with additional types of TBox statements. There is thus no need to define $\mathcal{SHI}$-concepts as these are simply $\mathcal{ALCI}$-concepts. A *role* is either a role name or an expression r^{-} with r a role name. A *$\mathcal{SHI}$-TBox* is a finite set of

- *concept inclusions* $C \sqsubseteq D$, C and D $\mathcal{ALCI}$-concepts,
- *role inclusions* $r \sqsubseteq s$, r and s roles, and
- *transitivity statements* $\mathtt{trans}(r)$, r a role name.

DL semantics is given in terms of interpretations. An *interpretation* takes that form $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set called the *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* which maps each concept name A to a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name r to a binary relation $r^{\mathcal{I}} \subseteq r^{\mathcal{I}} \times r^{\mathcal{I}}$. Note that an interpretation is simply a notational variant of a relational FO-structure that interprets only unary and binary relations. The interpretation function is extended to compound concepts in the standard way, as given in Figure 2. An intepretation is a *model* of a TBox $\mathcal{T}$ if it *satisfies* all statements in $\mathcal{T}$, that is,

- $C \sqsubseteq D \in \mathcal{T}$ implies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
- $r \sqsubseteq s \in \mathcal{T}$ implies $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$;
- $\mathtt{trans}(r) \in \mathcal{T}$ implies that $r^{\mathcal{I}}$ is transitive.

For roles r, s, we write $\mathcal{T} \models r \sqsubseteq s$ if every model $\mathcal{I}$ of $\mathcal{T}$ satisfies $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$.

In description logic, data is typically stored in so-called ABoxes. For uniformity with MDDLog, we use instances instead, identifying unary relations with concept names, binary

$$
\begin{aligned}
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \exists e \in C^{\mathcal{I}} : (d,e) \in r^{\mathcal{I}}\} \\
(\exists r^{-}.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \exists e \in C^{\mathcal{I}} : (e,d) \in r^{\mathcal{I}}\} \\
(\forall r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \forall e \in \Delta^{\mathcal{I}} : (d,e) \in r^{\mathcal{I}} \Rightarrow e \in C^{\mathcal{I}}\} \\
(\forall r^{-}.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \forall e \in \Delta^{\mathcal{I}} : (e,d) \in r^{\mathcal{I}} \Rightarrow e \in C^{\mathcal{I}}\}.
\end{aligned}
$$

Figure 2: Semantics of $\mathcal{ALCI}$-concepts

relations with role names, and disallowing relations of any other arity. An interpretation $\mathcal{I}$ is a *model* of an instance I if $A(a) \in I$ implies $a \in A^{\mathcal{I}}$ and $r(a,b) \in I$ implies $(a,b) \in r^{\mathcal{I}}$. We say that an instance I is *consistent* with a TBox $\mathcal{T}$ if I and $\mathcal{T}$ have a joint model.

An *ontology-mediated query (OMQ)* over a schema $\mathbf{S}_E$ is a triple $(\mathcal{T}, \mathbf{S}_E, q)$ where $\mathcal{T}$ is a TBox formulated in a description logic and q is a query. The TBox can introduce symbols that are not in $\mathbf{S}_E$, which allows it to enrich the schema available for formulating the query q. In fact, q can use symbols from $\mathbf{S}_E$, additional symbols from $\mathcal{T}$, and also completely fresh symbols (which is useful only in very rare cases). As the TBox language, we may use any of the description logics introduced above. Since all these logics admit only unary and binary relations, we assume that these are the only allowed arities in schemas throughout Section 8. As the actual query language, we use UCQs and CQs. The OMQ languages that these choices give rise to are denoted with $(\mathcal{ALC}, \mathrm{CQ})$, $(\mathcal{SHI}, \mathrm{UCQ})$, and so on. In the actual query, we generally disallow the use of role names r such that for some role name s, $\mathtt{trans}(s) \in \mathcal{T}$ and $\mathcal{T} \models s \sqsubseteq r$. In fact, admitting such roles in the query poses serious additional complications, which are outside the scope of this paper; see e.g. [BEL$^+$10, GPT13]. To make the restriction explicit, we add a superscript $\cdot^-$ to OMQ languages when the DL used permits transitivity statements in the TBox, such as in $(\mathcal{SHI}, \mathrm{UCQ})^-$.

The semantics of an OMQ is given in terms of certain answers. Let I be an $\mathbf{S}_E$-instance and $\mathbf{a}$ a tuple of constants from I. We write $I \models Q(\mathbf{a})$ and call $\mathbf{a}$ a *certain answer to Q on I* if for all models $\mathcal{I}$ of I and $\mathcal{T}$, we have $\mathcal{I} \models q(\mathbf{a})$. The latter denotes satisfaction of $q(\mathbf{a})$ in $\mathcal{I}$ in the usual sense of first-order logic.

Example 8.1. Let $Q = (\mathcal{T}, \mathbf{S}_E, q)$ be the following OMQ, formulated in $(\mathcal{ALC}, \mathrm{CQ})$:

$$
\begin{aligned}
\mathcal{T} = \{\ & \exists \mathtt{hasAbn}.\mathtt{CTest} \sqsubseteq \mathtt{Smoker} \sqcup \exists \mathtt{hasRisk}.\mathtt{MTC}, && (1)\\
& \exists \mathtt{hasAbn}.\mathtt{CTest} \sqcap \exists \mathtt{hasRisk}.\mathtt{MEN2} \sqsubseteq \exists \mathtt{hasRisk}.\mathtt{MTC}, && (2)\\
& \mathtt{PCCPatient} \sqsubseteq \exists \mathtt{hasRisk}.\mathtt{MEN2} && (3)\\
& \exists \mathtt{hasRelative}.\exists \mathtt{hasRisk}.\mathtt{MEN2} \sqsubseteq \exists \mathtt{hasRisk}.\mathtt{MEN2}\ \} && (4)\\
\mathbf{S}_E = \{\ & \mathtt{hasAbn}, \mathtt{CTest}, \mathtt{Smoker}, \mathtt{hasRelative}, \mathtt{PCCPatient}\ \} \\
q(x) = \ & \mathtt{hasRisk}(x,y) \wedge \mathtt{MTC}(y).
\end{aligned}
$$

The TBox $\mathcal{T}$ describes the risk of somebody having medullary thyroid cancer (MTC) in the presence of an abnormal calcitonin test (CTest). While abnormal calcitonin levels are a marker for MTC, there can also be false positives, for example due to smoking (Line 1 of $\mathcal{T}$). However, in the presence of a high risk for the genetic syndrome MEN2, high calcitonin levels immediately raise an MTC suspicion (Line 2). Pheochromocitoma patients (PCCPatient)

have a high MEN2 risk (Line 3). As MEN2 is caused by a genetic mutation, the risk carries within families (Line 4). On the $\mathbf{S}_E$-instance

$$\mathtt{hasAbn}(\mathtt{john},\mathtt{t}), \mathtt{CTest}(\mathtt{t}), \mathtt{Smoker}(\mathtt{john}), \mathtt{hasRelative}(\mathtt{john},\mathtt{anna}), \mathtt{PCCPatient}(\mathtt{anna}),$$

the only certain answer to Q is $\mathtt{john}$.

An OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ is *FO-rewritable* if there is an FO query $\varphi(x)$ over schema $\mathbf{S}_E$ (and possibly involving equality), called an *FO-rewriting* of Q, such that for all $\mathbf{S}_E$-instances I and $\mathbf{a} \subseteq \mathsf{dom}(I)$, we have $I \models Q(\mathbf{a})$ iff $I \models \varphi(\mathbf{a})$. Other notions of rewritability such as UCQ-rewritability and MDLog-rewritability are defined accordingly.

Note that the TBox $\mathcal{T}$ can be inconsistent with the input instance I, that is, there could be no joint model of $\mathcal{T}$ and I. It can thus be a sensible alternative to work with *consistent FO-rewritability*, considering only $\mathbf{S}_E$-instances I that are consistent w.r.t. $\mathcal{T}$. This can then be complemented with rewritability of inconsistency for $\mathcal{T}$, that is, rewritability of the Boolean OMQ $(\mathcal{T}, \mathbf{S}_E, \exists x\, A(x))$, $A(x)$ a fresh concept name, which is true on an $\mathbf{S}_E$-instance I iff I is inconsistent with $\mathcal{T}$. It is not hard to prove, though, that consistent $\mathcal{Q}$-rewritability can be reduced to $\mathcal{Q}$-rewritability in polynomial time for all OMQ languages condidered in this paper and all $\mathcal{Q} \in \{\mathrm{FO}, \mathrm{MDLog}, \mathrm{DLog}\}$; see the corresponding proof for query containment in [BL16]. Moreover, rewritability of consistency was studied in [BtCLW14] and shown to be NExpTime-complete for all OMQ languages considered in this paper.

8.2. **Rewritability of OMQs.** We now lift the results from earlier sections to OMQs. There is a known equivalence-preserving translation from the relevant OMQ languages to MDDLog, but it involves a double exponential blowup [BtCLW14] that most likely is unavoidable.[4] We refine this translation and carefully trace the parameters in which the blowup occurs to show that, despite these blowups, the complexity of the relevant problems does not increase. The following is our main result concerning OMQs.

Theorem 8.2. *In all OMQ languages between $(\mathcal{ALC}, UCQ)$ and $(\mathcal{SHI}, UCQ)^-$, as well as between $(\mathcal{ALCI}, CQ)$ and $(\mathcal{SHI}, CQ)^-$,*

(1) *FO-rewritability (equivalently: UCQ-rewritability) is 2*NExpTime*-complete; in fact, there is an algorithm which, given an OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$, decides in time $2^{2^{p(n_q \cdot \log n_\mathcal{T})}}$ whether Q is FO-rewritable;*

(2) *MDLog-rewritability is in 3*ExpTime *(and 2*NExpTime*-hard); in fact, there is an algorithm which, given an OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$, decides in time $2^{2^{2^{p(n_q \cdot \log n_\mathcal{T})}}}$ whether Q is MDLog-rewritable*

where n_q and $n_\mathcal{T}$ are the size of q and $\mathcal{T}$ and p is a polynomial.

Note that the runtime for deciding FO-rewritability stated in Theorem 8.2 is double exponential only in the size of the actual query q (which tends to be very small) while it is only single exponential in the size of the TBox (which can become large) and similarly for MDLog-rewritability, only one exponential higher.

The lower bounds in Theorem 8.2 are from [BL16]. To prove the upper bounds, we first give a refined translation from OMQs to MDDLog. A proof is provided in Appendix D.

[4] It was shown in [BtCLW14] that a single exponential blowup is unavoidable. Whether the blowup has to be double exponential is an open problem.

Theorem 8.3. *For every OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, UCQ)^-$, one can construct an equivalent MDDLog program Π such that*

(1) *the size of Π is bounded by $2^{2^{p(n_q \cdot \log n_\mathcal{T})}}$;*
(2) *the IDB schema of Π is of size at most $2^{p(n_q \cdot \log n_\mathcal{T})}$;*
(3) *the rule size of Π is bounded by n_q*

where n_q and $n_\mathcal{T}$ are the size of q and $\mathcal{T}$ and p is a polynomial. The construction takes time polynomial in the size of Π.

Let Q be an OMQ from $(\mathcal{SHI}, \mathrm{UCQ})^-$. Instead of deciding FO- or MDLog-rewritability of Q, we can decide the same problem for the MDDLog program delivered by Theorem 8.3. The bounds stated in Theorem 8.3, Lemmas 7.3 and 7.4, and Theorems 3.2 and 3.3, though, only guarantee that we obtain a CSP template with 3-exponentially many elements, which does not yield 2NExpTime upper bounds. However, it is possible to combine the construction underlying Theorem 8.3 with those underlying Lemmas 7.3 and 7.4 and Theorem 3.2 to obtain the following.

Lemma 8.4. *Given an OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, UCQ)^-$ with $\mathcal{T}$ of size $n_\mathcal{T}$ and q of size n_q, one can construct a simple MDDLog program Π_Q over an aggregation EDB schema $\mathbf{S}'_E$ such that*

(1) *Q is $\mathcal{Q}$-rewritable iff Π_Q is $\mathcal{Q}$-rewritable for every $\mathcal{Q} \in \{FO, UCQ, MDLog\}$;*
(2) *the size of Π_Q and the cardinality of $\mathbf{S}'_E$ are bounded by $2^{2^{p(n_q \cdot \log n_\mathcal{T})}}$ and the arity of relations in $\mathbf{S}'_E$ is bounded by $\max\{n_q, 2\}$;*
(3) *the IDB schema of Π_Q is of size $2^{p(n_q \cdot \log n_\mathcal{T})}$*

where p is a polynomial. The construction takes time polynomial in the size of Π_Q.

A proof is provided in Appendix E. Constructing a CSP template from this refined simple program and applying the decision procedures for rewritability of CSP templates, we obtain the upper bounds stated in Theorem 3.3.

We remark that it is not possible to extend Theorem 8.2 to description logics with functional roles or number restrictions since, in such DLs, FO-rewritability of OMQs is undecidable [BtCLW14]. The proof can be adapted to MDLog-rewritability.

The results about the shape of rewritings stated in Theorem 7.8 (of course) also apply to the OMQ case. Note that, in Points 1 and 2 of that theorem, we can then replace k with $\max\{n_q, 2\}$. Moreover, the canonical DLog programs introduced for MDDLog in Section 7 can also be utilized for OMQs via the translation underlying the proof of Theorem 8.3.

Regarding Datalog-rewritability of OMQs, we obtain a potentially incomplete decision procedure by combining Theorem 8.3 with Lemmas 7.3 and 7.4 and the algorithm from Section 6. It is possible to define a class of OMQs $(\mathcal{T}, \mathbf{S}_E, q)$ that *have equality* and for which this procedure is complete. Roughly, $\mathbf{S}_E$ needs to contain a relation $\mathtt{eq}$ and $\mathcal{T}$ enforces that for all models $\mathcal{I}$ of $\mathcal{T}$ and all $(d, e) \in \mathtt{eq}^\mathcal{I}$, d and e satisfy exactly the same subconcepts of $\mathcal{T}$ and exactly the same tree contractions of q and then taking a subquery. We refrain from working out the details.

9. Dichotomy and Deciding PTime Query Evaluation

There was a recent breakthrough in research on CSPs, independently achieved by Bulatov and by Zhuk, who have proved the long standing Feder-Vardi conjecture thus establishing a

dichotomy between PTime and NP for the complexity of CSPs [Bul17, Zhu17]. Together with results by Chen and Larose [CL17], this also implies that it is decidable and NP-complete whether the CSP defined by a given template has PTime complexity. We observe that, together with the translations given in this paper, we obtain several interesting results on MMSNP, MDDLog, and OMQs.

In particular, we consider the (data) complexity of query evaluation, which is defined in the expected way. For example, each OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ gives rise to the following *query evaluation problem*: given an $\mathbf{S}_E$-instance I and a tuple $\mathbf{a} \subseteq \mathsf{dom}(I)$, decide whether $I \models Q(\mathbf{a})$. This problem is guaranteed to be in coNP when Q is from any of the OMQ languages studied in this paper [BtCLW14], but of course there are also OMQs Q for which it is in PTime or even simpler and from a practical perspective it is very important to understand the exact complexity of evaluating the concrete queries that are relevant for the application at hand. The definition of query evaluation is analogous for MDDLog and for MMSNP; note that MMSNP only gives rise to Boolean queries and that there is an NP upper bound for the complexity rather than a coNP one.

The question of PTime query evaluation also comes with an associated 'meta problem': given an OMQ Q (or a query from some other relevant language), decide whether Q admits PTime query evaluation. We remark that the data complexity of OMQs as well as the associated meta problem and dichotomy questions have received significant interest [BtCLW14, LW17, LS17, HLW17, LSW15, LSW13, CDL+13]. The following theorem summarizes our results regarding the complexity of query evaluation.

Theorem 9.1. *In MDDLog and all OMQ languages between* $(\mathcal{ALC}, \mathit{UCQ})$ *and* $(\mathcal{SHI}, \mathit{UCQ})^-$*, as well as between* $(\mathcal{ALCI}, \mathit{CQ})$ *and* $(\mathcal{SHI}, \mathit{CQ})^-$*,*

(1) *there is a dichotomy in the complexity of query evaluation between* PTime *and* coNP*;*
(2) *deciding* PTime*-query evaluation is 2*NExpTime*-complete.*

The same holds for MMSNP, with coNP *replaced by* NP *in Point (1).*

Proof. For MMSNP, it is well-known that there is a dichotomy for query evaluation between PTime and NP iff there is such a dichotomy for the complexity of CSPs. This gives the MMSNP version of (1). In fact, it is even known that the constructions from the proofs of Theorems 3.2 and 3.3, which transform an MMSNP sentence into a CSP, preserve complexity up to polynomial time reductions [FV98, Kun13]. We thus obtain the upper bound in the MMSNP version of (2). The lower bound is obtained from the reduction used in [BL16] to show that the Datalog-rewritability of (the complement of) MMSNP sentences is 2NExpTime-hard. The proof is by reduction of a tiling problem. Given such a tiling problem, one constructs an MMSNP sentence φ such that φ is FO-rewritable if there is a tiling and equivalent to 3-colorability (thus not Datalog-rewritable and NP-hard) otherwise. Clearly, such a reduction also yields 2NExpTime-hardness of PTime query evaluation.

For the cases of MDDLog and for OMQs, lower bounds are obtained along the same lines, that is, by observing that the lower bound constructions from [BL16] are directly applicable. For the upper bounds and the dichotomies, we first observe that the construction in the proofs of Lemma 7.4 preserves complexity up to polynomial time reductions (which is implicit in the proof) and recall that the translation in Lemma 8.4 is even equivalence-preserving. It thus remains to deal with Lemma 7.3. There, an n-ary MDDLog program Π is translated into a family of Boolean MDDLog programs (with constants) $\Pi_{\mathbf{c}}$, $\mathbf{c} \in C^n$ where C is fixed set of n constants. The claim formulated in the proof of Lemma 7.3 provides

(1) a polynomial time reduction of evaluating Π to evaluating the programs $\Pi_{\mathbf{c}}$: given an instance I and an $\mathbf{a} \subseteq \mathtt{Ind}(I)$, to decide whether $I \models \Pi(\mathbf{a})$ choose $\mathbf{c}$ with $\mathbf{a} \approx \mathbf{c}$ and check whether $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$;
(2) for each $\mathbf{c} \in C^n$, a polynomial time reduction of evaluating $\Pi_{\mathbf{c}}$ to evaluating Π: given an instance I, to decide whether $I \models \Pi_{\mathbf{c}}$ check whether $I \models \Pi(\mathbf{c})$.

It follows that Π can be evaluated in PTime iff all of the programs $\Pi_{\mathbf{c}}$ can. This is enough to transfer the upper bound for deciding PTime query evaluation. For dichotomy, we additionally need that if one of the programs $\Pi_{\mathbf{c}}$ is coNP-hard, then so is Π, which follows from (2). □

10. Discussion

We have clarified the decidability status and computational complexity of FO- and MDLog-rewritability in MMSNP, MDDLog, and various OMQ languages based on expressive description logics and conjunctive queries, and we also made several interesting observations regarding dichotomies and the decidability and complexity of PTime query evaluation. For Datalog-rewritability, we were only able to obtain partial results, namely a sound algorithm that is complete on a certain class of inputs and potentially incomplete in general. This raises several natural questions: is our algorithm actually complete in general? Does an analogue of Lemma 4.4 (that is, rewritability on instances of high girth implies rewritability) hold for Datalog as a target language? What is the complexity of deciding Datalog-rewritability in the afore-mentioned languages? From an OMQ perspective, it would also be important to work towards more practical approaches for computing (FO-, MDLog-, and DLog-) rewritings. Given the high computational complexities involved, such approaches might have to be incomplete to be practically feasible. However, the degree/nature of incompleteness should then be characterized, and we expect the results in this paper to be helpful in such an endeavour.

References

[ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *JAIR*, 36:1–69, 2009.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[AtCKT11] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, 2011.

[Ats08] Albert Atserias. On digraph coloring problems and treewidth duality. *Eur. J. Comb.*, 29(4):796–820, 2008.

[Bar16] Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016.

[BBV16] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *Proc. of LICS2016*, pages 817–826. IEEE Computer Society, 2016.

[BCF12] Manuel Bodirsky, Hubie Chen, and Tomás Feder. On the complexity of MMSNP. *SIAM J. Discrete Math.*, 26(1):404–414, 2012.

[BD13] Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013.

[BEL+10] Meghyn Bienvenu, Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Simkus. Query answering in the description logic $\mathcal{S}$. In *Proc. of DL2010*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[BHLS17] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambride University Press, 2017.

[BHLW16] Meghyn Bienvenu, Peter Hansen, Carsten Lutz, and Frank Wolter. First order-rewritability and containment of conjunctive queries in horn description logics. In *Proc. of IJCAI2016*, pages 965–971, 2016.

[BKL08] Andrei A. Bulatov, Andrei A. Krokhin, and Benoit Larose. Dualities for constraint satisfaction problems. In *Complexity of Constraints - An Overview of Current Research Themes*, volume 5250 of *LNCS*, pages 93–124. Springer, 2008.

[BKR15] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages. In *Proc. of IJCAI2015*, pages 2826–2832. AAAI Press, 2015.

[BL16] Pierre Bourhis and Carsten Lutz. Containment in monadic disjunctive Datalog, MMSNP, and expressive description logics. In *Proc. of KR2016*, pages 207–216, 2016.

[BLW13] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-order rewritability of atomic queries in Horn description logics. In *Proc. of IJCAI2013*, pages 754 – 760, 2013.

[BO15] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proc. of Reasoning Web*, volume 9203 of *LNCS*, pages 218–307. Springer, 2015.

[BtCCV15] Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *Proc. of LICS 2015*, pages 293–304. IEEE Computer Society, 2015.

[BtCLW13] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. In *Proc. of PODS2013*, pages 213–224. ACM, 2013.

[BtCLW14] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.

[Bul17] Andrei A. Bulatov. A dichotomy theorem for nonuniform csps. In *Proc. of FOCS2017*, 2017.

[CDL+09] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. Ontologies and databases: The DL-Lite approach. In *Proc. of RW2009*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.

[CDL+13] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. *Artif. Intell.*, 195:335–360, 2013.

[CGL+07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[CL17] Hubie Chen and Benoit Larose. Asking the metaquestions in constraint tractability. *ACM Trans. Comput. Theory*, 9(3):11:1–11:27, 2017.

[CSS99] Gregory L. Cherlin, Saharon Shelah, and Niandong Shi. Universal graphs with forbidden subgraphs and algebraic closure. *Advances in Applied Mathematics*, 22:454–491, 1999.

[DL08] Víctor Dalmau and Benoit Larose. Maltsev + datalog –> symmetric datalog. In *Proc. of LICS2008*, pages 297–306. IEEE Computer Society, 2008.

[EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[ELT07] László Egri, Benoit Larose, and Pascal Tesson. Symmetric datalog and constraint satisfaction problems in logspace. In *Proc. of LICS2007*, pages 193–202. IEEE Computer Society, 2007.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FKL17] Cristina Feier, Antti Kuusisto, and Carsten Lutz. Rewritability in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics (Invited Talk). In *Proc. of 20th International Conference on Database Theory, ICDT 2017*, pages 1:1–1:17, 2017.

[FV98] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[GLHS08] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Answering conjunctive queries in the $\mathcal{SHIQ}$ description logic. *Journal of Artificial Intelligence Research*, 31:150–197, 2008.

[GPT13] Georg Gottlob, Andreas Pieris, and Lidia Tendera. Querying the guarded fragment with transitivity. In *Proc. of ICALP2013*, volume 7966 of *LNCS*, pages 287–298. Springer, 2013.

[HLISW15] Peter Hansen, Carsten Lutz, İnanç Seylan, and Frank Wolter. Efficient query rewriting in the description logic el and beyond. In *Proc. of IJCAI2015*, pages 3034–3040, 2015.

[HLW17] André Hernich, Julio Lemos, and Frank Wolter. Query answering in dl-lite with datatypes: A non-uniform approach. In *Proc of AAAI*, pages 1142–1148. AAAI Press, 2017.

[KNG14] Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In *Proc. of RR*, pages 76–91, 2014.

[Kun13] Gábor Kun. Constraints, MMSNP and expander relational structures. *Combinatorica*, 33(3):335–347, 2013.

[LLT07] Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007.

[LS17] Carsten Lutz and Leif Sabellek. Ontology-mediated querying with the description logic el: Trichotomy and linear datalog rewritability. In *Proc. of IJCAI2017*, pages 1181–1187, 2017.

[LSW13] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-based data access with closed predicates is inherently intractable (sometimes). In *Proc. of IJCAI2013*, pages 1024–1030, 2013.

[LSW15] Carsten Lutz, Inanç Seylan, and Frank Wolter. Ontology-mediated queries with closed predicates. In *Proc of IJCAI2015*, pages 3120–3126. AAAI Press, 2015.

[LW17] Carsten Lutz and Frank Wolter. The data complexity of description logic ontologies. *Logical Methods in Computer Science*, 2017.

[LZ07] Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra universalis*, 56(3):439–466, 2007.

[Mad09] Florent R. Madelaine. Universal structures and the logic of forbidden patterns. *Logical Methods in Computer Science*, 5(2), 2009.

[Mad10] Florent R. Madelaine. On the containment of forbidden patterns problems. In *Proc. of CP2010*, volume 6308 of *LNCS*, pages 345–359. Springer, 2010.

[MS07] Florent R. Madelaine and Iain A. Stewart. Constraint satisfaction, logic and forbidden patterns. *SIAM J. Comput.*, 37(1):132–163, 2007.

[Nes08] Jaroslav Nesetril. Many facets of dualities. In *Proc. of Workshop on Combinatorial Optimization*, pages 285–302. Springer, 2008.

[NT00] Jaroslav Nešetřil and Claude Tardif. Duality theorems for finite structures (characterising gaps and good characterisations). *J. Comb. Theory, Ser. B*, 80(1):80–97, 2000.

[PUMH10] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *JAL*, 8(2):186–209, 2010.

[RK13] Sebastian Rudolph and Markus Krötzsch. Flag & check: Data access with monadically defined queries. In *Proc. of PODS2013*, pages 151–162. ACM, 2013.

[Ros08] Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.

[Tob01] Stephan Tobies. *Complexity results and practical algorithms for logics in knowledge representation.* PhD thesis, RWTH Aachen University, Germany, 2001.

[TSCS15] Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015.

[Zhu17] Dmitry Zhuk. The proof of csp dichotomy conjecture. In *Proc. of FOCS2017*, 2017.

Appendix A. Translating Boolean MDDLog to generalized CSP

A.1. **From MDDLog to Simple MDDLog.** Let Π be a Boolean MDDLog program over schema $\mathbf{S}_E$ and of diameter k. We first construct from Π an equivalent Boolean MDDLog program Π_B such that the following conditions are satisfied:

(i) all rule bodies are biconnected, that is, when any single variable is removed from the body (by deleting all atoms that contain it), then the resulting rule body is still connected;

(ii) if $R(x,\dots,x)$ occurs in a rule body with R EDB, then the body contains no other EDB atoms.

A good way to think about what is achieved in this first step is that, when the resulting program is evaluated on an instance of bounded treewidth, then it suffices to map the rule bodies to individual bags while it is never necessary to cross 'bag boundaries'.

To construct Π_B, we first extend Π with all contractions of rules in Π; we will refer to this step as the *collapsing step*. We then split up rules that are not biconnected into multiple rules by exhaustively executing the following rewriting steps:

- replace every rule $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1) \wedge q_2(\mathbf{x}_2)$ where $\mathbf{x}_1$ and $\mathbf{x}_2$ share exactly one variable x but both contain also other variables with the rules $p_1(\mathbf{y}_1) \vee Q(x) \leftarrow q_1(\mathbf{x}_1)$ and $p_2(\mathbf{y}_2) \leftarrow Q(x) \wedge q_2(\mathbf{x}_2)$, where Q is a fresh monadic IDB relation and $p_i(\mathbf{y}_i)$ is the restriction of $p(\mathbf{y})$ to atoms that are nullary or contain a variable from $\mathbf{x}_i$, $i \in \{1,2\}$;
- replace every rule $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1) \wedge q_2(\mathbf{x}_2)$ where $\mathbf{x}_1$ and $\mathbf{x}_2$ share no variables and are both non-empty with the rules $p_1(\mathbf{y}_1) \vee Q() \leftarrow q_1(\mathbf{x}_1)$ and $p_2(\mathbf{y}_2) \leftarrow Q() \wedge q_2(\mathbf{x}_2)$, where $Q()$ is a fresh nullary IDB relation and the $p_i(\mathbf{y}_i)$ are as above;
- replace every rule $p(\mathbf{y}) \leftarrow R(x,\dots,x) \wedge q(\mathbf{x})$ where R is an EDB relation and q contains at least one EDB atom and the variable x, with the rules $Q(x) \leftarrow R(x,\dots,x)$ and $p(\mathbf{y}) \leftarrow Q(x) \wedge q(\mathbf{x})$, where Q is a fresh monadic IDB relation.

It is easy to see that the resulting program Π_B is equivalent to the original program Π and that all Π_B satisfies Conditions (i) and (ii) above.

We next construct from Π_B the desired simple program Π_S by replacing, in every rule, the EDB atoms in the rule body with a single EDB atom that represents the conjunction of all atoms replaced. We thus introduce fresh EDB relations that represent conjunctions of old EDB relations. Note that there can be implications between the new EDB relations that we will have to take care of in the construction of Π_B.

Let Q_Π denote the set of CQs that can be obtained from a rule body in Π_B by consistently renaming variables, using only variables that occur in Π_B. Let $\mathbf{S}_I$ be the IDB schema of Π_B. For every $q(\mathbf{x}) \in \mathcal{Q}_\Pi$, we write $q(\mathbf{x})|_{\mathbf{S}_E}$ to denote the restriction of $q(\mathbf{x})$ to $\mathbf{S}_E$-atoms, and likewise for $q(\mathbf{x})|_{\mathbf{S}_I}$ and IDB atoms. The EDB schema $\mathbf{S}'_E$ of Π_S consists of the relations $R_{q(\mathbf{x})|_{\mathbf{S}_E}}$, $q(\mathbf{x}) \in \mathcal{Q}_\Pi$, whose arity is the number of variables in $q(\mathbf{x})$ (which, by construction of Π_B, is identical to the number of variables in $q(\mathbf{x})|_{\mathbf{S}_E}$). The program Π_S consists of the following rules:

> whenever $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1)$ is a rule in Π_B, $q_2(\mathbf{x}_2) \in \mathcal{Q}_\Pi$, and $q_1(\mathbf{x}_1) \subseteq q_2(\mathbf{x}_2)$,
> then Π_S contains the rule $p(\mathbf{y}) \leftarrow R_{q_2(\mathbf{x}_2)}(\mathbf{x}_2) \wedge q_1(\mathbf{x}_1)|_{\mathbf{S}_I}$

The case where $q_1(\mathbf{x}_1)$ is identical to $q_2(\mathbf{x}_2)$ corresponds to adapting rules in Π_B to the new EDB signature and the other cases take care of implications between EDB relations.

Example A.1. Assume that Π_B contains the following rules, where A and r are EDB relations:

$$\begin{aligned} P(x_3) &\leftarrow A(x_1) \wedge r(x_1,x_2) \wedge r(x_2,x_3) \wedge r(x_3,x_1) \\ \texttt{goal}() &\leftarrow r(x_1,x_2) \wedge r(x_2,x_3) \wedge r(x_3,x_1) \wedge \\ &\qquad P(x_1) \wedge P(x_2) \wedge P(x_3) \end{aligned}$$

A new ternary EDB relation R_{q_2} is introduced for the EDB body atoms of the lower rule, where $q_2 = r(x_1,x_2) \wedge r(x_2,x_3) \wedge r(x_3,x_1)$, and a new ternary EDB relation R_{q_1} is introduced for the upper rule, $q_1 = A(x_1) \wedge q_2$. In Π_S, the rules are replaced with

$$\begin{aligned} P(x_3) &\leftarrow R_{q_1}(x_1,x_2,x_3) \\ \texttt{goal}() &\leftarrow R_{q_2}(x_1,x_2,x_3) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3) \\ \texttt{goal}() &\leftarrow R_{q_1}(x_1,x_2,x_3) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3) \end{aligned}$$

Note that $q_2 \subseteq q_1$ and thus q_1 logically implies q_2, which results in two copies of the goal rule to be generated.

Proof details for the following lemma can be found in [BL16]. Recall that k is the diameter of the original MDDLog program Π.

Lemma A.2.

(1) *If I is an $\mathbf{S}_E$-instance and I' the corresponding $\mathbf{S}'_E$-instance, then $I \models \Pi$ iff $I' \models \Pi_S$;*
(2) *If I' is an $\mathbf{S}'_E$-instance and I the corresponding $\mathbf{S}_E$-instance, then*
   (a) *$I' \models \Pi_S$ implies $I \models \Pi$;*
   (b) *$I \models \Pi$ implies $I' \models \Pi_S$ if the girth of I' exceeds k.*

The following example demonstrates why the restriction to high girth instances in Point 2b of Lemma A.2 is necessary, see also Example 3.1.

Example A.3. Consider the programs Π_B and Π_S from Example A.1. Take the $\mathbf{S}'_E$-instance I' defined by

$$R_{q_1}(a,a',c'), R_{q_1}(b,b',a'), R_{q_1}(c,c',b').$$

It can be verified that $I' \not\models \Pi_S$. But the corresponding $\mathbf{S}_E$-instance I is such that Π_B derives the IDB relation P at a', b', and c', and additionally I contains the facts

$$r(c',b'), r(b',a'), r(a',c')$$

which are not covered by any fact in I'. Thus clearly $I \models \Pi_B$.

A.2. **From Simple MDDLog to Generalized CSP.** Let Π be a simple MDDLog program over EDB schema $\mathbf{S}_E$ and with IDB schema $\mathbf{S}_I$. For $i \in \{0,1\}$, an *i-type* is a set t of relation symbols from $\mathbf{S}_I$ of arity at most i that does not contain $\texttt{goal}()$ and that satisfies all rules in Π which use only IDB relations of arity at most i and do not involve any EDB relations.

We build a template T_θ for each 0-type θ. The elements of T_θ are exactly the 1-types that agree with θ on nullary IDB relations. T_θ consists of the following facts:

(1) $P()$ for each nullary $P \in \theta$.
(2) $P(t)$ for each 1-type t and each monadic $P \in t$;
(3) $R(t_1,\dots,t_n)$ for each relation $R \in \mathbf{S}_E$ and all 1-types $t_1,\dots,t_n$ such that Π does not contain a rule

$$P(x_i) \leftarrow R(x_1,\dots,x_n) \wedge P_1(x_{i_1}) \wedge \cdots \wedge P_n(x_{i_n})$$

such that $P_j \in t_{i_j}$ for $1 \le j \le n$, and $P \notin t_i$.

The following was observed in [FV98].

Lemma A.4. *For any $\mathbf{S}_E$-instance I, we have $I \models \Pi$ iff $I \not\to T_\theta$ for all 0-types θ.*

Appendix B. MDLog-Rewritability of Generalized CSP

In the proof of the subsequent theorem, we use obstructions of CSPs, which are defined in Section 5.

Theorem B.1. *Given a finite set of templates S, it can be decided in* ExpTime *whether coCSP(S) is MDLog-rewritable.*

Proof. Consider coCSP(S) over schema $\mathbf{S}_E$ with $S := \{T_1, \dots, T_n\}$. We start with observing that we can assume that the templates in S are mutually homomorphically incomparable: if this is not the case, we remove templates that are not homomorphically minimal and further remove templates so that none of the remaining templates are homomorphically equivalent. Clearly, this is equivalence preserving and can be done in ExpTime.

We aim to show that coCSP(S) is MDLog-rewritable if and only if coCSP(T_i) is for all $i \in \{1, \dots, n\}$, which gives the desired ExpTime upper bound. The "if" direction is immediate since the union of MDLog programs is expressible as an MDLog program. For the "only if" direction, assume that coCSP(S) is MDLog-rewritable, and let Γ denote a concrete rewriting. Consider a template T_j and let $\mathcal{O}(T_j)$ denote the set of all finite $\mathbf{S}_E$-instances of treewidth $(1, k)$ that do not homomorphically map to T_j where k is the maximum number of variables that occur in a single rule of Γ. We will show that $\mathcal{O}(T_j)$ is an obstruction set for T_j. It then follows from Theorem 23 of [FV98], which says that the existence of an obstruction set of treewidth $(1, k)$ for some fixed k implies MDLog-rewritability, that coCSP(T_j) is MDLog-rewritable.

By definition of $\mathcal{O}(T_j)$, it is immediate that if $O \to I$ for some $O \in \mathcal{O}(T_j)$ and $\mathbf{S}_E$-instance I, then $I \not\to T_j$. We now establish the converse. Assume that $I \not\to T_j$. Consider the disjoint union U of I and T_j. Since the templates in S are homomorphically incomparable, $U \not\to T_i$ for all $i \in \{1, \dots, n\}$. Thus $U \models \Gamma$ and there is a proof tree for $\texttt{goal}()$ from U and Γ. From that tree, we can read off an $\mathbf{S}_E$-instance J such that $J \to U$, J has treewidth $(1, k)$, and $J \models \Gamma$. From the latter, we get $J \not\to T_j$. There must thus also be a connected component O of J with $O \not\to T_j$. We clearly have $O \in \mathcal{O}(T_j)$. Since $O \to U$, $O \not\to T_j$, and O is connected, we moreover get $O \to I$ which finishes the proof. □

Appendix C. Proof of Theorem 7.8

Theorem 7.8. *Let Π be an n-ary MDDLog program of diameter k. Then*

1. *if Π is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth $(1, k)$ with n parameters;*
2. *if Π is rewritable into MDLog with n parameters, then it has an MDLog-rewriting with n parameters of diameter k.*

Proof. We treat the two cases, FO-rewritability and MDLog-rewritability with parameters, in parallel in a uniform way. To achieve uniformity, recall that FO-rewritability coincides with UCQ-rewritability by Proposition 4.1 and observe that a UCQ-rewriting of treewidth

$(1, k)$ with n parameters can be converted into a non-recursive MDLog-rewriting with n parameters of diameter k and vice versa. We work with the latter.

Assume that an n-ary MDDLog program Π over EDB schema $\mathbf{S}_E$ is rewritable into (non-recursive) MDLog with n parameters. We can convert

(1) Π into Boolean MDDLog programs $\Pi_1, \dots, \Pi_k$ with constants (Lemma 7.3),
(2) $\Pi_1, \dots, \Pi_k$ into Boolean MDDLog programs $\Pi'_1, \dots, \Pi'_k$ without constants (Lemma 7.4),
(3) $\Pi'_1, \dots, \Pi'_k$ into simple Boolean MDDLog programs $\Pi''_1, \dots, \Pi''_k$ (Theorem 3.2), and
(4) $\Pi''_1, \dots, \Pi''_k$ into CSP templates $T_1, \dots, T_k$ (Theorem 3.3)

such that all these programs and (complements of) templates are rewritable into (non-recursive) MDLog. Moreover, in the proofs of the mentioned lemmas and theorems, it is shown how to construct (non-recursive) MDLog-rewritings of $\Pi''_1, \dots, \Pi''_k$ from given ones of $T_1, \dots, T_k$, for $\Pi'_1, \dots, \Pi'_k$ from given ones of $\Pi''_1, \dots, \Pi''_k$, and so on. We are going to analyze these constructions in more detail.

We first note that for any (non-recursive) MDLog-rewritable CSP, there is a (non-recursive) MDLog-rewriting where every rule body has at most one EDB atom that contains all variables which occur in the rule body. Since each program Π''_i is actually *equivalent* to the complement of the CSP template T_i in Step 4, the same is true for the programs Π''_i. Thus, there is a (non-recursive) MDLog-rewriting Γ''_i of Π''_i in which

(†) each rule body has at most one EDB atom that contains all variables.

The translation of Π'_i into Π''_i in Step 3 involves replacing the EDB schema $\mathbf{S}_E$ with an aggregation schema $\mathbf{S}'_E$. More precisely, $\mathbf{S}'_E$ consists of relations $R_{q(x)}$ where $q(x)$ is obtained from a rule body in Π'_i by first contracting, then splitting up the body into biconnected components, and finally dropping all IDB relations. When translating the rewriting Γ''_i of Π''_i into a rewriting Γ'_i of Π'_i, this change in schema is reverted. By (†), the diameter of Γ'_i is thus bounded by the arity of relations in Γ''_i and that arity, in turn, is bounded by the diameter of Π'_i. What's more important, though, is that we actually know what the rule bodies in Γ'_i look like:

(‡) every rule body in Γ'_i is obtained from a rule body in Π'_i by first contracting, then splitting up the body into biconnected components, then dropping all IDB relations, and finally decorating with some fresh IDB relations without introducing fresh variables.

Now consider the translation of Π_i into Π'_i in Step 2 and the corresponding translation of Γ'_i into a rewriting Γ_i of Π_i. In the former, we dejoin rule bodies by (sometimes) replacing different occurrences of the same variable x with different variables x_1, x_2 and adding the atoms $R_j(x_1)$ and $R_j(x_2)$ for some j, thus increasing the diameter. In the latter, we rejoin the dejoined rules in Γ'_i in the sense that we replace variables x, y with the same constant c_j whenever the rule body contains the (EDB) atoms $R_j(x)$ and $R_j(y)$. It can be verified that rejoining any rule body of the form (‡) results in a rule body whose diameter is bounded by the diameter of Π'_i. This gives the desired result since Step 1 preserves diameter. □

Appendix D. Proof of Theorem 8.3

Theorem 8.3. *For every OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, UCQ)^-$, one can construct an equivalent MDDLog program Π such that*

(1) *the size of Π is bounded by $2^{2^{p(n_q \cdot \log n_\mathcal{T})}}$;*
(2) *the IDB schema of Π is of size at most $2^{p(n_q \cdot \log n_\mathcal{T})}$;*

(3) *the rule size of* Π *is bounded by* n_q

where n_q *and* $n_\mathcal{T}$ *are the size of* q *and* $\mathcal{T}$ *and* p *is a polynomial. The construction takes time polynomial in the size of* Π.

Proof. Let $Q = (\mathcal{T}, \mathbf{S}_E, q_0)$ be an OMQ from $(\mathcal{SHI}, \text{UCQ})$ and let n_{q_0} and $n_\mathcal{T}$ be the size of q_0 and $\mathcal{T}$, respectively. We use $\mathtt{sub}(\mathcal{T})$ to denote the set of subconcepts of (concepts occurring in) $\mathcal{T}$. Moreover, let Γ be the set of all tree CQs that can be obtained from a CQ in q_0 by first existentially quantifying all answer variables, then contracting, and then taking a subquery. Every $q \in \Gamma$ can be viewed as a $\mathcal{ALCI}$-concept provided that we additionally choose a root x of the tree. We denote this concept with $C_{q,x}$. For example, the tree CQ $q = \exists x \exists y \exists z\, r(x,y) \wedge A(y) \wedge s(x,z)$ and choice of x as the root yields the $\mathcal{ALCI}$-concept $C_{q,x} = \exists r.A \sqcap \exists s.\top$. Let $\mathtt{con}(q_0)$ be the set of all these concepts $C_{q,x}$ and let $\mathbf{S}_I$ be the schema that consists of monadic relation symbols P_C and $\overline{P}_C$ for each $C \in \mathtt{sub}(\mathcal{T}) \cup \mathtt{con}(q_0)$ and nullary relation symbols P_q and $\overline{P}_q$ for each $q \in \Gamma$. We are going to construct an MDDLog program Π over EDB schema $\mathbf{S}_E$ and IDB schema $\mathbf{S}_I$ that is equivalent to Q.

By a *diagram*, we mean a conjunction $\delta(\mathbf{x})$ of atoms over the schema $\mathbf{S}_E \cup \mathbf{S}_I$. For an interpretation $\mathcal{I}$, we write $\mathcal{I} \models \delta(\mathbf{x})$ if there is a homomorphism from $\delta(\mathbf{x})$ to $\mathcal{I}$, that is, a map $h : \mathbf{x} \to \Delta^\mathcal{I}$ such that:

(1) $A(x) \in \delta$ with $A \in \mathbf{S}_E$ implies $h(x) \in A^\mathcal{I}$;
(2) $r(x,y) \in \delta$ with $r \in \mathbf{S}_E$ implies $(h(x), h(y)) \in A^\mathcal{I}$;
(3) $P_q() \in \delta$ implies $\mathcal{I} \models q$ and $\overline{P}_q() \in \delta$ implies $\mathcal{I} \not\models q$;
(4) $P_C(x) \in \delta$ implies $h(x) \in C^\mathcal{I}$ and $\overline{P}_C() \in \delta$ implies $h(x) \notin C^\mathcal{I}$.

We say that $\delta(\mathbf{x})$ is *realizable* if there is an model $\mathcal{I}$ of $\mathcal{T}$ with $\mathcal{I} \models \delta(\mathbf{x})$. A diagram $\delta(\mathbf{x})$ *implies* a CQ $q(\mathbf{x}')$, with $\mathbf{x}'$ a tuple of variables from $\mathbf{x}$, if every homomorphism from $\delta(\mathbf{x})$ to some model $\mathcal{I}$ of $\mathcal{T}$ is also a homomorphism from $q(\mathbf{x}')$ to $\mathcal{I}$. The MDDLog program Π consists of the following rules:

(1) the rule $P_q() \vee \overline{P}_q() \leftarrow \mathtt{true}(x)$ for each $q \in \Gamma$;
(2) the rule $P_C(x) \vee \overline{P}_C(x) \leftarrow \mathtt{true}(x)$ for each $C \in \mathtt{sub}(\mathcal{T}) \cup \mathtt{con}(q_0)$;
(3) the rule $\bot \leftarrow \delta(x)$ for each non-realizable diagram $\delta(x)$ that contains a single variable x and only atoms of the form $P_C(x)$, $C \in \mathtt{sub}(\mathcal{T}) \cup \mathtt{con}(q_0)$;
(4) the rule $\bot \leftarrow \delta(\mathbf{x})$ for each non-realizable connected diagram $\delta(\mathbf{x})$ that contains at most two variables and at most three atoms;
(5) the rule $\mathtt{goal}(\mathbf{x}') \leftarrow \delta(\mathbf{x})$ for each diagram $\delta(\mathbf{x})$ that implies $q_0(\mathbf{x})$, has at most n_{q_0} variable occurrences, and uses only relations of the following form: P_q, P_C with C a concept name that occurs in q_0, and role names from $\mathbf{S}_E$ that occur in q_0.

To understand Π, a good first intuition is that rules of type 1 and 2 guess an interpretation $\mathcal{I}$, rules of type 3 and 4 take care that the independent guesses are consistent with each other, with the facts in I and with the inclusions in the TBox $\mathcal{T}$, and rules of type 5 ensure that Π returns the answers to q_0 in $\mathcal{I}$.

However, this description is an oversimplification. Guessing $\mathcal{I}$ is not really possible since $\mathcal{I}$ might have to contain additional domain elements to satisfy existential quantifiers in $\mathcal{T}$ which may be involved in homomorphisms from (a CQ in) q_0 to $\mathcal{I}$, but new elements cannot be introduced by MDDLog rules. Instead of introducing new elements, rules of type 1 and 2 thus only guess the tree CQs that are satisfied by those elements. Tree CQs suffice because $\mathcal{SHI}$ has a tree-like model property and since we have disallowed the use of roles in the query that have a transitive subrole. The notion of 'diagram implies query' used

in the rules of type 5 takes care that the guessed tree CQs are taken into account when looking for homomorphisms from q_0 to the guessed model. This construction is identical to the one used in the proof of Theorem 1 of [BtCLW13], with two exceptions. First, we use predicates P_C and $\overline{P}_C$ for every concept $C \in \mathtt{sub}(\mathcal{T}) \cup \mathtt{con}(q_0)$ while the mentioned proof uses a predicate P_t for every subset $t \subseteq \mathtt{sub}(\mathcal{T}) \cup \mathtt{con}(q_0)$. And second, our versions of Rules 3-5 are formulated more carefully. It can be verified that the correctness proof given in [BtCLW13] is not affected by these modifications. The modifications do make a difference regarding the size of Π, though, which we analyse next.

It is not hard to see that, for some polynomial p, the number of rules of type 1 is bounded by $2^{p(n_{q_0})}$, the number of rules of type 2 and of type 4 is bounded by $2^{p(n_{q_0} \cdot \log n_{\mathcal{T}})}$, the number of rules of type 3 is bounded by $2^{2^{p(n_{q_0} \cdot \log n_{\mathcal{T}})}}$, and the number of rules of type 5 is bounded by $2^{2^{p(n_{q_0})}}$. Consequently, the overall number of rules is bounded by $2^{2^{p(n_{q_0} \cdot \log n_{\mathcal{T}})}}$ and so is the size of Π. The bounds on the size of the IDB schema and number of rules in Π stated in Theorem 8.3 are easily verified. It remains to argue that the construction can be carried out in double exponential time. It suffices to observe two facts. First, consistency of a given diagram $\delta(\mathbf{x})$ can be decided in ExpTime since the satisfiability of $\mathcal{SHI}$ concepts w.r.t. TBoxes is in ExpTime [Tob01]. And second, for a given diagram $\delta(\mathbf{x})$ and CQ $q(\mathbf{x}')$ with $\mathbf{x}'$ a tuple of variables from $\mathbf{x}$, it can be decided in time single exponential in the size of $\delta(\mathbf{x})$ and of $\mathcal{T}$ and double exponential in the size of $q(\mathbf{x}')$ whether $\delta(\mathbf{x})$ implies $q(\mathbf{x}')$. This is a consequence of the fact that, in $\mathcal{SHI}$, given an ABox $\mathcal{A}$ that may contain compound concepts (in place of concept names), a TBox $\mathcal{T}$, a CQ $q(\mathbf{x})$ and a candidate answer $\mathbf{a}$, it can be decided in time single exponential in the size of $\mathcal{A}$ and $\mathcal{T}$ and double exponential in the size of q whether a is a certain answer to q on $\mathcal{A}$ w.r.t. $\mathcal{T}$ [GLHS08]. □

Appendix E. Proof of Lemma 8.4

Lemma E.1. *Given an OMQ $Q = (\mathcal{T}, \mathbf{S}_E, q)$ from $(\mathcal{SHI}, UCQ)^-$ with $\mathcal{T}$ of size $n_{\mathcal{T}}$ and q of size n_q, one can construct a simple MDDLog program Π_Q over an aggregation EDB schema $\mathbf{S}'_E$ such that*

(1) *Q is $\mathcal{Q}$-rewritable iff Π_Q is $\mathcal{Q}$-rewritable for every $\mathcal{Q} \in \{FO, UCQ, MDLog\}$;*
(2) *the size of Π_Q and the cardinality of $\mathbf{S}'_E$ are bounded by $2^{2^{p(n_q \cdot \log n_{\mathcal{T}})}}$ and the arity of relations in $\mathbf{S}'_E$ is bounded by $\max\{n_q, 2\}$;*
(3) *the IDB schema of Π_Q is of size $2^{p(n_q \cdot \log n_{\mathcal{T}})}$*

where p is a polynomial. The construction takes time polynomial in the size of Π_Q.

Proof. We convert Q into an MDDLog program Π_0 as per Theorem 8.3 and then remove the answer variables according to the constructions in the proofs of Lemmas 7.3 and 7.4, which gives programs Π_1 and Π_2. Analyzing the latter constructions reveals that the number of rules on Π_1 is bounded by $r \cdot a^a$ rules where r is the number of rules in Π_0 and a is its arity. Moreover, the rule size does not increase and neither the IDB schema nor the EDB schema changes. The latter construction produces a program with $r' \cdot s^s$ rules where r' is the number of rules in Π_1 and s is the rule size of Π_1. Moreover, the IDB schema is not changed and the rule size at most doubles. The EDB schema of the new program comprises a fresh monadic relation symbols. It can thus be verified that the obtained Boolean MDDLog program Π_2 still satisfies Conditions 1-3 of Theorem 8.3 except that n_q in the last point has to be replaced by $2n_q$. We make this explicit for the reader's convenience:

(1) the size of Π_2 is bounded by $2^{2^{p(n_q \cdot \log n_{\mathcal{T}})}}$;
(2) the IDB schema of Π_2 is of size at most $2^{p(n_q \cdot \log n_{\mathcal{T}})}$;
(3) the rule size of Π_2 is bounded by $2n_q$.

We next convert Π_2 into a simple Boolean MDDLog program Π_Q according to Theorem 3.2. Let us analyze the construction in detail to understand the size of Π_Q, of its EDB schema $\mathbf{S}'_E$, and of its IDB schema $\mathbf{S}'_I$.

The initial variable identification step can be ignored. In fact, we start with at most $2^{2^{p(n_q \cdot \log n_{\mathcal{T}})}}$ rules, each of size at most $2n_q$. Thus variable identification results in a factor of $(2n_q)!$ regarding the program size and rule number, which is absorbed by $2^{2^{p(n_q \cdot \log n_{\mathcal{T}})}}$, and the other relevant parameters do not change; in particular, the IDB schema remains unchanged.

The next and central step is to make rules biconnected. Given that the rule size is at most $2n_q$, this can split up each rule into at most $2n_q$ rules. This is absorbed by the bounds on program size and rule number. However, on first glance it might seem that we end up with a double exponentially large IDB schema since we might have to split up a double exponential number of rules, each time introducing at least one fresh IDB relation. To argue that this is actually not the case, we distinguish rules of type 1-2 and 4-5 from the construction of Π_1 (proof of Theorem 8.3); note that the constructions in the proofs of Lemmas 7.3 and 7.4 modify the rules only in a very mild way and thus for every rule in Π_2 it is still clear which type it has.

We need not worry about rules of Type 1-2 and 4-5 since there are only $2^{p(|n_q| \cdot \log |n_{\mathcal{T}}|)}$ many such rules, each of size at most $2n_q$, and thus the number of additional IDB relations introduced for making them biconnected is also bounded by $2^{p(n_q \cdot \log n_{\mathcal{T}})}$. Rules of type 3 in Π_0, on the other hand, are of a very restricted form, namely

$$\bot \leftarrow P_{C_1}(x) \wedge \cdots \wedge P_{C_n}(x)$$

with $C_1, \ldots, C_n \in \mathtt{sub}(\mathcal{T}) \cup \mathtt{con}(q_0)$. These rules are biconnected and thus we are done when Q is Boolean. In the non-Boolean case, rules of the above lead to the introduction of additional rules in the construction in the proof of Lemma 7.4. This results in rules in Π_2 that are of the form

$$\bot \leftarrow P_{C_1}(x_1) \wedge R_i(x_1) \wedge \cdots \wedge P_{C_n}(x_n) \wedge R_i(x_n)$$

where R_a is one of the fresh IDB relations introduced in the mentioned construction. The latter rules have to be split up to be made biconnected. This will result in rules of the form

$$\bot \leftarrow Q_1() \wedge \cdots \wedge Q_n() \quad \text{and} \quad Q_i() \leftarrow P_C(x) \wedge R_a(x)$$

Clearly, there are only $2^{p(n_q \cdot \log n_{\mathcal{T}})}$ many rule bodies of the latter form and thus it suffices to introduce at most the same number of fresh IDB relations Q_i. Thus, the size of the IDB schema of Π_Q is bounded by $2^{p(n_q \cdot \log n_{\mathcal{T}})}$. Also note that, at this Point, the rule size has (potentially) decreased and is bounded by $\max\{n_q, 2\}$. This is obvious for rules of Type 1-2 and 4-5, and also for the rules obtained from making rules of Type 3 biconnected, see above.

The last step is the change of EDB schema. It involves no blowups and we thus obtain the bounds stated in Lemma 8.4. In particular, the arity of relations in $\mathbf{S}'_E$ is bounded by $\max\{n_q, 2\}$ since it is bounded by the rule size of the program that we had obtained before changing the EDB schema. □