Flag & Check: Data Access with Monadically Defined Queries (Extended Technical Report)

Technical Report 3030, Institute AIFB, Karlsruhe Institute of Technology

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 [40]. 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 [45], but remains $AC_0$ for data [32]. 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, 44]. 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) [32, 46].

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/NSpace data) [29]. Still, query containment remains undecidable. Widely known 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/NSpace 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 both monadic Datalog and C2RPQs are Datalog and monadic second-order logic (MSO) [37]. Query containment is decidable for neither of these. These languages are incomparable, even regarding query complexities (MSO has $PTime$ combined/PH data [45, 47, 37]), 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.

---

$^1$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.
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.
- 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 formulæ.
- 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. They are related to the presence of TGDs [24]. As MODEQs 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 undesirable, but can be approximated by various notions of acyclicity [25, 26, 24, 39, 30, 35].
- 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, 35, 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 Appendix.

2. PRELIMINARIES

We consider a standard language of first-order predicate logic, based on an infinite set C of constant symbols, an infinite set P of predicate symbols, and an infinite set V of first-order variables. Each predicate p ∈ P is associated with a natural number ar(p) called the arity of p. The list of predicates and constants forms the language’s signature J = (P, C). We generally assume J = (P, C) to be fixed, and only refer to it explicitly if needed.

Databases, Rules, and Queries. A term is a variable x ∈ V or a constant c ∈ 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 t, x, c denote finite lists of such entities. We use the standard predicate logic definitions of atom and formula, using symbols ψ, ϕ 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 ϕ[x] to emphasize that a formula ϕ has free variables x; we write ϕ{c/x} for the formula obtained from ϕ by replacing each variable in x by the respective constant in c (both lists must have the same length). A formula without free variables is a sentence.

A conjunctive query (CQ) is a formula Q[x] = ∃y ψ[x, y] where ψ[x, y] is a conjunction of atoms. A tuple-generating dependency (TGD) is a formula of the form ∀x, y, ϕ[x, y] → ∃z ϕ[x, 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 Π, Ρ, S.

We use the standard semantics of first-order logic (FOL). An interpretation I consists of (possibly infinite) sets Δt of called domain and a function - that maps constants c to domain elements ct ∈ Δt and predicate symbols p to relations pt ⊆ (Δt)ar(p), thereby pt is called the extension of p. A variable assignment for I is a function Z : V → Δt. Conditions for I and Z to satisfy a FOL formula ϕ (i.e., to be a model of ϕ, written I, Z ⊨ ϕ) are defined as usual. If ϕ is a sentence, then Z is irrelevant for satisfaction and can be omitted. An answer to a CQ Q[x] over a database D and set Σ of TGDs is a list of constants c for which D ⊔ Σ ⊨ Q[c/x].

Given an interpretation I and a formula ϕ[x] with free variables x = ⟨x1, ..., xn⟩, the extension of ϕ[x] is the subset of (Δt)n containing all those tuples ⟨δ1, ..., δn⟩ for which I, [x1/δ1, ..., xn/δn] ⊨ ϕ[x]. Two formulae ϕ[x] and ψ[x] with the same free variables x are called equivalent if their extensions coincide for every interpretation I.

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

Finding query answers is facilitated in practice since one may
focus on universal models. A universal model of a set of sentences
\(\Psi\) is an interpretation \(I\) such that (i) \(I \models \Psi\), and (ii) for every
interpretation \(\mathcal{J}\) with \(\mathcal{J} \models \Psi\), there is a homomorphism from \(I\) 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
\(I(D \cup \Sigma)\) denote the universal model, for which every homomor-
phism into any other model of \(D \cup \Sigma\) is injective, \(I(D \cup \Sigma)\) 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 \(Q[\lambda]\) is a query for
which the set of models of \(\exists \lambda Q[\lambda]\) is closed under homomorphisms,
then, for every database \(D\) and set \(\Sigma\) of TGDs, \(D \cup \Sigma \models Q[\lambda]\)
if and only if either \(D \cup \Sigma\) is inconsistent or \(I(D \cup \Sigma) \models Q[\lambda]\).

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 \(I(D)\) 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 \(\delta = (\delta_1, \ldots, \delta_n) \in (\Delta^m)^n\)
is in the predicate’s extension or not. To this end, the candidate tuple
is first “flagged” by associating each \(\delta_i\) with an auxiliary constant
name \(\lambda_i\) that represents this element. The “check” is performed
by running a Datalog program with this fixed interpretation of the
constants \(\lambda_i\). The check succeeds if a special fact \(hit\) is derived.

**EXAMPLE 1.** To illustrate the idea, we consider a typical trans-
itive 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:

\[
\text{certifiedBy}(x, y) \rightarrow \text{certChain}(x, y) \quad (1)
\]

\[
\text{certChain}(x, y) \land \text{certifiedBy}(y, z) \rightarrow \text{certChain}(x, z) \quad (2)
\]

Corresponding Datalog rules \(P_1\) for “flag & check” are:

\[
\text{certifiedBy}(x_1, y) \rightarrow U_1(x) \quad (3)
\]

\[
U_1(x) \land \text{certifiedBy}(y, z) \rightarrow U_1(z) \quad (4)
\]

\[
U_1(\lambda_1) \rightarrow \text{hit} \quad (5)
\]

We define \(\text{certChain}\) to contain all pairs \((\delta_1, \delta_2)\) for which \(P_1\) entails
\(hit\) when interpreting \(\lambda_1\) as \(\delta_1\) and \(\lambda_2\) as \(\delta_2\).

As in Example 1, the Datalog rules that we consider for the check-
ing phase only use \(hit\) or new, unary predicates \(U_i\) 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, re-
cursive 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
\(\mathcal{F}\), a monadically defined predicate (MODEP) of arity \(m\) is
based on a signature \(\mathcal{F}'\) that extends \(\mathcal{F}\) with \(m\) fresh constant
symbols \(\lambda_1, \ldots, \lambda_m\), a fresh nullary predicate \(hit\), and \(k \geq 0\) fresh
unary predicates \(U_1, \ldots, U_k\). A MODEP is a set \(P\) of Datalog rules
over \(\mathcal{F}'\) where only \(U_1, \ldots, U_k\) and \(hit\) occur in rule heads.

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

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

**EXAMPLE 2.** The rules (3)-(5) define a binary MODEP \(P_1\), so
the query of Example 1 can be written as a MODEQ \(Q_1[\lambda_1, \lambda_2]\) =
\(P_1(x, w)\). For another example, assume there are entities certifying
the security of message handling in certain nodes of a net-
work. We are interested in entities that can (directly) certify se-
tive 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
\(Q_2[w]\) = \(P_2(w)\), where \(P_2\) consists of the following rules:

\[
\text{certifiedBy}(x, \lambda_1) \rightarrow U_1(x) \quad (6)
\]

\[
\text{linkedTo}(alice, x) \land U_1(x) \rightarrow U_2(x) \quad (7)
\]

\[
U_2(x) \land U_1(x) \land \text{linkedTo}(x, x') \rightarrow U_2(x') \quad (8)
\]

\[
U_2(bob) \rightarrow \text{hit} \quad (9)
\]

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 \(Q_i[\lambda_i, \lambda_j]\) with \(i \in \{1, \ldots, k\}\), their union
is expressed as the MODEQ \(\{Q_i[\lambda_i, \lambda_j] \rightarrow \text{hit}\} | 1 \leq i \leq k\}) \times x\). Before
showing that MODEQs also capture more powerful query lan-
guages, we observe closure under homomorphism (see Fact 1) and
state a basic complexity result.

**THEOREM 2.** For every MODEQ \(Q[\lambda]\), the set of all models of
\(\exists \lambda Q[\lambda]\) is closed under homomorphisms.

**THEOREM 3 (MODEQ answering complexity).** Testing if \(c\)
is an answer to a MODEQ \(Q[\lambda]\) over a database \(D\) is \(P\)-complete
in the size of \(D\), and \(NP\)-complete in the combined size of \(D\) and \(\Sigma\).

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

Figure 2: Structure recognized by MODEQ \(Q_2\) in Example 2

![Diagram](image-url)
3.1 MODEQs Capture Regular Path Queries

We now show that MODEQs subsume conjunctive two-way regular path queries (C2RPs), 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.

C2RPs 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** (C2RP syntax and semantics). A two-way regular path predicate (2RPP) is a regular expression over the alphabet \( \Gamma = \{ p, p^* \mid \alpha(p) = 2 \} \) of normal and inverse binary predicate symbols. All 2RPPs are of arity 2. Consider an interpretation \( I \).

For inverse predicates \( p^* \), we define \( (p^*)^i = \{(\delta_1, \delta_2) \mid (\delta_1, \delta_2) \in p^i \} \). For a 2RPP \( P \), we set \( (\delta, \delta') \in P^i \) if there is a word \( \gamma_1 \ldots \gamma_n \) matching the regular expression \( P \), and a sequence \( \delta_0 \ldots \delta_n \) of domain elements such that \( \delta_0 = \delta = \delta_n = \delta' \), and \( (\delta_i, \delta_{i+1}) \in \gamma_i^c \) for every \( i \in \{0, \ldots, n-1\} \).

A conjunctive two-way regular path query (C2RPQ) is a conjunctive query that uses both normal predicates and 2RPPs in its \( \in \{ \). For every C2RPQ \( Q \), and a sequence \( \Gamma = \{ p \} \), we define \( \chi_{\text{C2RPQ}}(x, y) \).

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

\[
\text{mountain}(x) \land \text{continent}(y) \land \text{locatedIn}(x, y). \tag{10}
\]

Query answering for C2RPs is NP-complete regarding the size of the database and query, which is the same as for CQs. In terms of data complexity, C2RPs are NL-Complete-complete, and thus harder than CQs (AC). 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 \( A_P = (T, S, I, F) \) that recognizes \( P \). The binary MODEP \( \text{modep}(P) \) consists of the rules

\[
\begin{align*}
\text{U}(A) & \quad \text{for every initial state} \ s \in I, \\
\text{U}(z) \land p(z, z') & \quad \text{for every transition} \ (s, p, z') \in T, \\
\text{U}(z) \land p(z', z) & \quad \text{for every transition} \ (s, p', z') \in T, \\
\text{U}(A_2) & \quad \text{for every final state} \ s \in F.
\end{align*}
\]

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

The intuition behind the translation of C2RPs 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 \( U \) 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 C2RPs). For every C2RPQ \( Q \), the MODEQ \( \text{modeq}(Q) \) can be constructed in linear time, and is equivalent to \( Q \). In particular, the answers for \( Q \) and \( \text{modeq}(Q) \) coincide.

**Example 4.** Let \( Q \) be the regular path query (10). The corresponding MODEQ \( \text{modeq}(Q) \) is \( \text{mountain}(x) \land \text{continent}(y) \land \text{locatedIn}(x, y) \) where \( P = \text{modep}(\text{locatedIn}(\text{hasPart}^c)) \).

\[
\begin{align*}
\text{U}(A) & \quad \text{for every initial state} \ s \in I, \\
\text{U}(z) & \land \text{locatedIn}(z, z') \quad \text{for every transition} \ (s, p, z') \in T, \\
\text{U}(z) & \land \text{hasPart}(z', z) \quad \text{for every transition} \ (s, p', z') \in T, \\
\text{U}(A_2) & \quad \text{for every final state} \ s \in F.
\end{align*}
\]

Here we only need one “color” \( U \) that is propagated over locatedIn and inversely over hasPart. The pairs in \( P \) 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 C2RPs, 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 \( \Xi_2 \) 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 C2RPs.

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

Given an interpretation \( I \), the extension of \( (\text{goal}, \mathcal{S}) \) is the set of tuples \( \delta \) over \( \mathcal{A} \) for which every extension \( \gamma \) of \( I \) to \( \mathcal{A} \) which satisfies \( \mathcal{S} \) must also satisfy \( \delta \) \( \text{goal} \).

Note that sometimes in the literature, the arity of \( \text{goal} \) 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 \( Q = (\text{goal}, \mathcal{S}) \), we let \( \text{modeq}(Q) \) denote the MODEQ \( \Xi(x) \) where \( P \) is obtained from \( \mathcal{S} \) by

- replacing each rule \( \psi(x, y) \rightarrow \text{goal}(x) \) by \( \psi(A, y) \rightarrow \text{hit} \).
- replacing each IDB predicate, uniformly and injectively, by a predicate \( U \).

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

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 \( \mathcal{F} \) be the underlying signature. A nested monadically defined predicate (NEMODEP) of degree 1 over \( \mathcal{F} \) is a MODEP over \( \mathcal{F} \). Consider a finite set \( \mathcal{P}_i \) \((i = 1, \ldots, k)\) of NEMODEPs of degree \( d \) over \( \mathcal{F} \). A NEMODEP of degree \( d + 1 \) is a MODEP over a signature \( \mathcal{F}' \) that extends \( \mathcal{F} \) with additional predicate names \( \mathcal{P}_i \) 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 \( \lambda_i \), \( U_i \), and \( \text{hit} \) 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 \( \mathcal{P}_1 \) in \( \mathcal{C}_2 \), leading to a NEMODEP of degree 2 \( \mathcal{C}_3 = \mathcal{P}_2(w) \), where \( \mathcal{P}_2 \) coincides with \( \mathcal{P}_1 \) except that rule (6) is replaced by \( \mathcal{P}_1(x, A_1) \rightarrow U_3(x) \).

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

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

\[
\begin{align*}
certifiedBy(x, A_1) & \rightarrow U_3(x) \\
U_3(y) & \land certifiedBy(x, y) \rightarrow U_3(x) \\
linkedTo(alice, x) & \rightarrow U_3(x) \\
U_2(x) & \land \exists z \ linkedTo(x, z) \rightarrow U_2(x) \\
U_3(bob) & \rightarrow \text{hit}
\end{align*}
\]

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 \( \mathcal{P}_3 \) be the ternary MODEP that consists of the following rules:

\[
\begin{align*}
certifiedBy(x, A_1) & \rightarrow U_3(x) \\
U_3(y) & \land certifiedBy(x, y) \rightarrow U_3(x) \\
linkedTo(alice, x) & \rightarrow U_3(x) \\
U_2(x) & \land \exists z \ linkedTo(x, z) \rightarrow U_2(x) \\
U_3(bob) & \rightarrow \text{hit}
\end{align*}
\]

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 \( \mathcal{P}_6 \) be the binary NEMODEP with the following rules:

\[
\begin{align*}
U_1(y) & \land \mathcal{P}_3(x, y, z) \land \text{friend}(z) \rightarrow U_1(z) \\
U_1(y) & \land \exists z \mathcal{P}_3(y, z, A_2) \rightarrow \text{hit}
\end{align*}
\]

The NEMODEQ \( \mathcal{Q}_4 \) = \( \mathcal{P}_4(v, w) \) (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 \( \mathcal{Q}_4 \). A formal proof is given in Proposition 22 in the Appendix.

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 \( \mathcal{P} \) of arity \( m \), the set datalog(\( \mathcal{P} \)) of Datalog rules over an extended signature contains, for each rule in \( \mathcal{P} \), a new rule obtained by replacing

- each constant \( \lambda_i \) with a variable \( x_{\lambda_i} \),
- each atom \( U_i(z) \) with the atom \( \hat{U}_i(z, x_{\lambda_1}, \ldots, x_{\lambda_k}) \) where \( \hat{U}_i \) is a fresh predicate of arity \( m + 1 \),
- each atom \( \text{hit} \) with the atom \( p_{\text{hit}}(x_{\lambda_1}, \ldots, x_{\lambda_k}) \) where \( p_{\text{hit}} \) is a fresh predicate symbol of arity \( m \).

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

\[
datalog(\mathcal{P}) := datalog(\mathcal{P}') \cup \bigcup_{\mathcal{Q} \text{ a direct sub-NEMODEP of } \mathcal{P}} datalog(\mathcal{Q})\).
\]

Given a NEMODEQ \( \mathcal{Q}[x] = 3y \text{certifiedBy}[x, y] \), we define its translation datalog(\( \mathcal{Q} \)) as datalog(\( \{p_1[A, y] \rightarrow \text{hit}\} \)), where the predicate used to replace \( \text{hit} \) will be denoted by \( p_{\mathcal{Q}} \).

Note that the predicates \( \hat{U}_i \) must be globally fresh, even if multiple subqueries use the same \( U_i \). The rules in datalog(\( \mathcal{Q} \)) 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 \( \mathfrak{C} \), datalog(\( \mathfrak{C} \)) can be constructed in linear time. Moreover, the queries \( \mathfrak{C}[x] \) and \( (p, \text{datalog}(\mathfrak{C})) \) are equivalent, i.e., their answers coincide.

Example 7. The Datalog translation for the MODEQ \( \mathfrak{C}_3 \) from Example 5 is as follows:

<table>
<thead>
<tr>
<th>datalog(( \mathfrak{C}_3 ))</th>
<th>datalog(( \mathfrak{P}_1 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>certifiedBy(x_1, y) \rightarrow U_1(y, x_1, x_2)</td>
<td>p_3(x, x_1) \rightarrow \tilde{U}_3(x, x_1)</td>
</tr>
<tr>
<td>U_1(y, x_1, x_2) \land \text{certifiedBy}(y, z) \rightarrow \tilde{U}_1(z, x_1, x_2)</td>
<td>\tilde{U}_2(x, x_1) \land \text{linkedTo}(alice, x) \rightarrow \tilde{U}_2(x, x_1)</td>
</tr>
<tr>
<td>\tilde{U}_1(x_2, x_1, x_2) \rightarrow p_3(x, x_1)</td>
<td>\tilde{U}_2(bob, x_1) \rightarrow p_3(x_1)</td>
</tr>
</tbody>
</table>

Using backward-chaining, the goal \( p_C(x) \) can be expanded under the rules datalog(\( \mathfrak{C} \)) to obtain a (possibly infinite) set of CQs that do not contain auxiliary predicates \( p_C \). Thus \( \mathfrak{C} \) 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 \( \text{PTIME} \) 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 \( \mathfrak{C}[x] \), the set of models of \( \exists x. \mathfrak{C} \) 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 expansion 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 \( U \). 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 \( \lambda \), or monadic predicate \( U \), is used in at most one (sub-)predicate \( \mathfrak{P} \) of any NEMODEQ or NEMODEP we consider. This can always be achieved by renaming variables and predicates.

Definition 8 (NEMODEQ to MSO). For a MODEQ \( \mathfrak{P} \) of arity \( m \) with auxiliary unary predicates \( U_1, \ldots, U_t \), and a list of terms \( t = (t_1, \ldots, t_m) \), we define an MSO formula

\[ \text{ms}(\mathfrak{P}(t)) := \forall U_1, \ldots, U_t. \neg \bigwedge_{\rho \in \mathfrak{P}} \text{ms}(\rho, t) \]

where \( \text{ms}(\rho, t) \) is the rule obtained from rule \( \rho \) by replacing each occurrence of a constant \( \lambda \), by \( t \), each occurrence of the falsity atom \( \perp \), by \( t \), each occurrence of a unary predicate \( U_i \) by a set variable \( U_i \). We extend \( \text{ms}(\mathfrak{P}) \) to NEMODEP of higher degree by applying it recursively to NEMODEP atoms. For a NEMODEQ \( \mathfrak{C} \), we obtain \( \text{ms}(\mathfrak{C}) \) by replacing every NEMODEP atom \( \mathfrak{P}(t) \) in \( \mathfrak{C} \) by \( \text{ms}(\mathfrak{P}(t)) \).

By replacing \( \perp \) with \( \perp \), the derivation of a query match becomes the derivation of an inconsistency. The formula \( \text{ms}(\mathfrak{P}(t)) \) evaluates to true if this occurs for all possible interpretations of the predicates \( U_1 \), expressed here by universal quantification over the set variables \( U_i \). The interpretation of \( t \) corresponds to the flagged tuple \( \lambda \) 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 \( \mathfrak{C}[x] \), \( \text{ms}(\mathfrak{C}[x]) \) can be constructed in linear time and is equivalent to \( \mathfrak{C}[x] \).

Example 8. Consider NEMODEQ \( \mathfrak{C}_3 \) from Example 5. Then \( \text{ms}(\mathfrak{C}_3) \) is the MSO formula

\[ \forall v. (\forall U_1. (\forall y, z. (U_1(y) \land \text{certifiedBy}(y, z) \rightarrow U_1(z))) \rightarrow U_1(v)) \land \forall y. (\forall U_1. (\forall x. U_1(x) \land \text{linkedTo}(x, v) \rightarrow U_1(x))) \rightarrow U_2(v) \]

with free variable \( w \) (using \text{certBy} to abbreviate \text{certifiedBy})

The framed subformula is \( \text{ms}(\mathfrak{P}_1(v, w)) \).

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 \( \mathfrak{C} \) is an answer to a NEMODEQ \( \mathfrak{C}[x] \) over a database \( D \) is \( \text{P-complete} \) in the size of \( D \), and \( \text{PSpace-complete} \) in the size of \( D \) and \( \mathfrak{C} \).

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 \( \mathfrak{C} \) and \( \mathfrak{C} \) is the question whether the answers of \( \mathfrak{C} \) are contained in the answers of \( \mathfrak{C} \) 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 \( \text{goal}_1, \mathfrak{P} \) and any MSO query \( \psi \) whose set of models is closed under homomorphisms, if the query \( \text{goal}_1, \mathfrak{P} \) is contained in \( \psi \).
The underlying 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].

### 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:

\[
\begin{align*}
\text{hasAuthor}(x, y) & \rightarrow \text{publication}(x) \\
\text{cites}(x, y) & \rightarrow \text{publication}(x) \land \text{publication}(y) \\
\text{publication}(x) & \rightarrow \exists y. \text{hasAuthor}(x, y)
\end{align*}
\]

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

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 \(\Sigma\) is called bounded treewidth set (bts) if for every database \(D\), there is a universal model \(I(D \cup \Sigma)\) of bounded treewidth. The treewidth bound in each case can depend on \(\Sigma\) 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, 38, 39, 4, 35].

Cases where \(I(D \cup \Sigma)\) 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 [35].

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

\[
\begin{align*}
\exists \text{hasAuthor} & \sqsubseteq \text{publication} \\
\exists \text{cites} & \sqsubseteq \text{publication} \\
\text{publication} & \sqsubseteq \exists \text{hasAuthor} \land \exists \text{cites}
\end{align*}
\]

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 [36, 41].

As mentioned before, the entailment problem \(D, \Sigma \models Q\) is known to be decidable if \(\Sigma\) 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 \(\Sigma\) be a set of rules for which the treewidth of \(I(D \cup \Sigma)\) is bounded. Let \(\Omega[x]\) be a NEMODEQ.
1. \( D \cup \Sigma \models \exists y \forall x \exists w \, (\text{hasAuthor}(x, w)) \) if and only if \( D \cup \Sigma \cup \{ \neg \exists y \forall x \exists w , \text{hasAuthor}(x, w) \} \) has no countable model with bounded treewidth.

2. If Conjecture 12 holds, then \( D \cup \Sigma \models \exists y \forall x \exists w \, (\text{hasAuthor}(x, w)) \) is decidable.

The proof of this theorem exploits universality of \( I(D \cup \Sigma) \) 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 \( D \cup \Sigma \cup \{ \neg \text{msort}(\exists y \forall x \exists w , \text{hasAuthor}(x, w)) \} \) is equivalent to \( D \cup \Sigma \cup \{ \neg \exists y \forall x \exists w \}$ 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 Data-

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 initial query and TGDs.

Example 11. Given the rule set from Example 9 and the conjunctive query \( \exists y, w, (\text{hasAuthor}(y, w)) \), an appropriate first-order rewriting is \( \exists y, w, (\text{hasAuthor}(y, w)) \lor \exists y, (\text{publication}(y)) \).

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 FO query \( \exists y, z \, (r(x, y) \land r(y, z)) \rightarrow r(x, z) \) under the TGD \( r(x, y) \land r(y, z) \rightarrow r(x, z) \).

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

\[
\begin{align*}
& r(x, y) \rightarrow r_{\text{DB}}(x, y) \\
& r_{\text{DB}}(x, y) \land r_{\text{DB}}(y, z) \rightarrow r_{\text{DB}}(x, z) \\
& s(y) \land r_{\text{DB}}(y, w) \land s(w) \rightarrow \text{goal},
\end{align*}
\]

but also as a conjunctive regular path query

\[
\exists y, w, s(y) \land r(w, y) \land s(w).
\]

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 [41]. 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 \( \Sigma \) be a set of TGDs and let \( Q(x) \) be a CQ. A (NE)MODEQ \( \Sigma_{\text{Q}x} \) is a rewriting of \( Q \) under \( \Sigma \) if, for all databases \( D \) and potential query answers \( c \), we have \( D \cup \Sigma \models Q(c) \) iff \( D \models \Sigma_{\text{Q}x}(c) \). \( \Sigma \) is called (NE)MODEQ-rewritable if every conjunctive query \( Q \) has a (NE)MODEQ rewriting for \( \Sigma \) 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 of TGDs, \( \Sigma \), a conjunctive query \( Q = \exists y, \psi(x, y) \), and a NEMODEQ \( \Sigma_{\text{Q}x} \) that is a rewriting for \( Q \) and \( \Sigma \). Then \( Q \) and \( \Sigma \) are equivalent in all models of \( \Sigma \), i.e., \( \Sigma \models Q(x) \leftrightarrow \Sigma_{\text{Q}x}(x) \).

Let \( \psi'(x, y') \) be the conjunction of \( Q \) with variables \( x \) replaced by terms \( t \) and variables \( y \) replaced by variables \( y' \). We say that \( \psi'(x, y') \) is a match in a Datalog rule \( \rho \) if \( \rho \) is of the form \( \psi(x, y) \land \varphi \rightarrow \chi \), where \( y' \) occurs neither in \( \varphi \) nor in \( \chi \).

Given some NEMODEQ \( \Psi[z] \) over \( \Sigma \), let \( \Psi'[z] \) denote a NEMODEQ obtained by replacing a match \( \psi(x, y') \) of \( Q \) in some rule of \( \Psi \) by \( \Sigma(x) \), where the bound variables in \( \Sigma \) do not occur in \( \Psi \). Then \( \Psi \) and \( \Psi' \) are equivalent in all models of \( \Sigma \), i.e., \( \Sigma \models \forall z. \Psi(z) \leftrightarrow \Psi'(z) \).

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 Data-

log 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 \( \Sigma \) 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 \( p(t) \) for some head predicate \( p \) and the rule’s head contains an atom \( q(t') \), then \( t \) and \( t' \) 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 \( \Sigma_{\text{family}} \) is 3-oriented. We use atoms parentsSon(x, y, z) and parentsDgthr(x, y, z) to denote that \( z \) is the son and daughter of \( x \) and \( y \), respectively.

\[
\begin{align*}
& \text{parentsSon}(x, y, z) \land \text{hasBrother}(z, y) \rightarrow \text{parentsSon}(x, y, z) \\
& \text{parentsSon}(x, y, z) \land \text{hasSister}(z, y) \rightarrow \text{parentsDgthr}(x, y, z) \\
& \text{parentsDgthr}(x, y, z) \land \text{hasBrother}(z, y) \rightarrow \text{parentsSon}(x, y, z) \\
& \text{parentsDgthr}(x, y, z) \land \text{hasSister}(z, y) \rightarrow \text{parentsDgthr}(x, y, z)
\end{align*}
\]

Definition 11 (Single Predicate Rewriting). For a \( j \)-oriented set \( \Sigma \) of Datalog rules and a head predicate \( p \) of \( \Sigma \), a MODEP \( \exists p.x \)
is defined as follows. Let \( U \) be an auxiliary unary predicate for each head predicate \( q \) in \( \Sigma \), let \( V \) be a auxiliary unary predicate for each \( i \in \{1, \ldots, a(p)\} \) with \( i \neq j \), and let \( z_j \) be an additional variable not occurring in \( \Sigma \). Then \( \Sigma_{p,z} \) contains the following rules:

- A rule \( u_i \rightarrow \text{hit} \),
- for each set variable \( V \), a rule \( \rightarrow V \) with empty body,
- for each \( \psi \rightarrow q(t_1, \ldots, t_n) \in \Sigma \), a rule \( \psi' \rightarrow u_i(t) \) where \( \psi' \) is obtained from \( \psi \) by replacing each atom \( q(t_1, \ldots, t_n) \) (with \( q' \) a head predicate) by \( u_i(t) \), and by adding for each term \( t_i \) with \( i \) a new body atom \( V_i(t) \),
- A rule \( q'(t_1, \ldots, t_n) \rightarrow u_i(z_j) \) for each head predicate \( q' \).

For a list \( z \) of \( at(p) \) variables, the MODEQ \( \Sigma_{p,z} \) is defined as \( \Sigma_{p,z} \).

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

**Theorem 15** (Single predicate rewriting correctness). If \( \Sigma \) is \( j \)-oriented and \( p \) is a head predicate, then \( \Sigma_{p,z} \) is a rewritings for \( \Sigma \) and \( p(z) \).

**Example 15.** For the rule set in Example 14, we obtain the rewriting \( \Sigma \text{parentsSon}(z_1, z_2, z_3) = \Sigma_{\text{parentsSon}(z_1, z_2, z_3)} \) with rules

\[
\begin{align*}
\text{parentsDghtr}(z_1, z_2, z_3) & \rightarrow \text{hit} \rightarrow V_1(z_1) \rightarrow V_2(z_2), \\
V_1(z_1) & \wedge V_2(z_2) \rightarrow \text{parentsSon}(z_1, z_2, z_3).
\end{align*}
\]

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

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 \( \Sigma \)-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 losing 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 \( \rho_1 = B_1 \rightarrow H_1 \) and \( \rho_2 = B_2 \rightarrow H_2 \) be two TGDS. We say that \( \rho_2 \) depends on \( \rho_1 \), written \( \rho_1 \prec \rho_2 \), if there is

- a database \( D \),
- a substitution \( \theta \) of all variables in \( B_1 \) with terms in \( D \) such that \( \theta(B_2) \subseteq D \), and
- a substitution \( \theta' \) 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 \( \rho_2 \) strongly depends on \( \rho_1 \), written \( \rho_1 \ll \rho_2 \), if \( H_1 \) contains a predicate that occurs in \( B_2 \).

A (strong) cut of a set of rules \( \Sigma \) is a partition \( \Sigma_1 \cup \Sigma_2 \) of \( \Sigma \) such that no rule in \( \Sigma_1 \) (strongly) depends on a rule in \( \Sigma_2 \). It is denoted \( \Sigma_1 \perp \Sigma_2 \) \((\Sigma_1 \nsucc \Sigma_2)\).

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:

\[
\begin{align*}
A(x) \land B(x) & \rightarrow C(x) \quad (31) \\
C(x) & \rightarrow \exists v. p(x, v), A(v) \quad (32)
\end{align*}
\]

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 \( \theta = \theta' = [x \rightarrow c] \).

Intuitively speaking, we can evaluate a TGDS set \( \Sigma \) of the form \( \Sigma_1 \cup \Sigma_2 \) by first applying the rules in \( \Sigma_1 \), and then applying the rules of \( \Sigma_2 \). This is the essence of the following theorem, shown in [4].

**Theorem 17** (Baget et al.). Let \( \Sigma \) be a set of rules admitting a cut \( \Sigma_1 \nsucc \Sigma_2 \). Then, for every database \( D \) and every conjunctive query \( Q(x) \) we have that \( D \cup \Sigma \models Q(x) \) exactly if there is a Boolean conjunctive query \( Q' \) such that \( D \cup \Sigma_1 \models Q' \) and \( Q' \cup \Sigma_2 \models Q(x) \).

We can thus rewrite queries in “layers” based on cuts:

**Lemma 18** (Query rewriting with cuts). Let \( D \) be a database and let \( \Sigma_1 \nsucc \Sigma_2 \) be two sets of TGDS.

1. If \( \Sigma_{Q,x} \models Q(x) \) is a NEMODEQ-rewriting of a conjunctive query \( Q(x) \), then:

\[ D \cup \Sigma_1 \cup \Sigma_2 \models Q(x) \] if and only if \( D \cup \Sigma_1 \models Q_{Q,x} \).

2. If \( \Sigma_1 \) and \( \Sigma_2 \) 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 TGDS languages for which query answering is decidable, as expressed in the next theorem.

**Theorem 19** (query answering with cuts). Consider rule sets \( \Sigma_1 \nsucc \Sigma_2 \), such that NEMODEQ answering is decidable under \( \Sigma_1 \), and NEMODEQ rewriting is decidable under \( \Sigma_2 \). Then NEMODEQ answering is decidable under \( \Sigma_1 \cup \Sigma_2 \).
Using Theorem 13, this result specifically applies in cases where $\Sigma_i$ 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 $\prec$. The set $\Sigma$ is fully oriented if, for every $p \in \Sigma$, the equivalence class $[p]_{\approx_{\prec\prec}} = \{p' \in \Sigma \mid p \approx_{\prec\prec} p'\}$ 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 $[p]_{\approx_{\prec\prec}}$ as in Definition 11, and combine these rewritings using Lemma 14:

**Theorem 20 (Fully oriented rule sets are rewritable).** For a rule set $\Sigma$, it can be detected in polynomial time if $\Sigma$ is fully oriented. Every fully oriented set $\Sigma$ is MODEQ-rewritable.

The use of $\approx_{\prec\prec}$ 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 $\Sigma$, the rule set datalog($\Sigma$) of Definition 7 is fully oriented. Moreover, for every NEMODEQ-rewritable set $\Sigma$ of TGDs, there is a fully oriented set of Datalog rules $\Sigma'$ such that:

1. every predicate $p$ in $\Sigma$ has a corresponding head predicate $q_p$ in $\Sigma'$ that does not occur in $\Sigma$.
2. for every database $D$ and conjunctive query $Q(x)$ that do not contain predicates of the form $q_p$, and for every list of constants $c, D \cup \Sigma \models Q(c)$ if and only if $D \cup \Sigma' \models Q'(c)$ 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 $R_1(x_0, x_1) \land \ldots \land R_n(x_{n-1}, x_n) \rightarrow R_1(x_0, x_n)$, which can be viewed as a generalized transitivity statement. Even in relatively expressive DLs, role inclusion axioms lead to undecidability of CQ answering [36]; in more expressive DLs they even lead to undecidability of simpler reasoning problems [31]. To overcome this, syntactic restrictions are imposed on role inclusions, to ensure that all role inclusions can be captured using finite automata [31, 34]. This is equivalent to rewriting role inclusions to regular path queries, and has indeed been exploited to decide CQ answering over expressive DLs [41]. This can be viewed as an implicit application of Theorem 19. 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 [36, 41]. 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 dependencies that allow for universal models with bounded treewidth – a property shared by many of the known decidable TGD classes.

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 our 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


[33] Y. Kazakov. RIQ and SROIQ are harder than SHOIQ. In Brewka and Lang [8], pages 274–284.


As a conjunction containing normal atoms and MODEP atoms), we can check this for every conjunct separately. For normal ground atoms p(e), we have D |= p(e) if and only if p(e) ∈ D, which can be checked in polynomial time. For MODEP ground atoms P(e), we find that D |= P(e) if there is a Datalog derivation of hit from D via applying rules from P[e/A]. Due to the arity restriction of head predicates in P there are only polynomially many atoms that can be derived via P[e/A], therefore there must be a proof for P(e) of polynomial size.

Hence, we define the witness for e being an answer for D and \( C = \exists y \phi(x,y) \) to contain the tuple d providing us with bindings to the existentially quantified variables and a polynomial-size proof for every MODEP atom P(e) contained in \( \psi(e,d) \). Clearly this witness can be verified in polynomial time.

\[ \square \]

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

**Proof.** The linear construction time follows directly from the definition (taking into account that for every regular expression, the corresponding automaton can be obtained in linear time). The rest of the claim is an immediate consequence of the fact that P and modep(P) are equivalent for every conjunctive 2-way regular path predicate P. This can be seen as follows (letting \( I \) be an arbitrary interpretation):

To see that \( P^I \subseteq \text{modep}(P)^I \) we recall that \( P^I \) contains those pairs \( (\delta, \delta') \) where \( \delta' \) can be reached from \( \delta \) over a path \( \delta = \delta_0 \cdot \ldots \cdot \delta_n = \delta' \) such that \( \gamma_1 \ldots \gamma_k \) satisfies the regular expression \( P \). The latter is the case exactly if the corresponding automaton \( A_P \) accepts \( \gamma_1 \ldots \gamma_k \) i.e., there is a sequence \( \langle s_0, \ldots, s_n \rangle \) of states with \( s_0 \) being an initial and \( s_n \) being a final state such that \( \langle s_i,1, \gamma_i, s_i \rangle \in T \) for all \( i \in \{1, \ldots , n\} \). We can now show that \( (\delta, \delta') \) must be in \( \text{modep}(P)^I \) by noting that in every extension \( I' \) of \( I \) satisfying the rules from modep(P) as well as \( \delta = \lambda'_1 \ldots \lambda'_k \) and \( \delta' = \lambda'_1 \ldots \lambda'_k \) (1) we have \( \delta \in U_{\lambda_0} \) due to the according initial-state-rule, (2) we can thus infer by induction that \( \delta_i \in U_{\lambda_i} \) by the according transition rules (3) we hence arrive at \( \delta' \in U_{\lambda_n} \) and can apply the final-state-rule to prove that \( \text{hit} \) must hold.

The proof of \( \text{modep}(P)^I \subseteq P^I \) is very similar: starting from a derivation sequence for \( \text{hit} \), we can easily construct an according path from \( \delta \) to \( \delta' \).

\[ \square \]

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

**Proof.** The linear construction time follows directly from the definition. For showing the equivalence claim between the monadic Datalog query \( Q = (\text{goal}, \mathbb{S}) \) and modep(Q), we assume w.l.o.g. that all unary IDB predicates in Q are already of the shape \( U_i \) and therefore the rule sets of \( \mathbb{S} \) and modep(Q) coincide on all rules not referring to goal or hit. Given an interpretation \( I \), let \( I'' \) denote the extended interpretation obtained by saturating \( I \) under these rules. Then, a tuple \( \delta \) from \( I \) is in the extension of Q if \( I'' \models B(\delta/x) \) for the body of some rule \( B \rightarrow \text{goal}a[x] \) from \( \mathbb{S} \). Yet, since modep(Q) contains a rule \( B[a/x] \rightarrow \text{hit} \) this is exactly the case if \( \delta \) is in the extension of modep(Q).

\[ \square \]
assume that the MODEPs in |Q| use distinct names for their auxiliary unary predicates \( U \) and for their constants \( \lambda \), again without loss of generality. Let \( \ell \) be the maximal number of variables in any of the rules used (in MODEPs) in \( |Q| \), and let \( m \) be the total number of constants \( \lambda \) (in MODEPs) in \( |Q| \) (i.e., the sum of the arities of all MODEPs in \( |Q| \)).

We first define a structure \( S \) of the shape illustrated in Fig. 3 to contain exactly the following relations:

- \( S \) contains a chain of relations \( a \xrightarrow{\text{linkedTo}} c \xrightarrow{\text{linkedTo}} \ldots \xrightarrow{\text{linkedTo}} b \);
- for each \( e_i \) (\( i = 1, \ldots, 2\ell \)), \( S \) contains a chain \( e_i \xrightarrow{\text{certifiedby}} f_i \xrightarrow{\text{certifiedby}} \ldots \xrightarrow{\text{certifiedby}} c_i \);
- the elements \( a \) and \( b \) are in the unary relation friend.

Thus \( S \) has \( 2(\ell + 1) + 3 \) elements. We now use multiple copies of \( S \) to define a structure \( T \) of the shape that is illustrated in Fig. 4. The structure \( T \) consists of \( m + 1 \) copies of \( S \), connected in a chain. Formally, \( T \) is defined as the disjoint union of \( m + 1 \) copies \( S_1, \ldots, S_m \) of \( S \), factorized by the equivalence relation \( \{ b_i \approx a_{i+1} | 0 \leq i < m \} \), where we use \( a_i \) and \( b_i \) to denote the elements \( a \) and \( b \), respectively, from \( S_i \).

Clearly, \( \{ v \mapsto a_{0}, w \mapsto b_{0} \} \) is an answer to \( |Q| \) over \( T \). Thus, by assumption, \( \{ v \mapsto a_{0}, w \mapsto b_{0} \} \) is also an answer to \( |Q| \) over \( T \). This match is therefore entailed in every model of \( T \). By Theorem 2 and Fact 1, we only need to consider the universal model \( I(T) \), the elements of which are in one-to-one correspondence with the elements of \( T \). Thus, the validity of the answer \( \{ v \mapsto a_{0}, w \mapsto b_{0} \} \) is witnessed by a query match on \( I(T) \), which is based on a particular match for each MODEP \( Q \) in \( |Q| \). We can consider these matches as a list of \( m \) elements of \( I(T) \) (and thus also of \( T \)) which were used to interpret the constants \( \lambda \) in \( |Q| \). We call these the matched elements of \( T \). Since there are only \( m \) constants \( \lambda \) in \( |Q| \), there are at most \( m \) matched elements. Hence, there is some \( S_\ell \) in \( T \) which does not contain any matched elements, other than possibly \( a_0 \) and \( b_0 \) (since they also belong to adjacent substructures).

We construct a new structure \( T' \) from \( T \) as follows:

- For all \( e_0 \) in the substructure \( S_0 \) of \( T \) with \( i = 2, \ldots, 2\ell \), add a chain \( (e_0 \xrightarrow{\text{certifiedby}} f_{0} \xrightarrow{\text{certifiedby}} \ldots \xrightarrow{\text{certifiedby}} c_{0} ) \), where \( c_{0} \) and the \( f_{0} \) are fresh elements distinct from any element in \( T \).
- Delete all elements \( (f_{2i} \xrightarrow{\text{certifiedby}} f_{2i+1}, \ldots, (f_{2i} \xrightarrow{\text{certifiedby}} f_{2i+1}) \).

\( T' \) no longer contains a chain of links \( a_{0}, b_{0} \) that is certified by a single authority: only the first \( 2\ell - 1 \) elements are certified by \( e_0 \), and only the last \( 2\ell - 1 \) elements are certified by \( c_0 \). Likewise, it is not possible to split the chain between \( a_0 \) and \( b_0 \) into multiple chains of links, since none of the intermediate elements belongs to the class friend. Thus \( \{ v \mapsto a_{0}, w \mapsto b_{0} \} \) is not an answer to \( |Q| \).

Nevertheless, we can show that \( \{ v \mapsto a_{0}, w \mapsto b_{0} \} \) is an answer to \( |Q| \), which yields the required contradiction. To do this, it suffices to show that the matches of MODEPs in \( |Q| \) that witness the answer \( \{ v \mapsto a_{0}, w \mapsto b_{0} \} \) over \( T \) are still valid. Thus consider some MODEP \( P \in |Q| \) with auxiliary constants \( \lambda_1, \ldots, \lambda_{\ell_{\text{ar}}} \), that have been matched to elements \( \delta = \delta_1, \ldots, \delta_{\ell_{\text{ar}}} \) of \( T \) (or, equivalently, \( I(T) \)). By construction of \( T' \), the elements \( \delta \) are also in \( T' \) and \( I(T') \). We show that \( \delta \in \bigotimes_{\ell_{\text{ar}}} \).

By our assumption, every extension \( I \) of \( I(T) \) with \( \lambda_{0} = \delta_0 \) and \( I \models P \) satisfies hit. Hence there is a sequence of applications of rules from \( P \) by which \( \text{hit} \) can be derived (using the pre-defined interpretation of \( \lambda_0 \)). The same sequence of rules can be used to derive \( \text{hit} \) for extensions \( I' \) of \( I(T') \) with \( \lambda'_{0} = \delta_0 \). Indeed, rules of \( P \) have at most \( \ell \) variables, hence their applicability does only depend on a substructure of size at most \( \ell \). It is easy to see that \( I' \) and \( I(T') \) have the same substructures of size \( \epsilon \) up to renaming of elements that are not referred to by any constants in \( P \). It can be shown by induction that this property is preserved when considering unary predicates derived by any number of rule applications. This shows that \( I' \models P \) implies \( I' \models \text{hit} \). □
By construction, $\mathcal{I}'$ satisﬁes datalog($\mathcal{P}$) and therefore, by ($\ast$), follows $\mathcal{J}', \mathcal{Z} \models p(t(x))$, i.e., $(\mathcal{Z}(x_1), \ldots, \mathcal{Z}(x_n)) \in p(t)^\mathcal{J}'$. Again by construction of $\mathcal{J}'$, this is the case only if hit'' = $\{0\}$, whence we have shown $\mathcal{I}' \models \text{hit}$, which ﬁnishes the second direction.

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

Proof. As established in Theorem 6, NEMODEQs are expressible as Datalog queries. However, for the latter, preservation under homomorphisms is well known and easy to show. □

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

Proof. We show the equivalence for NEMODEPs by an induction on their degree. From this, the general claim is a straightforward consequence.

Let $\mathcal{I}$ be an arbitrary interpretation and $\mathcal{Z}$ a variable assignment for $x$. Then we can establish the following sequence of equivalent propositions:

• $\mathcal{I}, \mathcal{Z} \models \mathcal{P}(x)$
• every extension $\mathcal{J}'$ of $\mathcal{I}$ with $\lambda^\mathcal{J}' = \mathcal{Z}(x)$ and $\mathcal{I}' \models \mathcal{P}$
• every extension $\mathcal{J}'$ of $\mathcal{I}$ with $\lambda^\mathcal{J}' = \mathcal{Z}(x)$ and $\mathcal{I}' \models \mathcal{P}$ with $\mathcal{P} = \mathcal{P}[\lambda/\text{hit}]$ must satisfy $\lambda$; in other words, no such extension exists

• (via induction hypothesis) no extension $\mathcal{J}'$ of $\mathcal{I}$ with $\lambda^\mathcal{J}' = \mathcal{Z}(x)$ simultaneously satisﬁes mso($\rho$) for all $\rho \in \mathcal{P}$

• no extension $\mathcal{J}'$ of $\mathcal{I}$ with $\lambda^\mathcal{J}' = \mathcal{Z}(x)$ satisfies $\bigwedge_{\rho \in \mathcal{P}} \text{mso}(\rho)$

• all extensions $\mathcal{J}'$ of $\mathcal{I}$ with $\lambda^\mathcal{J}' = \mathcal{Z}(x)$ satisfy $\bigwedge_{\rho \in \mathcal{P}} \text{mso}(\rho)$

• letting $\mathcal{I}_t$ denote $\mathcal{I}$ extended by $\{ \lambda \mapsto \mathcal{Z}(x) | 1 \leq i \leq m \}$, $\mathcal{I}_t, \mathcal{Z} \models \bigwedge_{\rho \in \mathcal{P}} \text{mso}(\rho)$ holds for every set-variable assignment $\Xi:

\begin{align*}
\{U_1, \ldots, U_k\} \rightarrow 2^{|\mathcal{P}|} \\
\mathcal{I}_t \models \forall U_1, \ldots, U_k \rightarrow \bigwedge_{\rho \in \mathcal{P}} \text{mso}(\rho) \\
\mathcal{I}, \mathcal{Z} \models \bigwedge_{\rho \in \mathcal{P}} \text{mso}(\rho)[x/\lambda]
\end{align*}

Thus we have shown the claimed correspondence. □

Theorem 9 (Complexity of NEMODEQ answering). Checking if $\mathcal{C}$ is an answer to a NEMODEQ $\mathcal{Q}[x]$ over a database $D$ is P-complete in the size of $D$, and PSpace-complete in the size of $D$ and $\mathcal{Q}$.

Proof. PSpace membership for combined complexity is a direct consequence of PSpace completeness of model checking in monadic second-order logic [45, 47], keeping in mind that due to Fact 1, entailment coincides with model checking if the premise is a set of ground facts.

We show PSpace hardness by providing a reduction from the validity problem of quantiﬁed Boolean formulae (QBFs). We recall that for any QBF, it is possible to construct in polynomial time an equivalent QBF that has the speciﬁc shape

$$Q_1(x_1) \land Q_2(x_2) \land \ldots \land Q_n(x_n) \rightarrow \lambda^\mathcal{E},$$

with $Q_1, \ldots, Q_n \in \{\exists, \forall\}$ and $\mathcal{E}$ being a set of sets of literals over the propositional variables $x_1, \ldots, x_n$. In words, we assume our QBF to be in prenex form with the propositional part of the formula in disjunctive normal form. For every literal set $L = \{x_1, \ldots, x_k\}$, $\neg x_1, \ldots, \neg x_k$, we now define the $n$-ary MODEP $p_L = [(\lambda_{x_1} \land \ldots \land \lambda_{x_k}) \land f(x_{\lambda_{x_1}}, \ldots, f(x_{\lambda_{x_k}}) \rightarrow \text{hit}]$, $$\lambda$$, $\neg$ the $n$-ary MODEP $p_{\lambda} = [p_L(\lambda_1, \ldots, \lambda_n) \rightarrow \text{hit}]$ if $L \in L$. Letting $p_2 = \text{phi}$, we now deﬁne MODEPs $p_{1,\ldots,n}$ in descending order. If $Q_i = 3$, then the $i$-ary MODEP $p_{i-1}$ is deﬁned as the single-ton rule set $p_{i-1}(\lambda_{A_1}, \ldots, \lambda_{A_{i-1}}) \rightarrow \text{hit}$. In case $Q_i = 3$, we let $p_{i-1} = p_{i-1}(\lambda_{A_1}, \ldots, \lambda_{A_{i-1}}) \rightarrow U(y) (U(y) \land f(y) \rightarrow \text{hit})$. Now, let $D$ be the database containing the two individuals 0 and 1 and the facts $f(0)$ and $f(1)$. We now show that the considered QBF is true exactly if $D \models p_{\lambda}$. To this end, we ﬁrst note that by construction the extension of $p_L$ contains exactly those $n$-tuples $(\lambda_1, \ldots, \lambda_n)$ for which the corresponding truth value assignment $val$, sending $x_k$ to $true$ if $\delta = 1$, makes the formula $\bigwedge_{\mathcal{E}} \lambda^\mathcal{E}$ true. In the same way, the extension of $p_{\lambda}$ represents the set of truth value assignments $\lambda^\mathcal{E}$ satisfying $\bigwedge_{\mathcal{E}} \lambda^\mathcal{E}$. Then, by descending induction, we can show that the extensions of $p_i$ encode the assignments to free propositional variables of the subformula $Q_{i+1} x_{i+1} \ldots Q_n x_n \bigwedge_{\mathcal{E}} \lambda^\mathcal{E}$ that make this formula true. Consequently, $p_0$ has a nonempty extension if the entire considered QBF is true.

For data complexity, P membership follows from expressibility in Datalog shown in Theorem 6, for which the problem is known to be in P [23]. P hardness follows from the respective result for MODEQs established in Theorem 3. □

Proofs for Section 5

In this section, we establish the proof of Theorem 10, which we repeat here for convenience.

Theorem 10 (Deciding Datalog containment in MSO). Consider a Datalog query $\langle \mathcal{G}, \mathcal{P} \rangle$ and an MSO query $\mathcal{Q}$. If the models of $\mathcal{G}$ are closed under homomorphism, then it is decidable if the query $\langle \mathcal{G}, \mathcal{P} \rangle$ is contained in $\mathcal{Q}$.

We establish this by transferring earlier results of Courcelle [20]. To do this in a self-contained way, we recall the main notions used in said work and relate them to our present formalism. Further details and examples can be found in [20]. The results in [20] only apply to Datalog queries and MSO formulae without free variables and without constant symbols. The next two results show how one can lift this to the general case.

Lemma 23. There is a linear time algorithm, which takes as an input an arbitrary Datalog query $\langle \mathcal{G}, \mathcal{P} \rangle$ and an MSO query $\mathcal{Q}$, and that returns a Boolean Datalog query $\langle \mathcal{G}^\prime, \mathcal{P}^\prime \rangle$ and a Boolean MSO query $\mathcal{Q}^\prime$ such that $\langle \mathcal{G}, \mathcal{P} \rangle$ is contained in $\mathcal{Q}$ iff $\langle \mathcal{G}^\prime, \mathcal{P}^\prime \rangle$ is contained in $\mathcal{Q}^\prime$.

Proof. Let $\langle \mathcal{G}, \mathcal{P} \rangle$ and $\mathcal{Q}$ be as in the claim. Assume that $\mathcal{Q}$ has $ar(\mathcal{G})$ free ﬁrst-order variables and no free second-order variables – if this is not the case, then query containment is trivially impossible and one can easily ﬁnd queries that satisfy the claim.

We reduce the containment problem to a containment problem of Boolean queries. To this end, let result be a fresh predicate symbol with $ar(\text{result}) = ar(\mathcal{G})$. We deﬁne a new Boolean Datalog query $\langle \mathcal{G}^\prime, \mathcal{P}^\prime \rangle$ by introducing a new nullary IDB predicate goal and setting

$$\mathcal{P}^\prime := \mathcal{P} \cup \langle \mathcal{G}(y) \land \text{result}(y) \rightarrow \text{goal} \rangle.$$
Elements in \( \langle I \rangle \) and only if the answers of \( o \) rule \( \in \) \( v \) max \mbox{imal set of isomorphic concrete graphs over graphs}. A bijective homomorphism. sources called one can still work with some concrete graph that represents a graph. vertices should not have any impact. For most practical purposes, \( (c_1, x) \) that satisfies \( (c_1) \) induces an interpretation \( I \) of the original signature \( (c_1) \) by identifying all domain elements in \( c_1 \) with \( c' \). Note that it is possible for two constants to have the same interpretation. Conversely, every interpretation \( I' \) over the original signature induces an interpretation \( I ' \) over the new signature with \( c_1 = c_1 \). It is easy to see that the answers of \( (g(1), P) \) over \( I \) are contained in the answers of \( \varphi ' \) over \( I ' \) if and only if the answers of \( (g(1), P) \) over \( I ' \) are contained in the answers of \( \varphi ' \) over \( I ' \), as required. 

In the remainder of this section, we thus consider only Boolean queries that do not contain constant symbols.

Essential to the discussion in [20] are specific notions of (hyper)graphs and a monadic second-order logic that we will introduce first. These definitions depend on a set \( P \) of predicates; constant symbols, considered in our earlier signatures, do not feature here.

**Definition 14.** A concrete graph over \( P \) is a tuple of the form \( \langle V, E, \text{lab}, \text{vert} \rangle \) where \( V \) is a set of vertices, \( E \) is a set of edges, \( \text{lab} : E \rightarrow P \) is an edge labelling function, and \( \text{vert} \) is a total function from edges \( e \in E \) to tuples \( \text{vert}(e) \) of length \( \text{ar}\{\text{lab}(e)\} \).

Given graphs \( G_1 \) and \( G_2 \), a homomorphism \( h : G_1 \rightarrow G_2 \) is a pair of mappings \( h_v : V_1 \rightarrow V_2 \) and \( h_e : E_1 \rightarrow E_2 \) such that \( \text{lab}_2 \circ \text{lab} = \text{lab}_1 \) and \( \text{vert}_2 \circ \text{vert} = \text{vert}_1 \) (where we apply \( h \) to a tuple of vertices by applying it to each component). An isomorphism is a bijective homomorphism.

Two concrete graphs are isomorphic if there is an isomorphism between them, and this defines an equivalence relation on concrete graphs. A graph is an equivalence class of this relation, i.e., a maximal set of isomorphic concrete graphs over \( P \) (and some base set of vertex names). A homomorphism between graphs \( G_1 \) and \( G_2 \) is a homomorphism from a concrete graph in \( G_1 \) to a concrete graph in \( G_2 \).

Abstracting from concrete graphs is relevant when generating expansion graphs from Datalog queries below, where the naming of vertices should not have any impact. For most practical purposes, one can still work with some concrete graph that represents a graph. Courcelle further considers graphs that have distinguished nodes, called sources, which we do not require here (i.e., we consider only graphs with an empty list of sources). To state properties of graphs logically, Courcelle uses a specific notion of two-sorted monadic second-order logic.

**Definition 15.** The multi-sorted monadic second-order logic MSO\_\_G is defined over two sorts: the vertex sort \( v \) and the edge sort \( e \). Given a set of predicate symbols \( P \), the set \( P_G \) of two-sorted predicates is defined as \( P_G : = \{ \text{edg}_p \mid p \in P \} \) where \( \text{ar}(\text{edg}_p) = (e, v, \ldots, v) \) with \( \text{ar}(p) \) occurrences of \( v \) (recall that arities in multi-sorted logic are tuples of sorts).

An MSO\_\_G formula (MSO\_\_G interpretation) over a signature of (unsorted) predicate symbols \( P \) is a formula (interpretation) of two-sorted monadic second-order logic over the signature of predicate symbols \( P_G \). The term unsorted formula (unsorted interpretation) is used to emphasize that a formula (interpretation) is not MSO\_\_G.

In the following, we consider the predicate signature \( P \) to be fixed and do not mention it explicitly. MSO\_\_G interpretations \( \mathcal{I} \) can be considered as graphs \( G \) in an obvious way, with each tuple \( (e, \delta) \in \text{edg}\_\_G \) corresponding to a \( p \)-labelled edge between the vertices \( \delta \) in (any concrete graph representing) \( G \). We will freely switch between these perspectives. Both can capture structures with multiple edges that have the same label and the same list of vertices. By forgetting the multiplicity of edges, we can transform graphs into unsorted interpretations:

**Definition 16.** For an MSO\_\_G interpretation \( \mathcal{I} \), the unsorted interpretation \( F(\mathcal{I}) \) is defined as follows:

- \( \Delta^{F(\mathcal{I})} = \Delta^{\mathcal{I}} \).
- for all \( p \in P \) and \( \delta \in (\Delta^{\mathcal{I}})^{\varphi(p)} \), set \( \delta \in p^{F(\mathcal{I})} \) iff \( \langle e, \delta \rangle \in \text{edg}_p \) for some \( e \in \Delta^{\mathcal{I}} \).

Given a homomorphism of graphs \( h : \mathcal{I}_1 \rightarrow \mathcal{I}_2 = (h_v, h_e) \), the homomorphism of interpretations \( F(h) : F(\mathcal{I}_1) \rightarrow F(\mathcal{I}_2) \) is defined as \( F(h) = h \).

Conversely, for an unsorted interpretation \( I \), the MSO\_\_G interpretation \( G(I) \) is defined as follows:

- \( \Delta^{G(I)} : = \Delta^I \).
- for all \( p \in P \) set \( \text{edg}_p^{G(I)} : = \{(e, \delta) \mid e = \langle p, \delta \rangle \in \Delta^{G(I)} \} \).

Given a homomorphism of interpretations \( h : I_1 \rightarrow I_2 \) the homomorphism of graphs \( G(h) : G(I_1) \rightarrow G(I_2) \) consists of the mappings \( G(h)_v = h_v \) and \( G(h)_e \) defined as \( \langle p, \delta \rangle \mapsto \langle p, h(h) \rangle \) (applying \( h \) to the tuple \( \delta \) component-wise).

It is easy to verify that \( F \) and \( G \) are well-defined, in particular that \( F(h) \) and \( G(h) \) are indeed homomorphisms. Algebraically speaking, \( F \) and \( G \) thus define functors between the categories of unsorted interpretations (with their homomorphisms) and graphs (with their homomorphisms). \( G \) has also been defined in [20, Definition 3.2]. The following is immediate from the definition:

**Lemma 25.** \( I = F(G(I)) \) for all interpretations \( I \).

An MSO\_\_G formula \( \varphi \) is insensitive to multiple edges if \( \mathcal{J} \models \varphi \) if and only if \( G(F(\mathcal{J})) \models \varphi \). We can also transform MSO formulas to MSO\_\_G formulas:

**Definition 17.** Given an MSO formula \( \varphi \), the MSO\_\_G formula \( g(\varphi) \) is obtained by the following replacements:

- replace each unsorted (first-or second-order) variable \( v \) in \( \varphi \) by a fresh (first-or second-order) variable \( v' \) of sort \( v \);
- replace each atomic subformula \( p(x) \) in \( \varphi \) with \( p \in P \) by the formula \( \exists z. \text{edg}_p(z, x) \) where \( z \) is a fresh first-order variable of sort \( e \).

**Lemma 26.** For all MSO\_\_G interpretations \( \mathcal{J} \) and MSO formulæ \( \varphi \), we have \( \mathcal{J} \models g(\varphi) \iff F(\mathcal{J}) \models \varphi \).
Lemma 27. Every formula of the form $g(\varphi)$ is insensitive to multiple edges.

Proof. Given any MSO$_2$ interpretation $J$, by Lemma 26, $J \models g(\varphi)$ iff $F(J) \models \varphi$. By Lemma 25, the latter is equivalent to $F(G(F(J))) \models \varphi$. This is equivalent to $G(F(J)) \models g(\varphi)$, again by Lemma 26. Hence $J \models g(\varphi)$ iff $G(F(J)) \models g(\varphi)$. □

Lemma 28. If the models of $\varphi$ are closed under homomorphisms, then so are the models of $g(\varphi)$.

Proof. Consider a homomorphism $h : J_1 \rightarrow J_2$ between MSO$_2$ interpretations, and assume that $J_1 \models g(\varphi)$. By Lemma 26, $J_1 \models g(\varphi)$ implies $F(J_1) \models \varphi$. By Definition 16, $F(h) : F(J_1) \rightarrow F(J_2)$ is a homomorphism of interpretations. Since the models of $\varphi$ are closed under homomorphisms, $F(J_1) \models \varphi$ implies $F(J_2) \models \varphi$. By Lemma 26, $J_2 \models g(\varphi)$, as claimed. □

This completes our analysis of the relationship between our notion of MSO and Courcelle’s MSO$_2$ defined on graphs in the sense of Definition 14. We can relate Datalog queries to graphs as follows:

Definition 18. Let $\langle \text{goal}, \mathcal{P} \rangle$ be a Boolean Datalog query. A partial expansion is a conjunction of first-order atoms, with the set of partial expansions of $\langle \text{goal}, \mathcal{P} \rangle$ defined inductively as follows:

- The atom propositional $\text{goal}(\cdot)$ is a partial expansion.
- If there is a partial expansion of form $p(y) \land \varphi$ and a rule $\psi \rightarrow p(x) \in \mathcal{P}$, then $\psi \varphi \land \varphi$ is a partial expansion, where $\varphi$ is a renaming that bijectively maps variables in $\psi \rightarrow p(x)$ to fresh variable names not used in $p(y) \land \varphi$ yet, and $\psi$ is a most-general unifier of $p(x)\sigma$ and $p(y)$.

Moreover, a (complete) expansion of $\langle \text{goal}, \mathcal{P} \rangle$ is a partial expansion of $\langle \text{goal}, \mathcal{P} \rangle$ that contains no IDB predicates.

Every expansion corresponds to a concrete graph that has variables for its vertices and atoms labelled by predicates for its edges. This induces a set $L(\text{goal}, \mathcal{P})$ of expansion graphs of $\langle \text{goal}, \mathcal{P} \rangle$.

Expansion graphs are called computation graphs in [20]. It is well-known that every Datalog query is equivalent to the (infinite) disjunction of its expansions, where all variables are existentially quantified. Restating this in terms of expansion graphs and universal models, we obtain:

Lemma 29. A database $D$ entails a Boolean Datalog query $\langle \text{goal}, \mathcal{P} \rangle$ iff there is some $G \in L(\text{goal}, \mathcal{P})$ for which there is a homomorphism $G \rightarrow G(\text{I}(D))$.

Using the above terminology, we can reformulate part of Theorem 5.5 of [20] as follows:

Theorem 30. (Courcelle). If $\varphi$ is an MSO$_2$ formula that is insensitive to multiple edges and whose models are closed under homomorphisms, then it is decidable whether $G \models \varphi$ for all $G \in L(\text{goal}, \mathcal{P})$.

Note that we identify graphs and MSO$_2$ interpretations when writing $G \models \varphi$.

Lemma 31. Theorem 10 holds for Boolean queries without constant symbols.

Proof. Let $\langle \text{goal}, \mathcal{P} \rangle$ and $\varphi$ be as in the claim. We claim that $\langle \text{goal}, \mathcal{P} \rangle$ is contained in $\varphi$ if and only if $G \models g(\varphi)$ for all $G \in L(\text{goal}, \mathcal{P})$. By Lemma 28, the models of $g(\varphi)$ are closed under homomorphisms, and, by Lemma 27, $g(\varphi)$ is insensitive to duplicate edges. Hence, by Theorem 30, $(\ast)$ is decidable.

It remains to show that query containment is equivalent to $(\ast)$. For the “if” direction, assume that $(\ast)$ holds. Consider an arbitrary database $D$ that entails $\langle \text{goal}, \mathcal{P} \rangle$. By Lemma 29, this is equivalent to the existence of an expansion graph $G \in L(\text{goal}, \mathcal{P})$ for which exists a homomorphism $G \rightarrow G(\text{I}(D))$. By $(\ast)$ we have $G \models g(\varphi)$. Therefore, $G(\text{I}(D)) \models g(\varphi)$ by Lemma 28. By Lemma 26, $G(\text{I}(D)) \models \varphi$, and thus $D$ entails $\varphi$. Since $D$ was arbitrary, this shows the claimed query containment.

For the “only if” direction, assume that $\langle \text{goal}, \mathcal{P} \rangle$ is contained in $\varphi$. Let $G \in L(\text{goal}, \mathcal{P})$ be arbitrary. By Definition 18, $F(G)$ satisfies the expansion of $\langle \text{goal}, \mathcal{P} \rangle$ that induced $G$. Therefore, $F(G) \models \mathcal{P} \rightarrow \text{goal}$. It is easy to see that the (finite) interpretation $F(G)$ is equal to $I(D)$ for a database $D$. Thus $D$ entails $\langle \text{goal}, \mathcal{P} \rangle$, and, by the assumed containment, $D$ also entails $\varphi$. Hence, $I(D) = F(G) \models \varphi$.

Thus, by Lemma 26, $G \models g(\varphi)$. Since $G$ was arbitrary, this shows $(\ast)$. □

Proof of Theorem 10. According to Lemmas 23 and 24, the containment problem can be reduced to the containment problem of Boolean queries without constants. It is important to note that the constructions in both cases preserve the closure of an MSO formula under homomorphisms. The claim is then a consequence of Lemma 31. □

Proofs for Section 6

Theorem 13 (NEMODEQ answering under rts). Let $D$ be a database and let $\Sigma$ be a set of rules for which the treewidth of $I(D) \cup (\Sigma)$ is bounded. Let $\Sigma[x]$ be a NEMODEQ.

1. If $D \cup (\Sigma \cup \neg \Sigma[x])$ has no countable model with bounded treewidth.

2. If Conjecture 12 holds, the problem $D \Sigma[x]$ is decidable.

Proof. We start by proving the first claim. For the “only if” direction, it is obvious that $D \cup \Sigma \models \Sigma[x]$ implies that $D \cup \Sigma \cup \neg \Sigma[x]$ has no model (and therefore no model with bounded treewidth).

For proving the “if” direction, we assume that $D \cup \Sigma \cup \neg \Sigma[x]$ has no countable model with bounded treewidth. Then it must be unsatisfiable for the following reason: toward a contradiction assume it has a model $I$. Since $I \models G \models \Sigma$, there must be a homomorphism from $I(D) \cup \Sigma$ to $I$. Thus, by contraposition of the closedness of models of $\Sigma[x]$ under homomorphisms (Theorem 7), $I(D \cup \Sigma)$ itself satisfies $\neg \Sigma[x]$ and is therefore a model of $D \cup \Sigma \cup \neg \Sigma[x]$. Moreover, since $\Sigma$ is bts, we obtain that $I(D)$ of $\Sigma$ has bounded treewidth. Obviously it is also countable, which yields the desired contradiction and ensures unsatisfiability of $D \cup \Sigma \cup \neg \Sigma[x]$.

For the second claim we start from the previous claim and note that, since $D$ and $\Sigma$ are first-order theories and $\neg \Sigma[x]$ is expressible as an MSO formula, $D \cup \Sigma \cup \neg \Sigma[x]$ is expressible as an MSO theory. Thus, we can invoke Conjecture 12 to obtain the desired result. □

Proofs for Section 7

For the following considerations we capitalize on the fact that a set of Datalog rules can be viewed as a (possibly infinite) collection of conjunctive queries that are obtained by expanding rules by repeated backward-chaining (cf. Definition 18). Every expansion
\( \phi[x] \) with variables \( x \) can naturally be associated with an interpretation structure \( I(\phi) \): its domain \( \Delta^{(\phi)} \) is \( x \cup C \), for each constant \( c \in C \) we set \( c^{I(\phi)} := c \), and we have \( t^{I(\phi)} \) exactly if \( t(\phi) \) occurs in \( \phi \).

**Lemma 14 (Replacement Lemma).** Consider a set \( \Sigma \) of TGDs, a conjunctive query \( Q = 3y.p_1(t_1) \land \ldots \land p_n(t_n) \) is obtained from \( Q(x) \) by replacing all head atoms \( p(t) \) with the MODEQ \( \Sigma_0 \) \( \simeq (t_1/x) \) as in Definition 11. We assume a fixed sequence of replacement steps in the construction of the rewriting. Let \( \Sigma_0 \) denote the original query \( Q(x) \), let \( \Sigma \), with \( 1 \leq i \) denote the result after each subsequent replacement step, and let \( \Sigma \) be the final result.

One can show \( \Sigma \models \forall x. Q(x) \iff \Sigma[x] \) by induction over \( i \). The base case follows since \( \Sigma_0 \) is clearly equivalent to \( Q \). The induction steps follow from Lemma 14 and Theorem 15.

**Theorem 16**. The condition of Definition 9. Thus consider an arbitrary database \( D \) and a potential query answer \( c \). Without loss of generality, we assume that \( D \) contains no fact of the form \( q'(d) \) for some head predicate \( q' \) of \( \Sigma \). Indeed, if it does, we can replace \( q \) by a fresh predicate \( q'' \) and add a rule \( \forall x.q''(x) \to q'(x) \) to \( \Sigma \). This modification clearly preserves \( j \)-orientability.

**Proof.** A rewriting for a \( j \)-oriented rule set \( \Sigma \) and a CQ \( Q[x] = 3y.p_1(t_1) \land \ldots \land p_n(t_n) \) is obtained from \( Q(x) \) by replacing all head atoms \( p(t) \) with the MODEQ \( \Sigma_0 \simeq (t_1/x) \) as in Definition 11. We assume a fixed sequence of replacement steps in the construction of the rewriting. Let \( \Sigma_0 \) denote the original query \( Q(x) \), let \( \Sigma \), with \( 1 \leq i \) denote the result after each subsequent replacement step, and let \( \Sigma \) be the final result.

One can show \( \Sigma \models \forall x. Q(x) \iff \Sigma[x] \) by induction over \( i \). The base case follows since \( \Sigma_0 \) is clearly equivalent to \( Q \). The induction steps follow from Lemma 14 and Theorem 15.

**Theorem 19 (Query Answering with Cuts).** Consider rule sets \( \Sigma_1 \supset \Sigma_2 \) such that MODEQ answering is decidable under \( \Sigma_1 \) and MODEQ rewriting is decidable under \( \Sigma_2 \). Then MODEQ answering is decidable under \( \Sigma_1 \cup \Sigma_2 \).
Proof. This is a direct consequence of Lemma 18 (1). Given any conjunctive query $Q$, we can compute the rewriting $\Sigma_{Q|\Sigma}$, and compute its answers over $D \cup \Sigma$. 

**Theorem 20.** (Fully oriented rule sets are rewritable). For a rule set $\Sigma$, it can be detected in polynomial time if $\Sigma$ is fully oriented. Every fully oriented set $\Sigma$ is MODEQ-rewritable.

Proof. Clearly, the relation $\approx_{\Sigma}$ can be constructed in polynomial time by checking $\rho \ll \rho'$ for each pair of rules and constructing the reflexive symmetric transitive closure. The equivalence classes $[\rho]_{\approx_{\Sigma}}$ are obtained from this in linear time. It is clear that $j$-orientedness can be checked for each set of rules in polynomial time.

It remains to show the second part of the claim. We say that $[\rho]_{\approx_{\Sigma}}$ is maximal if, for all $\rho' \in \Sigma$, we have that $\rho \ll \rho'$ implies $\rho' \in [\rho]_{\approx_{\Sigma}}$. If $\Sigma$ is fully oriented and $[\rho]_{\approx_{\Sigma}}$ is maximal, then $\Sigma_1 := \Sigma \setminus [\rho]_{\approx_{\Sigma}}$ and $\Sigma_2 := [\rho]_{\approx_{\Sigma}}$ are such that $\Sigma_1 \triangleright \Sigma_2$, and thus $\Sigma_1 \triangleright \Sigma$. Moreover, $\Sigma_1$ again is fully oriented. Thus, one can apply Theorem 16 and Lemma 18 (2) to obtain the required rewriting.

**Theorem 21.** (NEMODEQ $\subseteq$ fully oriented Datalog). For every NEMODEQ $\Sigma$, the rule set $\text{datalog}(\Sigma)$ of Definition 7 is fully oriented. Moreover, for every NEMODEQ-rewritable set $\Sigma$ of TGDs, there is a fully oriented set of Datalog rules $\Sigma'$ such that:

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

Proof. The first part of the claim follows by induction over the degree $d$ of $\Sigma$. The case of $d = 1$ follows by observing that the rules with head predicate $p_{\Sigma}$ in $\text{datalog}(\Sigma)$ cannot be $<$-smaller than any other rule, and thus form a maximal $j$-oriented (for any position $j$ in $p_{\Sigma}$) subset of $\text{datalog}(\Sigma)$. Likewise, the set of rules with head of the form $\hat{U}(y, z)$ is clearly 1-oriented. To show the claim for $d > 1$, we observe that no rule in $\text{datalog}(\Sigma)$ is $<$-smaller than any rule in $\text{datalog}(\Sigma')$ for some subquery $\Sigma'$ of $\Sigma$. The claim follows by induction.

For the second part of the claim, let $Q_p[y]$ be the CQ $p(y)$ for each predicate $p$ that is not of the form $q_p$. There is a rewriting $\Sigma_{Q_p}[y]$ for $Q_p[y]$ and $\Sigma$. By Lemma 14, we have that $\Sigma \models \forall y . \Sigma_{Q_p}[y] \iff Q_p[y]$. From Theorem 6 follows $\models \forall y . \Sigma_{Q_p}[y] \iff \text{datalog}(\Sigma_{Q_p}) \rightarrow \text{datalog}(\Sigma_{Q_p})$. Therefore, $\Sigma \models \forall y . \Sigma_{Q_p}[y] \iff \text{datalog}(\Sigma_{Q_p}) \rightarrow \text{datalog}(\Sigma_{Q_p})$.

Using arguments as in the proof of Theorem 16, we find that $D \models \forall y . (\Sigma \rightarrow Q_p[y]) \leftrightarrow \text{datalog}(\Sigma_{Q_p}) \rightarrow \text{datalog}(\Sigma_{Q_p})$ whenever $D \cup \Sigma$ is consistent. To cover the case that $D \cup \Sigma$ is inconsistent, let $\Sigma_1$ be a MODEQ without free variables such that, for all databases $D'$, $D' \cup \Sigma$ is inconsistent iff $D' \models \Sigma_1$. To find such a $\Sigma_1$, consider $\Sigma_p$ for a predicate $p$ that does not occur in $\Sigma$ and delete from $\text{datalog}(\Sigma_p)$ all rules that use the predicate $p$ (these rules check for occurrences of $p$ in the input database). $\Sigma_1$ can easily be obtained from this.

By the first part of the claim, $\text{datalog}(\Sigma_p)$ and $\text{datalog}(\Sigma_{1})$ are fully oriented. We set $q_p := \text{datalog}(\Sigma_{Q_p})$. Let $\Sigma_1$ be the fully oriented rule set obtained from $\text{datalog}(\Sigma_{1})$ by replacing each rule $\forall x . \phi \rightarrow \text{that has an empty head by new rules $\forall x . y . \phi \rightarrow q_p(x)$ for each of the predicates $q_p$ for $x$ is a list of fresh variables of the appropriate length. Thus, $\Sigma_1$ entails all possible facts over predicates $q_p$ from $D$ whenever $D \cup \Sigma$ is inconsistent.

Now we can set $\Sigma' := \Sigma_1 \cup \bigcup_j \text{datalog}(\Sigma_{Q_p})$ where we assume w.l.o.g. that any two $\text{datalog}(\Sigma_{Q_p})$ and $\text{datalog}(\Sigma_{Q_p'})$ use mutually disjoint sets of head predicates. It is easy to verify that the claim is satisfied for this choice.