Flag & Check: Data Access with Monadically Defined Queries

Sebastian Rudolph
Fakultät Informatik
Technische Universität Dresden, DE
sebastian.rudolph@tu-dresden.de

Markus Krötzsch
Department of Computer Science
University of Oxford, UK
markus.kroetzsch@cs.ox.ac.uk

ABSTRACT

We introduce *monadically defined queries* (MODEQs) and *nested monadically defined queries* (NEMODEQs), two querying formalisms that extend conjunctive queries, conjunctive two-way regular path queries, and monadic Datalog queries. Both can be expressed as Datalog queries and in monadic second-order logic, yet they have a decidable query containment problem and favorable query answering complexities: a data complexity of P, and a combined complexity of NP (MODEQs) and PSpace (NEMODEQs).

We show that (NE)MODEQ answering remains decidable in the presence of a well-known generic class of tuple-generating dependencies. In addition, techniques to rewrite queries under dependencies into (NE)MODEQs are introduced. Rewriting can be applied partially, and (NE)MODEQ answering is still decidable if the non-rewritable part of the TGDs permits decidable (NE)MODEQ answering on other grounds.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*query languages*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*computational logic*

Keywords

query containment; tuple-generating dependencies; Datalog

1. INTRODUCTION

Query languages are fundamental to the design of database systems. A good query language should be able to express a wide range of common information needs, and allow queries to be answered efficiently with limited computational resources. Moreover, databases are often considered in combination with dependencies, e.g., in the form of *tuple-generating dependencies* (TGDs), which are also playing an important role in data exchange, information integration, and database integrity checking [1]. While query answering under dependencies is undecidable in general, there are many decidable cases, and a query language should be robustly applicable to such extensions [39]. Another important task in database management and optimization is to check *query containment*, that is, to determine whether the answers of one query are contained in the answers of another query over arbitrary databases, possibly under the additional assumption that certain constraints are satisfied. A query language should therefore allow for such checks.

Unfortunately, these basic requirements are in conflict. Very simple query languages like *conjunctive queries* (CQs, [17]) allow for efficient query answering (NP combined/AC_0 data[1]) and containment checking, but have very limited expressivity. First-order logic (FOL) queries extend expressivity, but are still restricted to "local" queries, excluding, e.g., the transitive closure of a relation. Query containment is undecidable for FOL, and query answering becomes PSpace-complete for combined complexity [44], but remains AC_0 for data [31]. Another extension of CQs is *Datalog*, which introduces rule-based recursion. The price are higher complexities (ExpTime combined/P data), and undecidability of query containment [23, 43]. FOL and Datalog are incomparable; both are subsumed by second-order logic (SO), which is more expressive but also more complex (ExpSpace combined/PH data) [31, 45].

To find more tractable query languages, various smaller fragments of Datalog have been considered. *Linear Datalog* allows only one inferred predicate per rule body, which significantly reduces query complexity (PSpace combined/ NLogSpace data) [29]. Still, query containment remains undecidable. Two query languages for which containment is decidable are *monadic Datalog* and *conjunctive 2-way regular path queries* (C2RPQs) [27, 15]. The query complexity of C2RPQs (NP combined/NLogSpace data) is slightly lower than that of monadic Datalog (NP combined/P data), but the expressivity of the languages is incomparable. In particular, monadic Datalog cannot express transitive closure. Two well-known query languages subsuming monadic Datalog and C2RPQs are Datalog and *monadic second-order logic* (MSO) [36]. Query containment is decidable for neither of these. Both languages are incomparable, even regarding query complexities (MSO has PSpace combined/PH data [44, 46, 36]), their common upper bound being SO.

This reveals a glaring gap in the landscape of known query languages: no formalism that captures monadic Datalog and C2RPQs ensures tractable data complexity and decidable query containment. To address this, we propose *monadically defined queries* (MODEQs) and *nested monadically defined queries* NEMODEQs as novel query formalisms that combine these desirable properties. Their relationship to the aforementioned languages is also illustrated in Fig. 1.

The contribution of this paper can be split in two parts. In the first part, we introduce the new querying formalism and clarify its relations to established query notions and the complexity for query answering. More precisely:

- We define MODEQs, discuss the underlying intuition, and provide examples demonstrating their expressivity.

[1]As usual, query complexities refer to the problem of deciding whether a query has a particular certain answer. *Combined complexity* is the complexity for arbitrary queries and databases; *data complexity* is the complexity if the query is fixed or bounded.

PODS'13, June 22–27, 2013, New York, New York, USA.

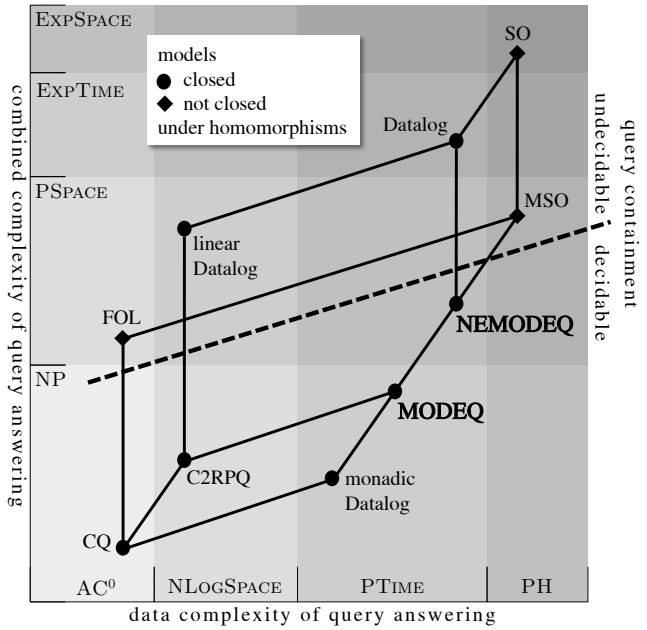

Figure 1: Overview of complexities and relations of monadically defined queries to other query formalisms as established in this paper; information indicated for conjunctive queries (CQ) and conjunctive 2-way regular path queries (C2RPQ) also hold when allowing unions (UCQ and UC2RPQ); all other formalisms are closed under unions

- We show that MODEQs capture (unions of) CQs, C2RPQs, and monadic Datalog queries.
- We prove MODEQ answering to be NP-complete for combined complexity (i.e., on par with CQs) and P-complete for data complexity, ensuring data-tractability as one of the central desiderata for querying large data sets.

We then extend MODEQs with nested subqueries, leading to a broader class of NEMODEQs that generalize MODEQs.

- We show that NEMODEQs (thus MODEQs) are expressible by both Datalog queries and MSO formulae.
- We prove NEMODEQ answering to be PSpace-complete for combined complexity (on par with FOL-based query languages like SQL) and P-complete for data complexity.
- We show that, unlike for Datalog and MSO queries, query containment for (NE)MODEQs is decidable.

In the second part of the paper, we study (NE)MODEQs in the context of dependencies and ontology-based data access. To this end, an important tool are (finite or infinite) *universal models*, which represent solutions to data exchange and constraint repair problems in the presence of TGDs [24]. As models of (NE)MODEQ are closed under homomorphisms, we find that universal models can be used to answer such queries, making them very robust to a wide class of TGDs.

- We immediately obtain decidability of query answering under all TGDs that admit a finite universal model. This property of TGDs is undecidable, but can be approximated by various notions of *acyclicity* [25, 26, 24, 38, 30, 34].
- More generally, we show that MODEQ answering is decidable in the presence of rules giving rise to (possibly infinite) universal models of *bounded treewidth*. This applies to many lightweight ontology languages as well as guarded TGDs and generalizations thereof [9, 4, 34, 5].
- In analogy to the popular notion of first-order rewritability, we introduce (NE)MODEQ rewritability, and we identify basic criteria for rewriting Datalog rules.
- Finally, we show that query answering is decidable under any set of TGDs that can be decomposed into one that is (NE)MODEQ-rewritable and one with the bounded-treewidth-model property.

Proofs omitted in the main paper are given in the accompanying technical report [42].

2. PRELIMINARIES

We consider a standard language of first-order predicate logic, based on an infinite set $\mathbf{C}$ of *constant symbols*, an infinite set $\mathbf{P}$ of *predicate symbols*, and an infinite set $\mathbf{V}$ of first-order *variables*. Each predicate $p \in \mathbf{P}$ is associated with a natural number $\mathsf{ar}(p)$ called the *arity* of p. The list of predicates and constants forms the language's *signature* $\mathscr{S} = \langle \mathbf{P}, \mathbf{C} \rangle$. We generally assume $\mathscr{S} = \langle \mathbf{P}, \mathbf{C} \rangle$ to be fixed, and only refer to it explicitly if needed.

Databases, Rules, and Queries. A *term* is a variable $x \in \mathbf{V}$ or a constant $c \in \mathbf{C}$. We use symbols s, t to denote terms, x, y, z, v, w to denote variables, a, b, c to denote constants. Expressions like $\boldsymbol{t}$, $\boldsymbol{x}$, $\boldsymbol{c}$ denote finite lists of such entities. We use the standard predicate logic definitions of *atom* and *formula*, using symbols φ, ψ for the latter. A formula is *ground* if it contains no variables. A *database*, usually denoted by D, is a finite set of ground atoms. We write $\varphi[\boldsymbol{x}]$ to emphasize that a formula φ has free variables $\boldsymbol{x}$; we write $\varphi[\boldsymbol{c}/\boldsymbol{x}]$ for the formula obtained from φ by replacing each variable in $\boldsymbol{x}$ by the respective constant in $\boldsymbol{c}$ (both lists must have the same length). A formula without free variables is a *sentence*.

A *conjunctive query* (CQ) is a formula $Q[\boldsymbol{x}] = \exists \boldsymbol{y}.\psi[\boldsymbol{x}, \boldsymbol{y}]$ where $\psi[\boldsymbol{x}, \boldsymbol{y}]$ is a conjunction of atoms. A *tuple-generating dependency* (TGD) is a formula of the form $\forall \boldsymbol{x}, \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z}.\psi[\boldsymbol{x}, \boldsymbol{z}]$ where φ and ψ are conjunctions of atoms, called the *body* and *head* of the TGD, respectively. TGDs never have free variables, so we usually omit the universal quantifier when writing them. We use the symbol Σ, possibly with subscripts, to denote sets of TGDs. A *Datalog rule* is a TGD without existentially quantified variables; sets of Datalog rules will be denoted by symbols $\mathbb{P}, \mathbb{R}, \mathbb{S}$.

We use the standard semantics of first-order logic (FOL). An *interpretation* $\mathcal{I}$ consists of a (possibly infinite) set $\Delta^{\mathcal{I}}$ called *domain* and a function $\cdot^{\mathcal{I}}$ that maps constants c to domain elements $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and predicate symbols p to relations $p^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^{\mathsf{ar}(p)}$, thereby $p^{\mathcal{I}}$ is called the *extension* of p. A *variable assignment* for $\mathcal{I}$ is a function $\mathcal{Z} : \mathbf{V} \rightarrow \Delta^{\mathcal{I}}$. Conditions for $\mathcal{I}$ and $\mathcal{Z}$ to satisfy a FOL formula φ (i.e., to be a *model* of φ, written $\mathcal{I}, \mathcal{Z} \models \varphi$) are defined as usual. If φ is a sentence, then $\mathcal{Z}$ is irrelevant for satisfaction and can be omitted. An *answer* to a CQ $Q[\boldsymbol{x}]$ over a database D and set Σ of TGDs is a list of constants $\boldsymbol{c}$ for which $D \cup \Sigma \models Q[\boldsymbol{c}/\boldsymbol{x}]$.

Given an interpretation $\mathcal{I}$ and a formula $\varphi[\boldsymbol{x}]$ with free variables $\boldsymbol{x} = \langle x_1, \ldots, x_m \rangle$, the *extension* of $\varphi[\boldsymbol{x}]$ is the subset of $(\Delta^{\mathcal{I}})^m$ containing all those tuples $\langle \delta_1, \ldots, \delta_m \rangle$ for which $\mathcal{I}, \{x_i \mapsto \delta_i \mid 1 \leq i \leq m\} \models \varphi[\boldsymbol{x}]$. Two formulae $\varphi[\boldsymbol{x}]$ and $\psi[\boldsymbol{x}]$ with the same free variables $\boldsymbol{x}$ are called *equivalent* if their extensions coincide for every interpretation $\mathcal{I}$.

Homomorphisms and Universal Models. Given interpretations $\mathcal{I}$, $\mathcal{J}$, a *homomorphism* π from $\mathcal{I}$ to $\mathcal{J}$ is a function $\pi : \Delta^{\mathcal{I}} \rightarrow \Delta^{\mathcal{J}}$ such that: (i) for all constants c, we have $\pi(c^{\mathcal{I}}) = c^{\mathcal{J}}$, and (ii) for all predicate symbols p and list of domain elements $\boldsymbol{\delta}$, we have $\boldsymbol{\delta} \in p^{\mathcal{I}}$ implies $\pi(\boldsymbol{\delta}) \in p^{\mathcal{J}}$.

Finding query answers is facilitated in practice since one may focus on universal models. A *universal model* of a set of sentences Ψ is an interpretation $\mathcal{I}$ such that (i) $\mathcal{I} \models \Psi$, and (ii) for every interpretation $\mathcal{J}$ with $\mathcal{J} \models \Psi$, there is a homomorphism from $\mathcal{I}$ to $\mathcal{J}$. For TGDs (and for plain databases), there is always a universal

model if there is any model at all. It can be defined by a (possibly infinite) construction process called the *chase*. In particular, we let $\mathcal{I}(D \cup \Sigma)$ denote the universal model, for which every homomorphism into any other model of $D \cup \Sigma$ is injective. $\mathcal{I}(D \cup \Sigma)$ always exists for satisfiable $D \cup \Sigma$, and it is unique up to isomorphism [24].

For a wide range of queries, entailment of query answers can be reduced to model checking in the universal model:

Fact 1 (Entailment via model checking). *If $Q[\boldsymbol{x}]$ is a query for which the set of models of $\exists \boldsymbol{x}.Q[\boldsymbol{x}]$ is closed under homomorphisms, then, for every database D and set Σ of TGDs, $D \cup \Sigma \models Q[\boldsymbol{c}/\boldsymbol{x}]$ if and only if either $D \cup \Sigma$ is inconsistent or $\mathcal{I}(D \cup \Sigma) \models Q[\boldsymbol{c}/\boldsymbol{x}]$.*

This applies to CQs and to all other query languages studied herein. The case $\Sigma = \emptyset$ shows that one can equivalently represent databases using models $\mathcal{I}(D)$ instead of sets of facts D. Our perspective is more natural when using TGDs.

3. MONADICALLY DEFINED QUERIES

We now introduce a new query formalism, called *monadically defined queries (MODEQs)*, and state complexity results on query answering in this language. To deepen our understanding for the expressivity of MODEQs, we show that they strictly generalize the well-known query formalisms of *conjunctive 2-way regular path queries* (Section 3.1) and *monadic Datalog queries* (Section 3.2).

The heart of our query formalism is a mechanism for defining new predicates based on existing ones. This mechanism – which we refer to as *"flag & check"* – specifies new predicates by providing a procedure for testing if a particular tuple $\boldsymbol{\delta}=\langle \delta_1, \ldots, \delta_m \rangle \in (\Delta^{\mathcal{I}})^m$ is in the predicate's extension or not. To this end, the candidate tuple is first "flagged" by associating each δ_i with an auxiliary constant name λ_i that represents this element. The "check" is performed by running a Datalog program with this fixed interpretation of the constants λ_i. The check succeeds if a special fact `hit` is derived.

Example 1. *To illustrate the idea, we consider a typical transitive closure query. Suppose that the binary predicate certifiedBy represents the direct certification of one entity by another, e.g., in a security application. We are interested in certification chains, which could be expressed in Datalog as follows:*

$$certifiedBy(x, y) \rightarrow certChain(x, y) \quad (1)$$
$$certChain(x, y) \wedge certifiedBy(y, z) \rightarrow certChain(x, z) \quad (2)$$

Corresponding Datalog rules $\mathbb{P}_1$ for "flag & check" are:

$$certifiedBy(\lambda_1, y) \rightarrow \mathtt{U}_1(y) \quad (3)$$
$$\mathtt{U}_1(y) \wedge certifiedBy(y, z) \rightarrow \mathtt{U}_1(z) \quad (4)$$
$$\mathtt{U}_1(\lambda_2) \rightarrow \mathtt{hit} \quad (5)$$

We define certChain to contain all pairs $\langle \delta_1, \delta_2 \rangle$ for which $\mathbb{P}_1$ entails `hit` *when interpreting λ_1 as δ_1 and λ_2 as δ_2.*

As in Example 1, the Datalog rules that we consider for the checking phase only use `hit` or new, unary predicates $\mathtt{U}_i$ in rule heads. Such unary predicates can be imagined as "colors" that are assigned to elements of the domain, and the check thus is a deterministic, recursive procedure of coloring the domain, starting from the flagged candidate elements. This idea is defined formally as follows.

Definition 1 (MODEQ syntax and semantics). *Given a signature $\mathscr{S}$, a* monadically defined predicate (MODEP) *of arity m is based on a signature $\mathscr{S}'$ that extends $\mathscr{S}$ with m fresh constant symbols $\lambda_1, \ldots, \lambda_m$, a fresh nullary predicate* `hit`*, and $k \geq 0$ fresh unary predicates $\mathtt{U}_1, \ldots, \mathtt{U}_k$. A MODEP is a set $\mathbb{P}$ of Datalog rules over $\mathscr{S}'$ where only $\mathtt{U}_1, \ldots, \mathtt{U}_k$, and* `hit` *occur in rule heads.*

Let $\mathcal{I}$ be an interpretation over $\mathscr{S}$. The extension $\mathbb{P}^{\mathcal{I}}$ of $\mathbb{P}$ is the set of all tuples $\langle \delta_1, \ldots, \delta_m \rangle \in (\Delta^{\mathcal{I}})^m$ for which $\mathcal{I}' \models \mathbb{P}$ implies $\mathcal{I}' \models$ `hit`*, for all interpretations $\mathcal{I}'$ that extend $\mathcal{I}$ to the symbols in $\mathscr{S}'$ such that $\langle \lambda_1^{\mathcal{I}'}, \ldots, \lambda_m^{\mathcal{I}'} \rangle = \langle \delta_1, \ldots, \delta_m \rangle$.*

A monadically defined query (MODEQ) *is a conjunctive query that uses both normal predicates and monadically defined predicates in its atoms. The semantics of MODEQs is defined in the obvious way via the semantics of MODEPs.*

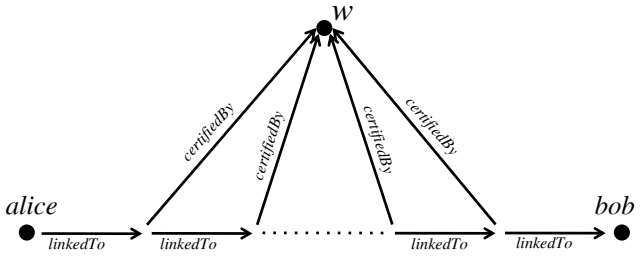

Figure 2: Structure recognized by MODEQ $\mathfrak{Q}_2$ in Example 2

Example 2. *The rules* (3)–(5) *define a binary MODEP $\mathbb{P}_1$, so the query of Example 1 can be written as a MODEQ $\mathfrak{Q}_1[v, w] = \mathbb{P}_1(v, w)$. For another example, assume there are entities certifying the security of message handling in certain nodes of a network. We are interested in entities y that can (directly) certify secure treatment of the message at all nodes on some path from Alice to Bob, as illustrated in Fig. 2. This is expressed by the MODEQ $\mathfrak{Q}_2[w] = \mathbb{P}_2(w)$, where $\mathbb{P}_2$ consists of the following rules:*

$$certifiedBy(x, \lambda_1) \rightarrow \mathtt{U}_3(x) \quad (6)$$
$$linkedTo(alice, x) \rightarrow \mathtt{U}_2(x) \quad (7)$$
$$\mathtt{U}_2(x) \wedge \mathtt{U}_3(x) \wedge linkedTo(x, x') \rightarrow \mathtt{U}_2(x') \quad (8)$$
$$\mathtt{U}_2(bob) \rightarrow \mathtt{hit} \quad (9)$$

This new query notion is rather powerful. It is easy to see that it subsumes conjunctive queries (CQs) as well as unions of CQs. Indeed, given k CQs $\exists \boldsymbol{y}_i.Q_i[\boldsymbol{x}, \boldsymbol{y}_i]$ with $i \in \{1, \ldots, k\}$, their union is expressed as the MODEQ $\{Q_i[\boldsymbol{\lambda}, \boldsymbol{y}_i] \rightarrow \mathtt{hit} \mid 1 \leq i \leq k\}(\boldsymbol{x})$. Before showing that MODEQs also capture more powerful query languages, we observe closure under homomorphism (see Fact 1) and state a basic complexity result.

Theorem 2. *For every MODEQ $\mathfrak{Q}[\boldsymbol{x}]$, the set of all models of $\exists \boldsymbol{x}.\mathfrak{Q}[\boldsymbol{x}]$ is closed under homomorphisms.*

Theorem 3 (MODEQ answering complexity). *Testing if $\boldsymbol{c}$ is an answer to a MODEQ $\mathfrak{Q}[\boldsymbol{x}]$ over a database D is* P*-complete in the size of D, and* NP*-complete in the combined size of D and $\mathfrak{Q}$.*

Hardness follows from the fact that MODEQs subsume monadic Datalog, shown in Section 3.2 below. P membership for data complexity is a consequence of the fact that Datalog subsumes MODEQs, demonstrated in Section 4.1. Membership in NP for combined complexity is established directly by showing that every query match is witnessed by a proof that can be guessed and verified in polynomial time.

3.1 MODEQs Capture Regular Path Queries

We now show that MODEQs subsume *conjunctive two-way regular path queries (C2RPQs)*, which generalize CQs by regular expressions over binary predicates [27, 15]. Variants of this type of queries are used, e.g., by the XPath query language for querying semi-structured XML data. Recent versions of the SPARQL 1.1 query language for RDF also support some of regular expressions that can be evaluated under a similar semantics.

C2RPQs are defined like MODEQs, but with MODEPs replaced by another form of defined predicates based on regular expressions over binary predicates and their inverses:

Definition 2 (C2RPQ syntax and semantics). *A* two-way regular path predicate (2RPP) *is a regular expression over the alphabet* $\Gamma = \{p, p^- \mid \mathsf{ar}(p) = 2\}$ *of normal and inverse binary predicate symbols. All 2RPPs are of arity 2. Consider an interpretation* $\mathcal{I}$*. For inverse predicates* p^-*, we define* $(p^-)^{\mathcal{I}} := \{\langle \delta_2, \delta_1\rangle \mid \langle \delta_1, \delta_2\rangle \in p^{\mathcal{I}}\}$*. For a 2RPP* P*, we set* $\langle \delta, \delta'\rangle \in P^{\mathcal{I}}$ *if there is a word* $\gamma_1 \dots \gamma_n$ *matching the regular expression* P*, and a sequence* $\delta_0 \dots \delta_n$ *of domain elements such that* $\delta_0 = \delta$*,* $\delta_n = \delta'$*, and* $\langle \delta_i, \delta_{i+1}\rangle \in \gamma_i^{\mathcal{I}}$ *for every* $i \in \{0, \dots, n-1\}$*.*

A conjunctive two-way regular path query (C2RPQ) *is a conjunctive query that uses both normal predicates and 2RPPs in its atoms. The semantics of C2RPQs is defined in the obvious way based on the semantics of 2RPPs.*

Example 3. *The query of Example 1 is expressed by the C2RPQ* $certifiedBy^*(x, y)$*. Another C2RPQ with inverses is*

$$mountain(x) \wedge continent(y) \wedge (locatedIn|hasPart^-)^*(x, y). \tag{10}$$

Query answering for C2RPQs is NP-complete regarding the size of the database and query, which is the same as for CQs. In terms of data complexity, C2RPQs are NLogSpace-complete, and thus harder than CQs (AC_0). One can show hardness via graph reachability, and membership via a translation to linear Datalog [13].

Definition 3 (C2RPQ to MODEQ translation). *Consider a 2RPP* P *and a finite automaton* $\mathcal{A}_P = \langle \Gamma, S, I, F, T\rangle$ *that recognizes* P*. The binary MODEP* $\mathsf{modep}(P)$ *consists of the rules*

$$\begin{aligned}
&\to \mathtt{U}_s(\lambda_1) && \text{for every initial state } s \in I,\\
\mathtt{U}_s(z) \wedge p(z, z') &\to \mathtt{U}_{s'}(z') && \text{for every transition } \langle s, p, s'\rangle \in T,\\
\mathtt{U}_s(z) \wedge p(z', z) &\to \mathtt{U}_{s'}(z') && \text{for every transition } \langle s, p^-, s'\rangle \in T,\\
\mathtt{U}_s(\lambda_2) &\to \mathtt{hit} && \text{for every final state } s \in F.
\end{aligned}$$

Given a C2RPQ Q*, a MODEQ* $\mathsf{modeq}(Q)$ *is obtained by replacing every 2RPP* P *in* Q *by* $\mathsf{modep}(P)$*.*

The intuition behind the translation of C2RPQs to MODEQs is to find bindings for x and y in $P(x, y)$ by simulating all possible runs of the automaton corresponding to a C2RPQ. Colors $\mathtt{U}_s$ are associated to states s of the automaton to record which domain elements can be reached in which states when starting in an initial state at x. The success criterion is that y is colored by a final state. One can thus show the following:

Theorem 4 (MODEQs capture C2RPQs). *For every C2RPQ* Q*, the MODEQ* $\mathsf{modeq}(Q)$ *can be constructed in linear time, and is equivalent to* Q*. In particular, the answers for* Q *and* $\mathsf{modeq}(Q)$ *coincide.*

Example 4. *Let* Q *be the regular path query* (10)*. The corresponding MODEQ* $\mathsf{modeq}(Q)$ *is* $mountain(x) \wedge continent(y) \wedge \mathbb{P}(x, y)$ *where* $\mathbb{P} = \mathsf{modep}((locatedIn|hasPart^-)^*)$ *consists of the rules:*

$$\to \mathtt{U}(\lambda_1) \tag{11}$$
$$\mathtt{U}(z) \wedge locatedIn(z, z') \to \mathtt{U}(z') \tag{12}$$
$$\mathtt{U}(z) \wedge hasPart(z', z) \to \mathtt{U}(z') \tag{13}$$
$$\mathtt{U}(\lambda_2) \to \mathtt{hit}. \tag{14}$$

Here we only need one "color" $\mathtt{U}$ *that is propagated over locatedIn and inversely over hasPart. The pairs in* $\mathbb{P}$ *are those for which this process, started at the first element, will eventually color the second argument.*

However, the expressivity of MODEQs goes well beyond that of C2RPQs, even when considering only binary predicates. This follows from the easy observation that for every C2RPQ Q there is an integer n, such that whenever Q matches into a graph G, it also matches into a graph G' where all vertices have degree $\leq n$ and from which there is a homomorphism into G. It is easy to see that the MODEQ $\mathfrak{Q}_2$ from Example 2 does not have this property.

3.2 MODEQs Capture Monadic Datalog

Monadic Datalog queries are another type of query language that enjoys favorable computational properties. They are used, e.g., for information extraction from the Web [28]. We now show that MODEQs can express monadic Datalog queries, which is another way to see that they are strictly more general than C2RPQs.

Definition 4 (Monadic Datalog query). *Given a signature* $\mathscr{S}$*, a* Datalog query *is based on a signature* $\mathscr{S}'$ *that extends* $\mathscr{S}$ *with additional predicates, called* intensional database (IDB) predicates*. A Datalog query is a pair* $\langle \mathtt{goal}, \mathbb{S}\rangle$*, where* goal *is an IDB predicate, and* $\mathbb{S}$ *is a set of Datalog rules over* $\mathscr{S}'$ *where only IDB predicates occur in rule heads and* goal *does not occur in rule bodies. A* monadic Datalog query *is a query where all IDB predicates other than* goal *have arity* 1*.*

Given an interpretation $\mathcal{I}$*, the* extension *of* $\langle \mathtt{goal}, \mathbb{S}\rangle$ *is the set of tuples* $\boldsymbol{\delta}$ *over* $\Delta^{\mathcal{I}}$ *for which every extension* $\mathcal{I}'$ *of* $\mathcal{I}$ *to* $\mathscr{S}'$ *which satisfies* $\mathbb{S}$ *must also satisfy* $\boldsymbol{\delta} \in \mathtt{goal}^{\mathcal{I}'}$*.*

Note that sometimes in the literature, the arity of goal is restricted to 1. Allowing it to be arbitrary does not affect the complexity of the formalism.

Definition 5 (Monadic Datalog to MODEQ). *Given a monadic Datalog query* $Q = \langle \mathtt{goal}, \mathbb{S}\rangle$*, we let* $\mathsf{modeq}(Q)$ *denote the MODEQ* $\mathbb{P}(\boldsymbol{x})$ *where* $\mathbb{P}$ *is obtained from* $\mathbb{S}$ *by*

- *replacing each rule* $\varphi[\boldsymbol{x}, \boldsymbol{y}] \to \mathtt{goal}(\boldsymbol{x})$ *by* $\varphi[\boldsymbol{\lambda}, \boldsymbol{y}] \to \mathtt{hit}$*,*
- *replacing each IDB predicate, uniformly and injectively, by a predicate* $\mathtt{U}_j$*.*

Theorem 5 (MODEQs capture monadic Datalog). *For every monadic Datalog query* Q*, the MODEQ* $\mathsf{modeq}(Q)$ *can be constructed in linear time, and is equivalent to* Q*. In particular, the answers for* Q *and* $\mathsf{modeq}(Q)$ *coincide.*

From the correspondence thus established it follows that the lower complexity bounds of monadic Datalog carry over and MODEQ answering on databases must thus be P-hard for data complexity and NP-hard for combined complexity [28], showing one direction of Theorem 3. MODEQs are strictly more expressive than monadic Datalog queries, shown by the fact that even a simple connectedness query like the one in Example 1 cannot be expressed in monadic Datalog.

We would like to specifically note that, although the rules used in the definitions of MODEPs are in fact monadic Datalog rules, the query evaluation schemes underlying monadic Datalog and MODEQs are fundamentally different: while in the case of monadic Datalog, all elements of the extension can be obtained at once by a forward chaining saturation process on the given interpretation (or database), the *flag & check* strategy that underlies the semantics definition of MODEPs crucially hinges on each potential extension element being verified in a *separate* saturation process. In Section 4.1, we will see that this idea can be captured by Datalog queries, but not by monadic ones.

4. NESTED MODEQS

Query nesting is the process of using an n-ary subquery instead of an n-ary predicate symbol within a query, with the obvious semantics. In this section, we use this mechanism to extend MODEQs, leading to the more general language of *nested monadically defined queries (NEMODEQs)*. We then show that queries of this type can be expressed in Datalog (Section 4.1) and monadic second-order logic (Section 4.2). These results extend to MODEQs as a special case of NEMODEPs, and help to establish some additional upper bounds for complexity.

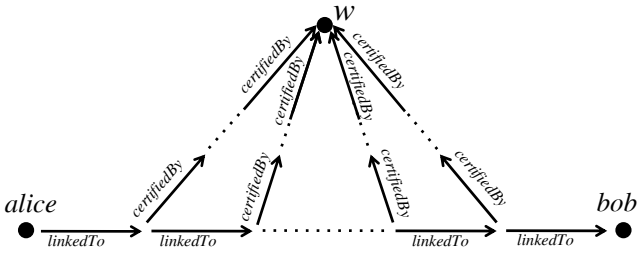

Figure 3: Structure recognized by $\mathfrak{Q}_3$ in Example 5

It is interesting to ask if nesting of queries actually leads to a new query language or not. A query language is *closed under nesting* if every query with nested subqueries can be expressed by some non-nested query. Many query languages are trivially closed under nestings as they allow nesting as part of their syntax (e.g., CQs, FOL, MSO, and SO), other languages require more or less complex reformulations to eliminate nested queries (e.g., UCQs, Datalog, and monadic Datalog), others are not closed under nestings (e.g., linear Datalog). Example 6 below shows that MODEQs are not closed under nestings, motivating the following definition.

Definition 6 (NEMODEQ syntax & semantics). *Let $\mathscr{S}$ be the underlying signature. A* nested monadically defined predicate (NEMODEP) *of degree* 1 *over $\mathscr{S}$ is a MODEP over $\mathscr{S}$. Consider a finite set $\mathbb{P}_i$ ($i = 1, \ldots, k$) of NEMODEPs of degree $\leq d$ over $\mathscr{S}$. A NEMODEP of degree $d+1$ is a MODEP over a signature $\mathscr{S}'$ that extends $\mathscr{S}$ with additional predicate names $\mathbb{P}_i$ that have the same arity as the respective queries.*

The semantics of NEMODEPs of degree $d > 1$ is defined as for MODEPs, based on the (recursively defined) semantics of NEMODEPs of degree $< d$. A nested monadically defined query (NEMODEQ) *is a conjunctive query that uses both normal predicates and nested monadically defined predicates in its atoms. The semantics of NEMODEQs is defined in the obvious way.*

Note that the auxiliary symbols λ_i, $\mathtt{U}_j$, and `hit` do not need to be distinct in different subqueries, or in queries and their subqueries. This does not cause semantic interactions.

Example 5. *Consider again Example 2, and assume that we are now also interested in entities that can certify the security of a communication indirectly, i.e., through a chain of certifications, as shown in Fig. 3. This can be expressed by nesting MODEP $\mathbb{P}_1$ in $\mathfrak{Q}_2$, leading to a NEMODEQ of degree 2 $\mathfrak{Q}_3 = \mathbb{P}_3(w)$, where $\mathbb{P}_3$ coincides with $\mathbb{P}_2$ except that rule* (6) *is replaced by $\mathbb{P}_1(x, \lambda_1) \rightarrow \mathtt{U}_3(x)$.*

However, even if a query is more easily expressed as a NEMODEQ, it might still be expressible as a MODEQ. The next example shows that this is the case for $\mathfrak{Q}_3$ above, and presents a query that cannot be expressed by any MODEQ.

Example 6. *$\mathfrak{Q}_3$ of Example 5 can equivalently be expressed by the MODEQ $\mathfrak{Q}_4[w] = \mathbb{P}_4(w)$, where $\mathbb{P}_4$ consists of the following rules:*

$$\mathit{certifiedBy}(x, \lambda_1) \rightarrow \mathtt{U}_3(x) \quad (15)$$
$$\mathtt{U}_3(y) \wedge \mathit{certifiedBy}(x, y) \rightarrow \mathtt{U}_3(x) \quad (16)$$
$$\mathit{linkedTo}(\mathit{alice}, x) \rightarrow \mathtt{U}_2(x) \quad (17)$$
$$\mathtt{U}_2(x) \wedge \mathtt{U}_3(x) \wedge \mathit{linkedTo}(x, x') \rightarrow \mathtt{U}_2(x') \quad (18)$$
$$\mathtt{U}_2(\mathit{bob}) \rightarrow \mathtt{hit} \quad (19)$$

To define a NEMODEQ that cannot be expressed as a MODEQ, we first modify this example to ask for all pairs of persons who are connected by a communication chain that is certified by a single entity, i.e., we query for the possible pairs of "alice" and "bob." Let $\mathbb{P}_5$ be the ternary MODEP that consists of the following rules:

$$\mathit{certifiedBy}(x, \lambda_1) \rightarrow \mathtt{U}_3(x) \quad (20)$$
$$\mathtt{U}_3(y) \wedge \mathit{certifiedBy}(x, y) \rightarrow \mathtt{U}_3(x) \quad (21)$$
$$\mathit{linkedTo}(\lambda_2, x) \rightarrow \mathtt{U}_2(x) \quad (22)$$
$$\mathtt{U}_2(x) \wedge \mathtt{U}_3(x) \wedge \mathit{linkedTo}(x, x') \rightarrow \mathtt{U}_2(x') \quad (23)$$
$$\mathtt{U}_2(\lambda_3) \rightarrow \mathtt{hit} \quad (24)$$

We now form a NEMODEQ that asks for all pairs of persons who can communicate through a chain of such secure channels that goes via multiple people. Moreover, we require that all of these people are "friends," that is, trustworthy in the context of the application. Let $\mathbb{P}_6$ be the binary NEMODEP with the following rules:

$$\rightarrow \mathtt{U}_1(\lambda_1) \quad (25)$$
$$\mathtt{U}_1(y) \wedge \mathbb{P}_5(x, y, z) \wedge \mathit{friend}(z) \rightarrow \mathtt{U}_1(z) \quad (26)$$
$$\mathtt{U}_1(y) \wedge \mathbb{P}_5(x, y, \lambda_2) \rightarrow \mathtt{hit} \quad (27)$$

The NEMODEQ $\mathfrak{Q}_4 = \mathbb{P}_6(v, w)$ (see Fig. 4) cannot be expressed as a MODEQ. To show this, one assumes the existence of such a MODEQ and constructs a database where it must accept a match that is not accepted by $\mathfrak{Q}_4$. Details are given in the technical report.

4.1 Expressing NEMODEQs in Datalog

We now show that NEMODEQs of arbitrary degree can be expressed as Datalog queries. To this end, the auxiliary predicates have to be "contextualized," which increases their arity. Hence the translation usually does not lead to monadic Datalog queries.

Definition 7 (NEMODEQ to Datalog). *Given a MODEP $\mathbb{P}$ of arity m, the set* $\mathsf{datalog}(\mathbb{P})$ *of Datalog rules over an extended signature contains, for each rule in $\mathbb{P}$, a new rule obtained by replacing*

- *each constant λ_i with a variable x_{λ_i},*
- *each atom $\mathtt{U}_i(z)$ with the atom $\hat{U}_i(z, x_{\lambda_1}, \ldots, x_{\lambda_m})$ where $\hat{U}_i$ is a fresh predicate of arity $m+1$,*
- *each atom `hit` with the atom $p_{\mathbb{P}}(x_{\lambda_1}, \ldots, x_{\lambda_m})$ where $p_{\mathbb{P}}$ is a fresh predicate symbol of arity m.*

For a NEMODEP $\mathbb{P}$ of degree $d > 1$, let $\mathbb{P}'$ be the MODEP obtained by replacing each direct sub-NEMODEP $\mathbb{Q}$ of $\mathbb{P}$ with the predicate $p_{\mathbb{Q}}$. The Datalog translation of $\mathbb{P}$ is recursively defined as:

$$\mathsf{datalog}(\mathbb{P}) := \mathsf{datalog}(\mathbb{P}') \cup \bigcup_{\mathbb{Q} \text{ a direct sub-NEMODEP of } \mathbb{P}} \mathsf{datalog}(\mathbb{Q}).$$

Given a NEMODEQ $\mathfrak{Q}[\boldsymbol{x}] = \exists \boldsymbol{y}.\varphi[\boldsymbol{x}, \boldsymbol{y}]$, we define its translation $\mathsf{datalog}(\mathfrak{Q})$ *as* $\mathsf{datalog}(\{\varphi[\boldsymbol{\lambda}, \boldsymbol{y}] \rightarrow \mathtt{hit}\})$*, where the predicate used to replace `hit` will be denoted by $p_{\mathfrak{Q}}$.*

Note that the predicates $\hat{U}_i$ must be globally fresh, even if multiple subqueries use the same $\mathtt{U}_i$. The rules in $\mathsf{datalog}(\mathfrak{Q})$ might be unsafe, i.e., they may contain universally quantified variables in the head that do not occur in the body. This is no problem with the logical semantics we consider.

Theorem 6 (Datalog expressibility of NEMODEQs). *For any NEMODEQ $\mathfrak{Q}$,* $\mathsf{datalog}(\mathfrak{Q})$ *can be constructed in linear time. Moreover, the queries $\mathfrak{Q}[\boldsymbol{x}]$ and $\langle p_{\mathfrak{Q}}, \mathsf{datalog}(\mathfrak{Q})\rangle$ are equivalent, i.e., their answers coincide.*

Example 7. *The Datalog translation for the MODEQ $\mathfrak{Q}_3$ from Example 5 is as follows:*

$\mathsf{datalog}(\mathfrak{Q}_3)$

$\mathsf{datalog}(\mathbb{P}_3)$

$\mathsf{datalog}(\mathbb{P}_1)$

$$\begin{aligned}
\mathit{certifiedBy}(x_{\lambda_1}, y) &\rightarrow \hat{U}_1(y, x_{\lambda_1}, x_{\lambda_2}) \\
\hat{U}_1(y, x_{\lambda_1}, x_{\lambda_2}) \wedge \mathit{certifiedBy}(y, z) &\rightarrow \hat{U}_1(z, x_{\lambda_1}, x_{\lambda_2}) \\
\hat{U}_1(x_{\lambda_2}, x_{\lambda_1}, x_{\lambda_2}) &\rightarrow p_{\mathbb{P}_1}(x_{\lambda_1}, x_{\lambda_2}) \\
p_{\mathbb{P}_1}(x, x_{\lambda_1}) &\rightarrow \hat{U}_3(x, x_{\lambda_1}) \\
\mathit{linkedTo}(\mathit{alice}, x) &\rightarrow \hat{U}_2(x, x_{\lambda_1}) \\
\hat{U}_2(x, x_{\lambda_1}) \wedge \hat{U}_3(x, x_{\lambda_1}) \wedge \mathit{linkedTo}(x, x') &\rightarrow \hat{U}_2(x', x_{\lambda_1}) \\
\hat{U}_2(\mathit{bob}, x_{\lambda_1}) &\rightarrow p_{\mathbb{P}_3}(x_{\lambda_1}) \\
p_{\mathbb{P}_3}(x_{\lambda_1}) &\rightarrow p_{\mathfrak{Q}_3}(x_{\lambda_1})
\end{aligned}$$

Using backward-chaining, the goal $p_{\mathfrak{Q}}(\boldsymbol{x})$ can be expanded under the rules $\mathsf{datalog}(\mathfrak{Q})$ to obtain a (possibly infinite) set of CQs that do not contain auxiliary predicates $p_{\mathfrak{Q}'}$. Thus $\mathfrak{Q}$ can be considered as a union of (possibly infinitely many) conjunctive queries.

The linear translation of NEMODEQs (and thus also MODEQs) to Datalog leads to various results. First, NEMODEQs inherit Datalog's PTime upper bound for data complexity of query answering [23]. The results of Section 3.2 show that this bound is tight. Second, we find that the models of NEMODEQs are closed under homomorphisms, since Datalog has this property. Again, this shows that query entailment coincides with model checking (Fact 1).

Theorem 7. *For any NEMODEQ $\mathfrak{Q}[\boldsymbol{x}]$, the set of models of $\exists\boldsymbol{x}.\mathfrak{Q}$ is closed under homomorphisms.*

4.2 Expressing NEMODEQs in MSO Logic

In this section, we show that NEMODEQs can also be expressed in *monadic second-order logic (MSO)*, the extension of first-order logic with *set variables*, used like predicates of arity 1. To distinguish them from object variables x, y, z, we denote set variables by the uppercase letter U, possibly with subscripts, hinting at their close relation to the unary coloring predicates $\mathtt{U}$. We adhere to the standard semantics of MSO that we will not repeat here.

To simplify the presentation of the next definition, we henceforth assume that every variable x, constant λ_i, or monadic predicate $\mathtt{U}_j$ is used in at most one (sub-)predicate $\mathbb{P}$ of any NEMODEQ or NEMODEP we consider. This can always be achieved by renaming variables and predicates.

Definition 8 (NEMODEQ to MSO). *For a MODEP $\mathbb{P}$ of arity m with auxiliary unary predicates $\mathtt{U}_1, \ldots, \mathtt{U}_k$, and a list of terms $\boldsymbol{t} = \langle t_1, \ldots, t_m \rangle$, we define an MSO formula*

$$\mathsf{mso}(\mathbb{P}(\boldsymbol{t})) := \forall U_1, \ldots, U_k. \neg \bigwedge_{\rho \in \mathbb{P}} \mathsf{mso}(\rho, \boldsymbol{t})$$

where $\mathsf{mso}(\rho, \boldsymbol{t})$ is the rule obtained from rule ρ by replacing each occurrence of a constant λ_i by t_i, each occurrence of `hit` *by $\bot$ (the falsity atom), and each occurrence of a unary predicate $\mathtt{U}_i$ by a set variable U_i. We extend* mso *to NEMODEPs of higher degree by applying it recursively to NEMODEP atoms. For a NEMODEQ $\mathfrak{Q}$, we obtain $\mathsf{mso}(\mathfrak{Q})$ by replacing every NEMODEP atom $\mathbb{P}(\boldsymbol{t})$ in $\mathfrak{Q}$ by $\mathsf{mso}(\mathbb{P}(\boldsymbol{t}))$.*

By replacing `hit` with $\bot$, the derivation of a query match becomes the derivation of an inconsistency. The formula $\mathsf{mso}(\mathbb{P}(\boldsymbol{t}))$ evaluates to true if this occurs for all possible interpretations of the predicates $\mathtt{U}_j$, expressed here by universal quantification over the set variables U_j. The interpretation of $\boldsymbol{t}$ corresponds to the flagged tuple $\boldsymbol{\lambda}$ that is to be checked. It is thus easy to see that the translation captures the semantic conditions of Definitions 1 and 6.

Theorem 8 (MSO expressibility of NEMODEQs). *For every NEMODEQ $\mathfrak{Q}[\boldsymbol{x}]$, $\mathsf{mso}(\mathfrak{Q}[\boldsymbol{x}])$ can be constructed in linear time and is equivalent to $\mathfrak{Q}[\boldsymbol{x}]$.*

Example 8. *Consider NEMODEQ $\mathfrak{Q}_3$ from Example 5. Then $\mathsf{mso}(\mathfrak{Q}_3)$ is the MSO formula*

$$\forall U_2, U_3. \neg \begin{pmatrix} \forall v.(\boxed{\forall U_1. \neg \begin{pmatrix} \forall y.(\mathit{certBy}(v,y) \rightarrow U_1(y)) \\ \wedge \forall y, z.(U_1(y) \wedge \mathit{certBy}(y,z) \rightarrow U_1(z)) \\ \wedge \quad (U_1(w) \rightarrow \bot) \end{pmatrix}} \rightarrow U_3(v)) \\ \wedge \quad \forall x.(\mathit{linkedTo}(\mathit{alice}, x) \rightarrow U_2(x)) \\ \wedge \quad \forall x, x'.(U_2(x) \wedge U_3(x) \wedge \mathit{linkedTo}(x, x') \rightarrow U_2(x')) \\ \wedge \quad (U_2(\mathit{bob}) \rightarrow \bot) \end{pmatrix}$$

with free variable w (using certBy to abbreviate certifiedBy). The framed subformula is $\mathsf{mso}(\mathbb{P}_1(v, w))$.

Expressibility of NEMODEQs (and thus also MODEQs) in MSO is a useful feature, which we will further exploit below. For the moment, we just note the direct consequence that the PSpace combined complexity of model checking in MSO directly gives us PSpace-membership of query answering for NEMODEQs and MODEQs.

The following theorem closes the gap w.r.t. the combined complexity of query answering by showing PSpace hardness for NEMODEQs by a reduction from the validity problem of quantified Boolean formulae.

Theorem 9 (Complexity of NEMODEQ answering). *The task of checking if $\boldsymbol{c}$ is an answer to a NEMODEQ $\mathfrak{Q}[\boldsymbol{x}]$ over a database D is* P*-complete in the size of D, and* PSpace*-complete in the size of D and $\mathfrak{Q}$.*

Figure 1 gives an overview of the relationships established so far, regarding both expressivity and complexity. MODEQs feature the same complexities as monadic Datalog, while providing a significant extension of expressivity. The step to NEMODEQs leads to increased combined complexity. Nevertheless, combined complexity is still lower than for Datalog and data complexity is still lower than for MSO. In addition, in the next sections, we will show that NEMODEQs (and MODEQs) are also more well-behaved than these two when it comes to checking containment or interaction with rule sets that give rise to infinite structures.

5. DECIDING QUERY CONTAINMENT

Checking query containment is an essential task in database management, facilitating query optimization, information integration and exchange, and database integrity checking. The *containment* or *subsumption problem* of two queries $\mathfrak{P}$ and $\mathfrak{Q}$ is the question whether the answers of $\mathfrak{Q}$ are contained in the answers of $\mathfrak{P}$ over any database. In this section, we show that this problem is decidable for NEMODEQs. At its core, this result is based on previous work by Courcelle [20], from which we can derive the following general theorem, which is interesting in its own right:

Theorem 10 (Deciding Datalog Containment in MSO). *There exists an algorithm that decides, given any Datalog query $\langle \mathtt{goal}, \mathbb{P} \rangle$ and any MSO query φ whose set of models is closed under homomorphisms, if the query $\langle \mathtt{goal}, \mathbb{P} \rangle$ is contained in φ.*

The underling result in [20] is formulated for queries without free variables and without constant symbols, using a variant of multi-sorted monadic second-order logic and a notion of graph grammar derived from Datalog queries. The appendix recalls the relevant notions and relates them to our setting to prove Theorem 10.

By Theorems 6, 7, and 8, NEMODEQs can be considered as Datalog queries and as MSO queries whose models are closed under homomorphisms. This shows the following:

Theorem 11 (Deciding NEMODEQ containment). *The query containment problem for NEMODEQs is decidable.*

The complexity of NEMODEQ containment remains to be determined. A lower bound is the 2ExpTime-hardness of monadic Datalog containment shown recently [7].

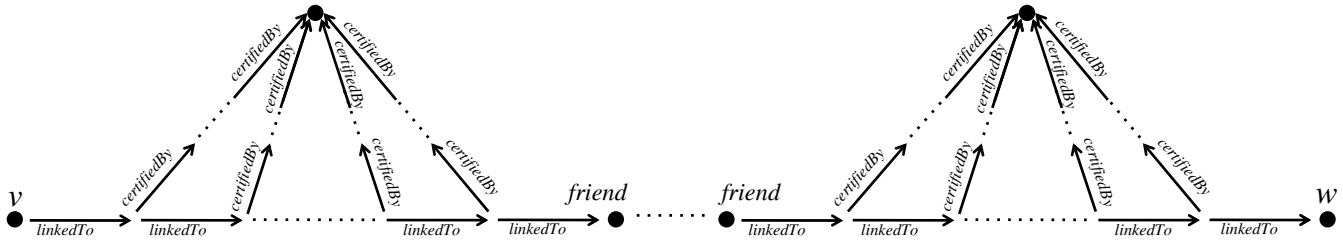

Figure 4: Structure recognized by $\mathfrak{Q}_4$ in Example 6

6. QUERYING UNDER DEPENDENCIES

Dependencies play an important role in many database applications, be it to formulate constraints, to specify views for data integration, or to define relationships in data exchange. Dependencies can be viewed as logical implications, like the TGDs introduced in Section 2, and one is generally interested in answering queries w.r.t. to their logical entailments. Universal models (Section 2) can be viewed as solutions to data exchange problems or as minimal ways of repairing constraint violations over a database. Querying under dependencies thus corresponds to finding *certain answers*, as is common, e.g., in data integration scenarios.

EXAMPLE 9. *Consider the following set of TGDs:*

$$\begin{aligned} \mathit{hasAuthor}(x,y) &\rightarrow \mathit{publication}(x) \\ \mathit{cites}(x,y) &\rightarrow \mathit{publication}(x) \wedge \mathit{publication}(y) \\ \mathit{publication}(x) &\rightarrow \exists y.\mathit{hasAuthor}(x,y) \end{aligned}$$

For a database $\{\mathit{hasAuthor}(a,c), \mathit{cites}(a,b)\}$*, the conjunctive query* $\exists z.(\mathit{hasAuthor}(x,z))$ *has* $x \mapsto a$ *as its only answer. When taking the above dependencies into account, the certain answers additionally contain* $x \mapsto b$*.*

The extension of TGDs with equality is known as *embedded dependencies*, a special case of which are *equality-generating dependencies* [1]. We will not focus on equality here, and state most of our results for TGDs.

The problem of computing CQ answers under TGDs is undecidable in general [16, 6]. A practical approach for computing certain answers is to compute a finite universal model, if possible. All combinations of TGDs and databases have universal models, but it is undecidable whether any such model is finite; the *core chase* is a known semi-decision procedure that computes a finite universal model whenever it exists [24]. Many other variants of the chase have been proposed to compute (finite) universal models in certain cases.

One can compute certain answers under TGDs even in cases where no finite universal model exists, as long as there is a universal model that is sufficiently "regular" to allow for a finite representation. One of the most general criteria is based on the well-known notion of *treewidth* (see, e.g., [4] for a formal definition). A rule set Σ is called *bounded treewidth set (bts)* if for every database D, there is a universal model $\mathcal{I}(D \cup \Sigma)$ of bounded treewidth. The treewidth bound in each case can depend on Σ and D. Recognizing whether a rule set has this property is undecidable in general [4], but many sufficient conditions have been identified. This includes the case of rule sets with finite models as a special case of models with bounded treewidth. TGDs without existential quantifiers, called *full dependencies* or Datalog rules [1], trivially have finite models. More elaborate are various notions of *acyclicity* based on analyzing the interaction of TGDs [25, 26, 37, 38, 4, 34].

Cases where $\mathcal{I}(D \cup \Sigma)$ may be infinite but treewidth-bounded are also manifold. A basic case are *guarded TGDs*, inspired by the guarded fragment of first-order logic [2]. These have been generalized to *weakly guarded TGDs* [9] and *frontier-guarded rules* [4], both of which are subsumed by *weakly frontier-guarded TGDs* [4]. The most expressive currently known bts fragments are *greedy bts TGDs* [5] and *glut-guarded TGDs* [34].

Another type of dependencies comes from the area of *Description Logics* (DLs). Originally conceived as ontology languages, DLs have also been applied to express database constraints [14]. DLs use a different syntax, but share a similar first-order semantics that makes them compatible with TGDs. Most DLs considered in database applications are Horn logics (that allow for universal models), and can thus be presented as rules [10].

EXAMPLE 10. *The set of TGDs in Example 9 can equivalently be expressed by the DL-Lite ontology*

$$\begin{aligned} \exists \mathit{hasAuthor} &\sqsubseteq \mathit{publication} & \exists \mathit{cites} &\sqsubseteq \mathit{publication} \\ \mathit{publication} &\sqsubseteq \exists \mathit{hasAuthor} & \exists \mathit{cites}^- &\sqsubseteq \mathit{publication}. \end{aligned}$$

Many DLs enjoy tree-model properties, but some expressive DLs are not bounded treewidth. This mainly applies to DLs that support transitivity or its generalizations [35, 40].

As mentioned before, the entailment problem $D, \Sigma \models Q$ is known to be decidable if Σ is bts and Q is a conjunctive query. Our main result of this section is that this extends to NEMODEQs provided that the following holds.

CONJECTURE 12. *Satisfiability of monadic second-order logic on countable interpretations of bounded treewidth is decidable.*

This statement is often taken for granted and proof sketches have been communicated. A similar result was shown in [19] for a different notion of *width*. A modern account of the relevant proof techniques that uses our notion of treewidth is given in [22] for finite graphs. Formulating the proof of [19] in these terms, one could show Conjecture 12 [21]. We cautiously characterize this statement as a conjecture since no full proof has been published.

THEOREM 13 (NEMODEQ ANSWERING UNDER BTS). *Let D be a database and let Σ be a set of rules for which the treewidth of $\mathcal{I}(D \cup \Sigma)$ is bounded. Let $\mathfrak{Q}[\boldsymbol{x}]$ be a NEMODEQ.*

1. *$D \cup \Sigma \models \mathfrak{Q}[\boldsymbol{c}/\boldsymbol{x}]$ if and only if $D \cup \Sigma \cup \{\neg\mathfrak{Q}[\boldsymbol{c}/\boldsymbol{x}]\}$ has no countable model with bounded treewidth.*
2. *If Conjecture 12 holds, then $D \cup \Sigma \models \mathfrak{Q}[\boldsymbol{c}/\boldsymbol{x}]$ is decidable.*

The proof of this theorem exploits universality of $\mathcal{I}(D \cup \Sigma)$ and preservation of NEMODEQ matches under homomorphisms (Theorem 7). The second claim follows from the first by applying Conjecture 12 and the observation that the MSO theory $D \cup \Sigma \cup \{\neg\mathsf{mso}(\mathfrak{Q}[\boldsymbol{c}/\boldsymbol{x}])\}$ is equivalent to $D \cup \Sigma \cup \{\neg\mathfrak{Q}[\boldsymbol{c}/\boldsymbol{x}]\}$ by Theorem 8.

7. MODEQ REWRITABILITY

Query rewriting is an important technique for answering queries under dependencies, and the main alternative to the chase. The general idea is to find "substitute queries" that can be evaluated directly over the database, without taking TGDs into account, and

yet deliver the same answer. We now extend this approach to MODEQs and NEMODEQs, and we establish basic cases where Datalog queries can be rewritten to MODEQs, which we extend further in Section 8.

The most common notion is *first-order rewritability*, where a conjunctive query and a set of TGDs is rewritten into a first-order query – typically a union of conjunctive queries [1]. Importantly, the rewriting does not depend on the underlying database but only on the inital query and TGDs.

Example 11. *Given the rule set from Example 9 and the conjunctive query* $\exists v, w.(hasAuthor(v, w))$*, an appropriate first-order rewriting is*

$$\exists v, w.(hasAuthor(v, w)) \vee \exists v, w.(cites(v, w)) \vee \exists v.(publication(v)).$$

First-order rewritability is a desirable property as there are efficient implementations for evaluating first-order queries (that is, SQL). First-order rewritable sets of TGDs are also called *finite unification sets* [4]. It is undecidable whether a set of TGDs belongs to this class, but an iterative backward chaining algorithm can be defined that terminates on FO-rewritable rule sets and provides the rewritten FO formula [4]. Known sufficient conditions for FO-rewritability led to the definition of *atomic-hypothesis rules* and *domain restricted rules* [4], *linear Datalog+/–* [10], as well as *sticky sets of TGDs* and *sticky-join sets of TGDs* [11, 12]. These criteria were recently found to be subsumed by an efficiently checkable condition that gives rise to the class of *weakly recursive TGDs* [18]. Important FO-rewritable description logics include the DL-Lite family of logics [14]. However, many useful TGDs can not be expressed as first-order queries.

Example 12. *First-order logic cannot express transitive closure, so there is no first-order rewriting for the CQ* $s(v) \wedge r(v, w) \wedge s(w)$ *under the TGD* $r(x, y) \wedge r(y, z) \rightarrow r(x, z)$*.*

This motivates the consideration of more expressive query languages in query rewriting.

Example 13. *The TGD and query of Example 12 can be rewritten as a Datalog query*

$$r(x, y) \rightarrow r_{\text{IDB}}(x, y) \tag{28}$$
$$r_{\text{IDB}}(x, y) \wedge r_{\text{IDB}}(y, z) \rightarrow r_{\text{IDB}}(x, z) \tag{29}$$
$$s(v) \wedge r_{\text{IDB}}(v, w) \wedge s(w) \rightarrow \texttt{goal}, \tag{30}$$

but also as a conjunctive regular path query

$$\exists v, w.s(v) \wedge r^*(v, w) \wedge s(w).$$

Rewriting CQs under TGDs into Datalog queries is interesting, but the evaluation of Datalog queries remains complex. The undecidability of Datalog query containment also makes it intrinsically difficult to determine if such a query captures a set of TGDs. The use of C2RPQs is more interesting. Yet, to the best of our knowledge, rewriting of conjunctive queries into C2RPQs has been addressed only very implicitly by now [40]. Moreover, as discussed in Section 3.1, C2RPQs are still rather constrained: besides further structural restrictions, they only allow for recursion over binary predicates. We therefore consider query rewritability under MODEQs and NEMODEQs, defined as follows:

Definition 9 (NEMODEQ rewritability). *Let* Σ *be a set of TGDs and let* $Q[\boldsymbol{x}]$ *be a CQ. A (NE)MODEQ* $\mathfrak{Q}_{Q,\Sigma}$ *is a* rewriting *of* Q *under* Σ *if, for all databases* D *and potential query answers* $\boldsymbol{c}$*, we have* $D \cup \Sigma \models Q[\boldsymbol{c}/\boldsymbol{x}]$ *iff* $D \models \mathfrak{Q}_{Q,\Sigma}[\boldsymbol{c}/\boldsymbol{x}]$*.* Σ *is called* (NE)MODEQ-rewritable *if every conjunctive query* Q *has a (NE)MODEQ rewriting for* Σ *and* Q*.*

Rewritability of conjunctive queries entails rewritability of MODEQs, so the conditions of Definition 9 hold even when considering MODEQs instead of CQs. This is shown by replacing CQs in rule bodies with MODEQs, where care must be taken that the existentially quantified variables in the CQ are not used anywhere else in the rule body:

Lemma 14 (Replacement Lemma). *Consider a set* Σ *of TGDs, a conjunctive query* $Q = \exists \boldsymbol{y}.\psi[\boldsymbol{x}, \boldsymbol{y}]$*, and a NEMODEQ* $\mathfrak{Q}[\boldsymbol{x}]$ *that is a rewriting for* Σ *and* Q*. Then* Q *and* $\mathfrak{Q}$ *are equivalent in all models of* Σ*, i.e.,* $\Sigma \models \forall \boldsymbol{x}.Q[\boldsymbol{x}] \leftrightarrow \mathfrak{Q}[\boldsymbol{x}]$*.*

Let $\psi[\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{y}'/\boldsymbol{y}]$ *be the conjunction of* Q *with variables* $\boldsymbol{x}$ *replaced by terms* $\boldsymbol{t}$ *and variables* $\boldsymbol{y}$ *replaced by variables* $\boldsymbol{y}'$*. We say that* $\psi[\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{y}'/\boldsymbol{y}]$ *is a* match *in a Datalog rule* ρ *if* ρ *is of the form* $\psi[\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{y}'/\boldsymbol{y}] \wedge \varphi \rightarrow \chi$ *where the* $\boldsymbol{y}'$ *occur neither in* φ *nor in* χ*.*

Given some NEMODEQ $\mathfrak{P}[\boldsymbol{z}]$ *over* Σ*, let* $\mathfrak{P}'[\boldsymbol{z}]$ *denote a NEMODEQ obtained by replacing a match* $\psi[\boldsymbol{t}/\boldsymbol{x}, \boldsymbol{y}'/\boldsymbol{y}]$ *of* Q *in some rule of* $\mathfrak{P}$ *by* $\mathfrak{Q}[\boldsymbol{t}/\boldsymbol{x}]$*, where the bound variables in* $\mathfrak{Q}$ *do not occur in* $\mathfrak{P}$*. Then* $\mathfrak{P}$ *and* $\mathfrak{P}'$ *are equivalent in all models of* Σ*, i.e.,* $\Sigma \models \forall \boldsymbol{z}.\mathfrak{P}[\boldsymbol{z}] \leftrightarrow \mathfrak{P}'[\boldsymbol{z}]$*.*

All known FO-rewritable classes are rewritten to unions of CQs. Since every union of CQs can be expressed as MODEQ, all such rule sets are also MODEQ-rewritable. Moreover, Section 3.2 implies that every set of monadic Datalog rules is MODEQ-rewritable. Since MODEQs are strictly more expressive than monadic Datalog queries, one would expect to find larger classes of MODEQ-rewritable TGDs. An appropriate generalization of monadic Datalog is as follows:

Definition 10 (j-oriented rule set). *We call a set of Datalog rules* Σ j-oriented *for the integer* j *if all head predicates have the same arity* n*, and* $1 \leq j \leq n$*, and we have: if a rule's body contains an atom* $p(\boldsymbol{t})$ *for some head predicate* p *and the rule's head contains an atom* $q(\boldsymbol{t}')$*, then* $\boldsymbol{t}$ *and* $\boldsymbol{t}'$ *agree on all positions other than possibly* j*.*

Intuitively speaking, recursive derivations in j-oriented rule sets can only modify the content of a single position j while keeping all other arguments fixed in all derived facts.

Example 14. *The following rule set* Σ_{family} *is* 3*-oriented. We use atoms* $parentsSon(x, y, z)$ *and* $parentsDghtr(x, y, z)$ *to denote that* z *is the son and daughter of* x *and* y*, respectively.*

$$parentsSon(x, y, z) \wedge hasBrother(z, z') \rightarrow parentsSon(x, y, z')$$
$$parentsSon(x, y, z) \wedge hasSister(z, z') \rightarrow parentsDghtr(x, y, z')$$
$$parentsDghtr(x, y, z) \wedge hasBrother(z, z') \rightarrow parentsSon(x, y, z')$$
$$parentsDghtr(x, y, z) \wedge hasSister(z, z') \rightarrow parentsDghtr(x, y, z')$$

Definition 11 (Single predicate rewriting). *For a* j*-oriented set* Σ *of Datalog rules and a head predicate* p *of* Σ*, a MODEP* $\mathbb{P}_{p,\Sigma}$ *is defined as follows. Let* $\mathtt{U}_q$ *be an auxiliary unary predicate for each head predicate* q *in* Σ*, let* $\mathtt{V}_i$ *be a auxiliary unary predicate for each* $i \in \{1, \ldots, \mathsf{ar}(p)\}$ *with* $i \neq j$*, and let* $\tilde{z}_j$ *be an additional variable not occurring in* Σ*. Then* $\mathbb{P}_{p,\Sigma}$ *contains the following rules:*

- *a rule* $\mathtt{U}_p(\lambda_j) \rightarrow \mathtt{hit}$*;*
- *for each set variable* $\mathtt{V}_i$*, a rule* $\rightarrow \mathtt{V}_i(\lambda_i)$ *with empty body;*
- *for each* $\psi \rightarrow q(t_1, \ldots, t_n) \in \Sigma$*, a rule* $\psi' \rightarrow \mathtt{U}_q(t_j)$ *where* ψ' *is obtained from* ψ *by replacing each atom* $q'(t_1, \ldots, t'_j, \ldots, t_n)$ *(with* q' *a head predicate) by* $\mathtt{U}_{q'}(t'_j)$*, and by adding for each term* t_i *with* $i \neq j$ *a new body atom* $\mathtt{V}_i(t_i)$*;*
- *a rule* $q'(\lambda_1, \ldots, \tilde{z}_j, \ldots, \lambda_n) \rightarrow \mathtt{U}_{q'}(\tilde{z}_j)$ *for each head predicate* q'*.*

For a list $\boldsymbol{z}$ *of* $\mathsf{ar}(p)$ *variables, the MODEQ* $\mathfrak{Q}_{p,\Sigma}[\boldsymbol{z}]$ *is defined as* $\mathbb{P}_{p,\Sigma}(\boldsymbol{z})$*.*

This operation allows us to express the extension of a predicate p by means of a MODEQ.

THEOREM 15 (SINGLE PREDICATE REWRITING CORRECTNESS). *If Σ is j-oriented and p is a head predicate, then $\mathfrak{Q}_{p,\Sigma}[\boldsymbol{z}]$ is a rewriting for Σ and $p(\boldsymbol{z})$.*

EXAMPLE 15. *For the rule set in Example 14, we obtain the rewriting $\mathfrak{Q}_{parentsSon,\Sigma}[z_1,z_2,z_3] = \mathbb{P}_{parentsSon,\Sigma}(z_1,z_2,z_3)$ with rules*

$$\begin{aligned}
\mathtt{U}_{parentsSon}(\lambda_3) &\to \mathtt{hit} \qquad \to \mathtt{V}_1(\lambda_1) \qquad \to \mathtt{V}_2(\lambda_2)\\
\mathtt{V}_1(x)\wedge\mathtt{V}_2(y)\wedge\mathtt{U}_{parentsSon}(z)\wedge hasBrother(z,z') &\to \mathtt{U}_{parentsSon}(z')\\
\mathtt{V}_1(x)\wedge\mathtt{V}_2(y)\wedge\mathtt{U}_{parentsSon}(z)\wedge hasSister(z,z') &\to \mathtt{U}_{parentsDghtr}(z')\\
\mathtt{V}_1(x)\wedge\mathtt{V}_2(y)\wedge\mathtt{U}_{parentsDghtr}(z)\wedge hasBrother(z,z') &\to \mathtt{U}_{parentsSon}(z')\\
\mathtt{V}_1(x)\wedge\mathtt{V}_2(y)\wedge\mathtt{U}_{parentsDghtr}(z)\wedge hasSister(z,z') &\to \mathtt{U}_{parentsDghtr}(z')\\
parentsSon(\lambda_1,\lambda_2,\tilde{z}_3) &\to \mathtt{U}_{parentsSon}(\tilde{z}_3)\\
parentsDghtr(\lambda_1,\lambda_2,\tilde{z}_3) &\to \mathtt{U}_{parentsDghtr}(\tilde{z}_3).
\end{aligned}$$

The $\mathtt{V}$ predicates are not really needed here, since rule bodies do not impose any conditions on the respective variables. If no constants from $\mathbf{C}$ occur, one could always replace $\mathtt{V}$ with the respective λs, but expressions like $\mathtt{V}_1(c)$ would require an equality predicate to state $\lambda_1 \approx c$. For the semantics of NEMODEQs to be meaningful, constants λ_i must always be allowed to be equal to other constants, even if a unique name assumption is adopted for constants in $\mathbf{C}$.

Using the Replacement Lemma 14, we can extend Theorem 15 to arbitrary conjunctive queries:

THEOREM 16 (j-ORIENTEDNESS IMPLIES REWRITABILITY). *Every j-oriented rule set is MODEQ-rewritable.*

8. REWRITING LAYERS OF TGDS

In the previous section, we have identified a first criterion for MODEQ-rewritability, and thus decidability of query entailment. However, there are many cases where only some of the given TGDs are rewritable. On the other hand, our results from Section 6 guarantee that NEMODEQ answering is still decidable in the presence of TGDs that are in bts, based on techniques that do not require rewriting. We now show how to combine both results by applying query rewriting to a subset of TGDs that is suitably "layered above" the remaining TGDs. This allows us to define a class of *fully oriented rule sets* that generalizes j-oriented rule sets to cover the full expressiveness of NEMODEQs. More generally, the combination of bts and NEMODEQ-rewritability captures some of the most expressive ontology languages for which query answering is known to be decidable.

Given a set of TGDs, we first clarify which subsets of TGDs can be rewritten into queries that can be evaluated over the remaining TGDs and databases without loosing results. To this end, we consider a notion of *rule dependency*. Related notions were first described in [3], and independently in [24]. Our presentation is closely related to [4].

DEFINITION 12 (RULE DEPENDENCY, CUT). *Let $\rho_1 = B_1 \to H_1$ and $\rho_2 = B_2 \to H_2$ be two TGDs. We say that ρ_2* depends on *ρ_1, written $\rho_1 \prec \rho_2$, if there is*

- *a database D,*
- *a substitution θ of all variables in B_1 with terms in D such that $\theta(B_1) \subseteq D$, and*
- *a substitution θ' of all variables in B_2 with terms in $D \cup \theta(H_1)$ such that $\theta'(B_2) \subseteq D \cup \theta(H_1)$ but $\theta'(B_2) \not\subseteq D$.*

We say that ρ_2 strongly depends on *ρ_1, written $\rho_1 \prec\!\prec \rho_2$, if H_1 contains a predicate that occurs in B_2.*

A (strong) cut *of a set of rules Σ is a partition $\Sigma_1 \cup \Sigma_2$ of Σ such that no rule in Σ_1 (strongly) depends on a rule in Σ_2. It is denoted $\Sigma_1 \triangleright \Sigma_2$ ($\Sigma_1 \triangleright\!\!\triangleright \Sigma_2$).*

The notion of rule dependencies encodes which rule can possibly trigger which other rule. Checking if a rule depends on another is an NP-complete task [4]. We thus introduce the simpler notion of strong dependency that can be checked in polynomial time. Clearly, dependency implies strong dependency, but the converse might not be true.

EXAMPLE 16. *Consider the following Datalog rules:*

$$A(x) \wedge B(x) \to C(x) \tag{31}$$
$$C(x) \to \exists v.p(x,v), A(v) \tag{32}$$

Rule (32) *strongly depends on* (31) *and vice versa. Moreover,* (32) *depends on* (31)*, where the database of Definition 12 could be $D = \{A(c), B(c)\}$ using $\theta = \theta' = \{x \mapsto c\}$. However,* (31) *does not depend on* (32)*: a substitution θ' can map x to v (introduced as a new term when applying* (32)*), but the required fact $B(v)$ is not derived by* (32) *and cannot be in any initial database D (since it is not ground).*

Intuitively speaking, we can evaluate a TGD set Σ of the form $\Sigma_1 \triangleright \Sigma_2$ by first applying the rules in Σ_1, and then applying the rules of Σ_2. This is the essence of the following theorem, shown in [4].

THEOREM 17 (BAGET ET AL.). *Let Σ be a set of rules admitting a cut $\Sigma_1 \triangleright \Sigma_2$. Then, for every database D and every conjunctive query $Q[\boldsymbol{x}]$ we have that $D \cup \Sigma \models Q[\boldsymbol{x}/\boldsymbol{c}]$ exactly if there is a Boolean conjunctive query Q' such that $D \cup \Sigma_1 \models Q'$ and $Q', \Sigma_2 \models Q[\boldsymbol{x}/\boldsymbol{c}]$.*

We can thus rewrite queries in "layers" based on cuts:

LEMMA 18 (QUERY REWRITING WITH CUTS). *Let D be a database and let $\Sigma_1 \triangleright \Sigma_2$ be two sets of TGDs.*

1. *If $\mathfrak{Q}_{Q,\Sigma_2}[\boldsymbol{x}]$ is a NEMODEQ-rewriting of a conjunctive query $Q[\boldsymbol{x}]$, then:*

   $$D \cup \Sigma_1 \cup \Sigma_2 \models Q[\boldsymbol{c}/\boldsymbol{x}] \text{ if and only if } D \cup \Sigma_1 \models \mathfrak{Q}_{Q,\Sigma_2}[\boldsymbol{c}/\boldsymbol{x}].$$

2. *If Σ_1 and Σ_2 are NEMODEQ-rewritable, then so is $\Sigma_1 \cup \Sigma_2$.*

This observation has two useful implications. The second item outlines an approach of extending and combining rewriting procedures that we will elaborate on in the remainder of this section. The first item hints at a very general approach for constructing TGD languages for which query answering is decidable, as expressed in the next theorem.

THEOREM 19 (QUERY ANSWERING WITH CUTS). *Consider rule sets $\Sigma_1 \triangleright \Sigma_2$, such that NEMODEQ answering is decidable under Σ_1, and NEMODEQ rewriting is decidable under Σ_2. Then NEMODEQ answering is decidable under $\Sigma_1 \cup \Sigma_2$.*

Using Theorem 13, this result specifically applies in cases where Σ_1 is a bounded treewidth set. We have noted that there are a number of effectively checkable criteria for this general class of TGDs. Much less is known about NEMODEQ rewritability beyond rewritability to unions of CQs. For a more general criterion, we can extend j-oriented rules along the lines of Lemma 18 (2).

DEFINITION 13 (FULLY ORIENTED RULE SET). *Let $\approx_{\prec\!\prec}$ be the reflexive symmetric transitive closure of $\prec\!\prec$. The set Σ is* fully oriented *if, for every $\rho \in \Sigma$, the equivalence class $[\rho]_{\prec\!\prec} = \{\rho' \in \Sigma \mid \rho \approx_{\prec\!\prec} \rho'\}$ is j-oriented (not necessarily for the same j and predicate arity).*

Given a fully oriented rule set, we can construct NEMODEQ rewritings for individual classes $[\rho]_{\ll}$ as in Definition 11, and combine these rewritings using Lemma 14:

THEOREM 20 (FULLY ORIENTED RULE SETS ARE REWRITABLE). *For a rule set Σ, it can be detected in polynomial time if Σ is fully oriented. Every fully oriented set Σ is MODEQ-rewritable.*

The use of $\ll$ instead of $\prec$ is relevant for deciding full orientedness in polynomial time. Even with this restriction, fully oriented Datalog queries have the same expressivity as NEMODEQs. Moreover, every NEMODEQ-rewritable TGD set can be expressed as a set of rules that can be transformed into a MODEQ using Theorem 20.

THEOREM 21 (NEMODEQS = FULLY ORIENTED DATALOG). *For every NEMODEQ $\mathfrak{Q}$, the rule set* datalog($\mathfrak{Q}$) *of Definition 7 is fully oriented. Moreover, for every NEMODEQ-rewritable set Σ of TGDs, there is a fully oriented set of Datalog rules Σ' such that:*

- *every predicate p in Σ has a corresponding head predicate q_p in Σ' that does not occur in Σ,*
- *for every database D and conjunctive query $Q[\boldsymbol{x}]$ that do not contain predicates of the form q_p, and for every list of constants $\boldsymbol{c}$, $D \cup \Sigma \models Q[\boldsymbol{c}/\boldsymbol{x}]$ iff $D \cup \Sigma' \models Q'[\boldsymbol{c}/\boldsymbol{x}]$ where Q' is obtained from Q by replacing all predicates p by q_p.*

Another interesting criterion for NEMODEQ rewritability has been studied for Description Logics (DLs), where all predicates are of arity one or two. Expressive DL ontologies consist of two kinds of terminological axioms: concept inclusion axioms and role inclusion axioms. A role inclusion axiom is a Datalog rule of the form $R_1(x_0, x_1) \wedge \ldots \wedge R_n(x_{n-1}, x_n) \rightarrow R(x_0, x_n)$, which can be viewed as a generalized transitivity statement. Even in relatively inexpressive DLs, role inclusion axioms lead to undecidability of CQ answering [35]. To overcome this, syntactic restrictions are imposed on role inclusions, to ensure that all role inclusions can be captured using finite automata [33]. This is equivalent to rewriting role inclusions to regular path queries, and has indeed been exploited to decide CQ answering over expressive DLs [40]. This can be viewed as an implicit application of Theorem 19.[2] Many reasoning procedures for DLs are based on tree-like models, so various approaches to DL CQ answering can indeed be viewed as special combinations of bounded treewidth sets and NEMODEQ rewritable sets of TGDs [35, 40]. This supports the relevance of the general relationships observed here, and it motivates the further study of criteria for NEMODEQ rewritability. The rewriting methods used in DLs are based on regular languages, so it seems promising to consider their generalizations to graph grammars when dealing with arbitrary TGDs [22].

9. CONCLUSION

Monadically defined queries and their nested extension achieve a balance between expressivity and computability. They capture and significantly extend the query capabilities of (unions of) conjunctive queries as well as (unions of) conjunctive two-way regular path queries and monadic Datalog queries, the prevailing querying paradigms for structured and semi-structured databases. At the same time, they are conveniently expressible both in Datalog and monadic second-order logic. Yet, as opposed to these two, they ensure decidability of query containment and of query answering in the presence of depdendencies that allow for universal models with bounded treewidth – a property shared by many of the known decidable TGD classes.

[2]DL concept and role inclusion axioms are not always separated by a cut. There are well known rewriting methods to achieve this [32].

The novel notions of MODEQ-rewritability and NEMODEQ-rewritability significantly extend first-order rewritability, which has been proven useful for theoretical considerations and practical realization of query answering alike. This extension allows for capturing much larger classes of TGD sets covering features like transitivity, which are considered difficult to handle within the known decision frameworks. Moreover, (NE)MODEQ-rewritable TGD sets can smoothly be integrated with bounded treewidth TGD sets as long as certain dependency constraints are obeyed. This provides a valuable perspective on rule-based data access as a task that can be solved by combining bottom-up techniques like the *chase* with top-down techniques like query rewriting.

Our work raises a number of interesting questions for future research: How general are (NE)MODEQs? Are there larger, more expressive fragments which jointly satisfy all the established properties? Is every rewriting for a TGD set and a given CQ that is expressible in MSO logic equivalent to a NEMODEQ? What is the precise complexity of deciding query containment? Which more general syntactic criteria ensure NEMODEQ-rewritability? Can all fragments of TGDs (including those considered in description logics) for which conjunctive query answering is known to be decidable be captured as a combination of bts and NEMODEQ-rewriting? Answering these questions will not only contribute to our understanding of NEMODEQs, but also provide a more unified view on query answering under dependencies in general.

Acknowledgments. We acknowledge contributions of various people: Bruno Courcelle and Detlef Seese gave very helpful comments on their work on MSO and structures of bounded treewidth; Diego Calvanese helped clarifying complexities of C2RPQs; Pierre Bourhis provided feedback leading to our formulation of Theorem 10; various anonymous reviewers gave valuable input on earlier versions of this work.

This work was supported by the Royal Society, the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, 'Optique', and the EPSRC projects ExODA, Score! and MaSI3.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1994.

[2] H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, 1998.

[3] J.-F. Baget. Improving the forward chaining algorithm for conceptual graphs rules. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *KR*, pages 407–414. AAAI Press, 2004.

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

[5] J.-F. Baget, M.-L. Mugnier, S. Rudolph, and M. Thomazo. Walking the complexity lines for generalized guarded existential rules. In Walsh [47], pages 712–717.

[6] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 73–85. Springer, 1981.

[7] M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, editors, *ICALP (2)*, volume 7392 of *LNCS*, pages 79–91. Springer, 2012.

[8] G. Brewka and J. Lang, editors. *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*. AAAI Press, 2008.

[9] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In Brewka and Lang [8], pages 70–80.

[10] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In Paredaens and Su [41], pages 77–86.

[11] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *Proceedings of VLDB 2010*, 3(1):554–565, 2010.

[12] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog+/-. In P. Hitzler and T. Lukasiewicz, editors, *Web Reasoning and Rule Systems*, volume 6333 of *LNCS*, pages 1–17. Springer, 2010.

[13] D. Calvanese. Personal communication, September 2011.

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

[15] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[16] A. K. Chandra, H. R. Lewis, and J. A. Makowsky. Embedded implicational dependencies and their inference problem. In *Conference Proceedings of the 13th Annual ACM Symposium on Theory of Computation (STOC'81)*, pages 342–354. ACM, 1981.

[17] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In J. E. Hopcroft, E. P. Friedman, and M. A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing (STOC'77)*, pages 77–90. ACM, 1977.

[18] C. Civili and R. Rosati. A broad class of first-order rewritable tuple-generating dependencies. In P. Barceló and R. Pichler, editors, *Proceedings of the 2nd Workshop on the Resurgence of Datalog in Academia and Industry (Datalog 2.0, 2012)*, volume 7494 of *LNCS*. Springer, 2012.

[19] B. Courcelle. The monadic second-order logic of graphs, ii: Infinite graphs of bounded width. *Mathematical Systems Theory*, 21(4):187–221, 1989.

[20] B. Courcelle. Recursive queries and context-free graph grammars. *Theoretical Computer Science*, 78(1):217–244, 1991.

[21] B. Courcelle. Personal communication, August 2011.

[22] B. Courcelle and J. Engelfriet. Graph structure and monadic second-order logic, a language theoretic approach. manuscript, to be published at Cambridge University Press; available at http://www.labri.fr/perso/courcell/Book/TheBook.pdf, April 2011.

[23] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[24] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In M. Lenzerini and D. Lembo, editors, *Proc. 27th Symposium on Principles of Database Systems (PODS'08)*, pages 149–158. ACM, 2008.

[25] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In D. Calvanese, M. Lenzerini, and R. Motwani, editors, *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*, volume 2572 of *LNCS*, pages 225–241. Springer, 2003.

[26] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

[27] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the seventeenth ACM symposium on Principles of database systems*, PODS '98, pages 139–148. ACM, 1998.

[28] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[29] G. Gottlob and C. H. Papadimitriou. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1):104–122, 2003.

[30] S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *Proceedings of VLDB 2010*, 3(1):93–104, 2010.

[31] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[32] Y. Kazakov. $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In Brewka and Lang [8], pages 274–284.

[33] Y. Kazakov. An extension of complex role inclusion axioms in the description logic $\mathcal{SROIQ}$. In *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR 2010)*, LNCS. Springer, 2010.

[34] M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In Walsh [47], pages 963–968.

[35] M. Krötzsch, S. Rudolph, and P. Hitzler. Conjunctive queries for a tractable fragment of OWL 1.1. In K. Aberer et al., editor, *Proceedings of the 6th International Semantic Web Conference (ISWC'07)*, volume 4825 of *LNCS*, pages 310–323. Springer, 2007.

[36] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[37] B. Marnette. Generalized schema-mappings: from termination to tractability. In Paredaens and Su [41], pages 13–22.

[38] M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *Proceedings of VLDB 2009*, 2(1):970–981, 2009.

[39] M.-L. Mugnier. Ontological query answering with existential rules. In S. Rudolph and C. Gutierrez, editors, *Web Reasoning and Rule Systems (RR 2011)*, volume 6902 of *LNCS*, pages 2–23. Springer, 2011.

[40] M. Ortiz, S. Rudolph, and M. Simkus. Query answering in the Horn fragments of the description logics $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. In Walsh [47], pages 1039–1044.

[41] J. Paredaens and J. Su, editors. *Proc. 28th Symposium on Principles of Database Systems (PODS'09)*. ACM, 2009.

[42] S. Rudolph and M. Krötzsch. Flag & check – data-tractable expressive queries for intelligent databases (extended technical report). Technical Report 3030, Institute AIFB, Karlsruhe Institute of Technology, 2012. http://www.aifb.kit.edu/web/Techreport3030.

[43] O. Shmueli. Equivalence of DATALOG queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.

[44] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

[45] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[46] M. Y. Vardi. The complexity of relational query languages. In H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, editors, *STOC*, pages 137–146. ACM, 1982.

[47] T. Walsh, editor. *Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11)*. AAAI Press/IJCAI, 2011.