Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Monotone Rewritability and the Analysis of Queries, Views, and Rules

Michael Benedikt[1] , **Stanislav Kikot** , **Johannes Marti** , **Piotr Ostropolski-Nalewaja**[2]
[1]University of Oxford
[2]University of Wrocław, Poland; TU Dresden, Germany

Abstract

We study the interaction of views, queries, and background knowledge in the form of existential rules. The motivating questions concern monotonic determinacy of a query using views w.r.t. rules, which refers to the ability to recover the query answer from the views via a monotone function. We study the decidability of monotonic determinacy, and compare with variations that require the "recovery function" to be in a well-known monotone query language, such as conjunctive queries or Datalog. Surprisingly, we find that even in the presence of basic existential rules, the borderline between well-behaved and badly-behaved answerability differs radically from the unconstrained case. In order to understand this boundary, we require new results concerning entailment problems involving views and rules.

1 Introduction

Views are a means to define an interface to a datasource. In the context of relational databases, views allow one to associate a complex transformation – a view definition – to a new relation symbol, indicating that the symbol will stand for the output of the transformation. Given a set of views $\mathbf{V}$ and an instance $\mathcal{I}$ of the original schema, an external user will have access to the *view image of* $\mathcal{I}$, $\mathbf{V}(\mathcal{I})$, in which each view symbol is interpreted by evaluating the corresponding definition on $\mathcal{I}$. Views have many uses, including privacy and query optimization (Afrati and Chirkova 2019).

Our work is motivated by the question of rewriting queries using views. We have a set of views $\mathbf{V}$ defined over some "base schema", a query Q over the same schema, and we want to *rewrite* Q *using* $\mathbf{V}$: find a function R on the view schema such that applying R to the output of the views definitions will implement Q. We will be interested in the setting where Q and the view definitions are *monotone queries* – defined in languages that do not have negation or difference. Conjunctive queries (CQs), their unions (UCQs), and Datalog are query languages that define only monotone queries. In this case, it is natural to desire a rewriting that is a monotone function of the view images. We write $\mathbf{V}(\mathcal{I}) \subseteq \mathbf{V}(\mathcal{I}')$ if for every view definition in $\mathbf{V}$, its evaluation on $\mathcal{I}$ is contained in its evaluation on $\mathcal{I}'$. Then Q is *monotonically determined over* $\mathbf{V}$ (Perez 2011; Nash, Segoufin, and Vianu 2010; Calvanese et al. 2007) if for every $\mathcal{I}$, $\mathcal{I}'$ with $\mathbf{V}(\mathcal{I}) \subseteq \mathbf{V}(\mathcal{I}')$, $\mathsf{Output}(Q, \mathcal{I}) \subseteq \mathsf{Output}(Q, \mathcal{I}')$. A query Q may not have any rewriting using the views $\mathbf{V}$ over all datasources, but it may be well-behaved for the data of interest to an application. Here we will consider datasource restrictions given by standard classes of knowledge in the form of *existential rules*:[1] rules with existential quantifiers in the head. Given a set of views $\mathbf{V} = V_1 \dots V_k$, query Q, and rules Σ, we say that Q *is monotonically determined over* $\mathbf{V}$ *w.r.t.* Σ if for $\mathcal{I}, \mathcal{I}'$ satisfying Σ, $\mathbf{V}(\mathcal{I}) \subseteq \mathbf{V}(\mathcal{I}')$ implies $\mathsf{Output}(Q, \mathcal{I}) \subseteq \mathsf{Output}(Q, \mathcal{I}')$.

We say a query R over the view schema is a *rewriting* of Q with respect to Σ if for every $\mathcal{I}$ satisfying Σ, $R(\mathbf{V}(\mathcal{I})) = \mathsf{Output}(Q, \mathcal{I})$ where $\mathbf{V}(\mathcal{I})$ is the view image of $\mathcal{I}$.

Example 1. *Suppose our base signature has binary relations R and S, and let Q be $\exists x\ R(x, x)$. Consider the view $V(x, y) := R(x, y) \vee S(x, y)$. It is easy to see that Q is not rewritable over V, monotonically or otherwise: given a pair in V, we cannot tell if it is in R or in S. But suppose we know that our schema satisfies the rule $\forall x\ (S(x, x) \rightarrow R(x, x))$. Then Q has a simple rewriting $\exists x\ V(x, x)$.*

There are two natural questions about monotonic determinacy: 1. *Decidability*: Fixing languages for specifying queries, views, and rules, can we decide monotonic determinacy? 2. *Required expressiveness of rewritings*: can we say that when Q is monotonically determined by $\mathbf{V}$ w.r.t. Σ there is necessarily a rewriting in a restricted monotone language (e.g. CQ, UCQ, Datalog)?

The monotonicity requirement is motivated by the fact that when we begin with a monotone query, we expect a monotone rewriting, and also by the need to make the two questions above manageable. If we do not impose that a rewriting is monotone, it is known that the behavior of even very simple queries and views – CQs– is badly-behaved, even in the absence of rules. We cannot decide whether a CQ query has a rewriting over CQ views, and in cases where such a rewriting exists we can say basically nothing about how complex it may need to be (Gogacz and Marcinkowski 2015; Gogacz and Marcinkowski 2016). In contrast, it is known that monotonic determinacy behaves very well for CQ queries and views: decidable in NP (Nash, Segoufin,

[1]Existential rules are also called *tuple-generating dependencies* (Abiteboul, Hull, and Vianu 1995), *conceptual graph rules* (Salvat and Mugnier 1996), Datalog$^{\pm}$ (Lukasiewicz, Calì, and Gottlob 2012), and $\forall\exists$*-rules* (Baget et al. 2011).

and Vianu 2010; Levy et al. 1995). And when rewritings exist, they can be taken to be CQs (Nash, Segoufin, and Vianu 2010). These results have been shown to extend to many classes of rules (Benedikt et al. 2016).

Monotone determinacy is well understood for first-order queries and views, in the absence of rules. It is decidable in the case of UCQ views and queries, and whenever one has monotonic determinacy, one has a rewriting that is a UCQ (Benedikt et al. 2016). Prior work also investigated the situation when views and queries are *monotone and recursive*, either as *regular path queries* or *Datalog* (Calvanese et al. 2002; Calvanese et al. 2007; Francis, Segoufin, and Sirangelo 2015; Benedikt et al. 2023). Recall that Datalog is a standard language for defining recursive monotone queries. For example, the Datalog query Q_R consisting of the following two rules computes the pairs (x, y) where x reaches y via edges in a binary relation R:

$$\text{REACH}(x, y) := R(x, y)$$
$$\text{REACH}(x, y) := R(x, z), \text{REACH}(z, y)$$

Let us give a rough summary of the situation for Datalog, based on the most recent work (Benedikt et al. 2023). Monotonic determinacy does not behave well for general Datalog queries and views – it is undecidable, and when it holds one may need arbitrarily complex rewriting. At a high level, two well-behaved paradigms for monotonic determinacy have been identified: 1) *non-recursive queries (i.e. UCQ) and general Datalog views*, and 2) *both views and queries guarded*. Specifically, for decidability, we have:

- *The query being non-recursive suffices for good behaviour*: The query Q being a UCQ suffices for decidability, even for general Datalog views;
- *Guarded queries and views behave well*: If the views and queries are both in the standard "guarded recursive language", frontier-guarded Datalog, monotonic determinacy is decidable, even if there is recursion in both;
- *One case of a recursive query and non-guarded views behaves well:* If the query is recursive, monotonic determinacy behaves well if the query is "very guarded"– Monadic Datalog (MDL) – and the views are CQs.
- *Recursive queries and unguarded views are a problem*: If the views are general (unguarded) UCQs and we allow recursion in the query Q (even MDL), monotonic determinacy is undecidable.

All the cases above that are well-behaved for decidability are also well-behaved for expressiveness of rewritings: whenever monotonic determinacy holds there is a rewriting in Datalog. But there are additional "tame" cases for rewritability; *when a Datalog query is monotonically determined in terms of CQ views or guarded views, there is a rewriting in Datalog*. That is, for views that are CQs or in guarded Datalog, we do not need to restrict the Datalog query Q to be sure that monotonic determinacy is witnessed by a Datalog rewriting. Figures 1 and 2 provide a detailed look at the case without any knowledge in the form of rules. In the table Und. means undecidable, and "n. n. Datalog" means that rewritings are not necessarily in Datalog.

We consider how this changes with background knowledge. Clearly, for "arbitrary knowledge" in first-order logic, nothing interesting can be said about decidability or about the necessary language for rewritings. We will thus focus on "tame existential rules" such as those in the Datalog$^{\pm}$ family (Calì et al. 2011), frontier-guarded TGDs, which are known to behave well for standard decidability questions. In the presence of tame existential rules, a number of new complications arise. First, we need to distinguish two notions of monotonic determinacy: *finite monotonic determinacy*, in which the rewriting must exist only for finite databases, and *unrestricted monotonic determinacy*, in which the rewriting holds for all relational structures, finite or infinite. In the case of Datalog without rules, these two notions coincide (Benedikt et al. 2023). But even in the presence of very simple rules, they differ.

Example 2. *Let our schema consist of a binary relation R, along with a rule:* $\forall xy\ R(x, y) \rightarrow \exists z\ R(y, z)$

This is a Linear TGD*: a universally quantified implication with a single atom in the hypothesis. It is even a* unary inclusion dependency *(UID): there are no repeated variables, and only one variable occurs on both sides of the implication. Such rules are extremely well-behaved. For example, the implication problem between them is known to be decidable in PSPACE (Abiteboul, Hull, and Vianu 1995). Observe that in a* finite *database satisfying the rule, there must be a cycle of R edges.*

Consider the Datalog program $Q = Q_R \cup \{\text{GOAL} := \text{REACH}(x, x)\}$*, where Q_R is defined above, and* GOAL *the goal predicate. Then Q is a Boolean query checking for the existence of an R cycle. Suppose also that we have only the Boolean view $V() := \exists xy\ R(x, y)$. Clearly, we cannot answer Q using the views over all instances. But over finite databases, Q is equivalent to $V()$, so it is monotonically determined over finite instances.*

In this work, we deal mainly with *unrestricted monotonic determinacy*, admittedly because it is simpler to analyze. But even for the unrestricted case, our proofs will involve interplay between finite and infinite. And we will sometimes be able to infer that the finite and unrestricted cases agree. We show that the case where both views and queries are guarded is still well-behaved for decidability in the presence of guarded rules. But the case of non-guarded views is significantly different. Without rules, non-recursiveness can substitute for guardedness. But we show that *even when both the query and views are non-recursive, and even when the rules are in a well-behaved class, monotonic determinacy is undecidable*. We show that decidability can be regained by restricting the view and query to be CQs, and also imposing that the rules are extremely simple: linear TGDs, a case analogous to SQL referential constraints.

For expressiveness of rewritings, the most significant differences introduced by rules come for guarded views and queries. We show that for monotonically determined guarded queries and views, we obtain rewritings in fixpoint logic, and thus in polynomial time. In certain cases we can conclude rewritability in Datalog.

Contributions, and How to Read this Paper. Our first contribution is a set of tools for reasoning with queries, views, and rules:

Query \ Views	CQ	MDL, FGDL	UCQ	DL
CQ, UCQ	NP	2EXP	CONEXP	2EXP
MDL	in 3EXP		Undecidable	
FGDL	Open			
DL				

Figure 1: Decidability/Complexity without rules

Query \ Views	CQ	MDL, FGDL	UCQ	DL
CQ	CQ			
UCQ	UCQ			
MDL	FGDL	MDL	n. n. Datalog	
FGDL		DL		
DL				

Figure 2: Rewritability without rules

- We extend the "forward-backward" method for obtaining rewritings to the presence of rules. Briefly, *we show that sometimes one can use automata to generate Datalog rewritings for monotonically determinacy queries*.
- We provide results on *certain-answer rewritability* of Datalog queries with rules (Theorem 3, Proposition 4), and on *finite controllability of recursive queries with rules* (Theorem 4). *We recommend that readers interested in traditional KR reasoning problems start with these results*.
- We present a new kind of "compactness property" for entailments involving Datalog queries, rules, and views. This is more technical, and arguably more specific to the setting of view rewriting. It states that when an infinite view instance entails that a query holds on the base instance it is derived from, then the same holds for some finite subinstance. This is crucial to rewritability and non-rewritability results.

While we apply these tools to get positive results about decidability and rewritings for monotonic determinacy, we believe that all of them – but particularly the results in the second item – are of broader interest.

Our second major contribution is a set of surprising undecidability results – strongly contrasting with the case without background knowledge. For example, Theorem 16 implies that monotonic determinacy and also CQ rewritability for CQ queries and CQ views is undecidable even for quite simple rules: combinations of frontier-one and linear. While Theorem 18 shows that if the views have disjunction, there is undecidability even with single attribute referential constraints. Briefly, *the presence of simple rules implies that CQ rewriting and determinacy problems become undecidable. The reader mostly interested in results about rewriting with views is recommended to start with these results (Section 5) which use direct encodings, not the tools above*.

Organization. Section 2 contains preliminaries. Section 3 presents key tools for our positive results. Section 4 applies the tools to get positive results on decidability and rewritability, while Section 5 presents the surprising undecidability results. The paper ends with a summary in Section 6. Many proofs, along with some auxiliary results, are deferred to the full version (Benedikt et al. 2024).

2 Preliminaries

We assume the usual notion of relational schema: a finite set of relations, with each relation associated with a number its *arity*. For predicate R of arity n, an R-fact is of the form $R(c_1 \dots c_n)$. A *database instance* (or simply *instance*) is a set of facts. The *active domain* of an instance, $\text{ADOM}(\mathcal{I})$, is the set of elements that occur in some fact. A *query* of arity n over schema $\mathbf{S}$ is a function from instances over $\mathbf{S}$ to relations of arity n. A *Boolean query* is a query of arity 0. The output of a query Q on instance $\mathcal{I}$ is denoted as $\mathsf{Output}(Q, \mathcal{I})$. We also use the standard notations $\mathcal{I} \models Q(\boldsymbol{c})$ or $\mathcal{I}, \boldsymbol{c} \models Q$ to indicate that $\boldsymbol{c}$ is in the output of Q on input $\mathcal{I}$. A query is *monotone* if $\mathcal{I} \subseteq \mathcal{I}'$ implies that $\mathsf{Output}(Q, \mathcal{I}) \subseteq \mathsf{Output}(Q, \mathcal{I}')$. A *homomorphism* from instance $\mathcal{I}$ to instance $\mathcal{I}'$ is a mapping h such that $R(c_1 \dots c_n) \in \mathcal{I}$ implies $R(h(c_1) \dots h(c_n)) \in \mathcal{I}'$. We assume familiarity with standard *tree automata* over finite trees of fixed branching depth. A few results use automata over infinite trees (Thomas 1997). *Büchi tree automata* have the same syntactic form as regular tree automata: a finite set of states, subsets that are initial and accepting, and a transition relation. An automaton accepts an infinite tree if there is a run that assigns states to each vertex of the tree, obeying the transition relation, and where every path hits an accepting state infinitely often.

CQs and Datalog. We assume familiarity with FO logic, including the notion of free and bound variable. A *conjunctive query* (CQ) is a formula of the form $Q(\boldsymbol{x}) = \exists \boldsymbol{y}\, \phi(\boldsymbol{x}, \boldsymbol{y})$, where $\phi(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of relational atoms. Given any CQ Q, its *canonical database*, denoted $\text{CANONDB}(Q)$, is the instance formed by treating each atom of Q as a fact. The output of a CQ Q on some instance $\mathcal{I}$ is the set of all tuples $\boldsymbol{t}$ such that there is homomorphism $h : \text{CANONDB}(Q) \to \mathcal{I}$ satisfying $\boldsymbol{t} = h(\boldsymbol{x})$. A *union of conjunctive queries* UCQ is a disjunction of CQs with the same free variables.

Datalog is a language for defining queries over a relational schema $\mathbf{S}$. Datalog rules are of the form: $P(\boldsymbol{x}) := \phi(\boldsymbol{x}, \boldsymbol{y})$ where $P(\boldsymbol{x})$ is an atom s.t. $P \notin \mathbf{S}$, and ϕ is a conjunction of atoms, with $\boldsymbol{y}$ implicitly existentially quantified. The left side of the rule is the *head*, while the right side is the *body* of the rule. In a set of rules, the relation symbols in rule heads are called *intensional database predicates* (IDBs), while relations in $\mathbf{S}$ are called *extensional relations* (EDBs). A *Datalog program* is a finite collection of rules. For an instance $\mathcal{I}$ and Datalog program Π, we let $\mathsf{FPEval}(\Pi, \mathcal{I})$ denote the $\supseteq$-minimal IDB-extension of the input $\mathcal{I}$ which satisfies Π treated as a set of universal FO implications. A *Datalog query* $Q = (\Pi, \text{GOAL})$ is a Datalog program Π together with a distinguished intensional *goal relation* GOAL of arity $k \geq 0$. The *output* of Datalog query Q on an instance $\mathcal{I}$ (denoted as $\mathsf{Output}(Q, \mathcal{I})$ or simply $Q(\mathcal{I})$) is the set $\{\boldsymbol{c} \mid \text{GOAL}(\boldsymbol{c}) \in \mathsf{FPEval}(\Pi, \mathcal{I})\}$. *Monadic Datalog* (MDL) is the fragment of Datalog where all IDB predicates are unary. *Frontier-guarded Datalog* (FGDL) requires that in each rule $P(\boldsymbol{x}) := \phi(\boldsymbol{x}, \boldsymbol{y})$ there exists an EDB atom in ϕ containing $\boldsymbol{x}$. Frontier-guarded Datalog does not contain MDL; for example, in an MDL program we can have a rule $I_1(x) := I_2(x)$, where I_1 and I_2 are both intensional. However every MDL program can be rewritten in FGDL (Benedikt et al. 2023), and thus we declare, as a convention, that any MDL program is FGDL.

When Datalog query Q holds for tuple $\boldsymbol{t}$ in instance $\mathcal{I}$,

this means one of an infinite sequence of CQ queries Q_n holds. Here Q_n is obtained by *unfolding the intensional predicates by their rule bodies some number of times*. Such a query is called a *CQ approximation of* Q. Datalog can also be seen as a subset of Least Fixpoint Logic (LFP), the extension of first-order logic with a least fixed point operator construct: if $\phi(x_1 \dots x_K, X)$ is a formula with free variables $x_1 \dots x_k$ and an additional k-ary second order free variable X, $\mu_{X,\boldsymbol{x}}\phi(y_1 \dots y_k)$ is a formula in which X is not free, but $\boldsymbol{y}$ are free. In this work we will also consider an extension of Datalog where we have both the least fixed point operator, the dual greatest fixpoint operator ν, but no negation or universal quantification. This logic, denoted POSLFP, can still only express monotone queries, but it can express properties beyond Datalog: e.g., the POSLFP formula $\nu_{P,x}.[A(x) \wedge (\exists y\ R(x,y) \wedge P(y))](z)$ holds of element z when it is in a unary predicate A and has paths to an A node of finite length.

Existential rules. Semantic relationships between relations can be described using *existential rules* - also referred to as *Tuple Generating Dependencies* (TGDs) - that are logical sentences of the form $\forall \boldsymbol{x}\ \lambda(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}\ \rho(\boldsymbol{x}, \boldsymbol{y})$, where $\lambda(\boldsymbol{x})$ and $\rho(\boldsymbol{x}, \boldsymbol{y})$ are conjunctions of relational atoms whose free variables are contained in $\boldsymbol{x}$, and $\boldsymbol{x} \cup \boldsymbol{y}$ correspondingly. The left-hand side of a TGD (i.e., the conjunction $\lambda(\boldsymbol{x})$) is the *body* of the dependency, and the right-hand side is the *head*. By abuse of notation, we often treat heads and bodies as sets of atoms, and we commonly omit the leading universal quantifiers. The variables that appear in both the head and the body are the *frontier* of the rule, and are also said to be the *exported variables*.

Let $\mathcal{I}$ be an instance and let τ be a TGD. The notion of τ *holding* in $\mathcal{I}$ (or $\mathcal{I}$ *satisfying* τ, written $\mathcal{I} \models \tau$) is the usual one in first-order logic.

Given CQs Q_1 and Q_2 along with dependencies Σ, we say Q_1 is contained in Q_2 relative to Σ if for every instance $\mathcal{I}$ satisfying Σ, $\mathsf{Output}(Q_1, \mathcal{I}) \subseteq \mathsf{Output}(Q_2, \mathcal{I})$. We also write $\Sigma \wedge Q_1 \models Q_2$ in this case. Given a set of facts D, Boolean CQ Q, and dependencies Σ, we say D and Σ *entails* Q if Q holds in every instance containing D and satisfying Σ. We also write $\Sigma \wedge D \models Q$. We can similarly talk about "finite entailment", where "every instance containing D" is replaced by "every finite instance containing D". For a class of queries and existential rules, we say *entailment is finitely controllable* if entailment agrees with finite entailment for each finite instance D and each query and rules in the class.

Special classes of rule. A rule is *linear* if the body has a single atom, and *full* if there are no existential quantifiers in the head. It is *frontier-guarded* (FGTGD) if there is a body atom that contains all exported variables. It is *frontier-one* (FR-1) if there is only one variable in the frontier. A FR-1 linear TGD with no constants or repeated variables is a *Unary Inclusion Dependency* (UID). A set of rules Σ is *Source-to-Target* if for every $\rho, \rho' \in \Sigma$ the head-predicates of ρ do not appear in the body of ρ'.

The chase. In certain arguments we use the characterization of logical entailment between CQs in the presence of existential rules in terms of the chase procedure (Maier, Mendelzon, and Sagiv 1979; Fagin et al. 2005). The chase modifies an instance by a sequence of *chase steps* until all dependencies are satisfied. Let $\mathcal{I}$ be an instance, and consider a TGD $\tau = \forall \boldsymbol{x}.\lambda \rightarrow \exists \boldsymbol{y}.\rho$. Let h be a *trigger* – a homomorphism from λ into $\mathcal{I}$. Performing a chase step for τ and h to $\mathcal{I}$ extends $\mathcal{I}$ with each fact of the conjunction $h'(\rho(\boldsymbol{x}, \boldsymbol{y}))$, where h' is a substitution such that $h'(x_i) = h(x_i)$ for each variable $x_i \in \boldsymbol{x}$, and $h'(y_j)$, for each $y_j \in \boldsymbol{y}$, is a fresh labeled null that does not occur in $\mathcal{I}$. For Σ a set of TGDs and $\mathcal{I}$ an instance, we use $\text{CHASE}_\Sigma(\mathcal{I})$ to denote any instance (possibly infinite) formed from $\mathcal{I}$ by iteratively applying chase steps, where Σ holds. We say that $\text{CHASE}_\Sigma(\mathcal{I})$ *is finite* if we can perform finitely many chase steps and obtain an instance satisfying Σ. We say that Σ *has terminating chase* if $\text{CHASE}_\Sigma(\mathcal{I})$ is finite for every finite $\mathcal{I}$.

Views and rewritability. A *view* over some relational schema $\mathbf{S}$ is a tuple (V, Q_V) where V is a view relation and Q_V is an associated query over $\mathbf{S}$ whose arity matches that of V. Q_V is referred to as the *definition* of view V. By $\mathbf{V}$ we denote a collection of views over a schema $\mathbf{S}$. We sometimes refer to the vocabulary of the definitions Q_V as the *base schema* for $\mathbf{V}$, denoting it as Σ_B, while the predicates components V are referred to as the *view schema*, denoted $\Sigma_\mathbf{V}$. For an instance $\mathcal{I}$ and set of views $\mathbf{V} = \{(V, Q_V) \mid V \in \Sigma_\mathbf{V}\}$, the *view image* of $\mathcal{I}$, denoted by $\mathbf{V}(\mathcal{I})$, is the instance over $\Sigma_\mathbf{V}$ where each view predicate $V \in \Sigma_\mathbf{V}$ is interpreted by $\mathsf{Output}(Q_V, \mathcal{I})$. Recall from the introduction that query Q is monotonically determined over views $\mathbf{V}$ relative to rules Σ if for each $\mathcal{I}, \mathcal{I}'$ satisfying Σ with $\mathbf{V}(\mathcal{I}) \subseteq \mathbf{V}(\mathcal{I}')$, $Q(\mathcal{I}) \subseteq Q(\mathcal{I}')$. Given views $\mathbf{V}$, rules Σ and a query Q, a query R over the view schema $\Sigma_\mathbf{V}$ is a *rewriting* of Q with respect to $\mathbf{V}$ and Σ if: for each $\mathcal{I}$ over $\mathbf{S}$ satisfying Σ, the output of R on $\mathbf{V}(\mathcal{I})$ is the same as the output of Q on $\mathcal{I}$. A rewriting that can be specified in a particular language L (e.g. Datalog, CQs) is an L-rewriting of Q w.r.t. $\mathbf{V}$ and Σ, and if this exists we say Q is *L-rewritable* over $\mathbf{V}, \Sigma$. We drop $\mathbf{V}$ and/or Σ from the notation when it is clear from context.

It is clear that if Q has a rewriting in a language that defines only monotone queries, like Datalog, then Q must be monotonically determined. We will be concerned with the converse to this question. The main questions we will consider, fixing languages L_Q, $L_\mathbf{V}$, L_Σ for the queries, views, and rules (e.g. Datalog, fragments of Datalog for the first two, special classes of existential rules for the third) are:

- can we decide whether a Q in L_Q is monotonically determined over $\mathbf{V}, \Sigma$ in $L_\mathbf{V}, L_\Sigma$?
- fixing another language L for rewritings, if Q is monotonically determined over $\mathbf{V}, \Sigma$, does it necessarily have a rewriting in L?

Finite and unrestricted variants. We have noted in the introduction that there are variants of our definitions of monotonic determinacy and rewritability depending on whether only finite instances are considered, or all instances. In the definitions above, we make unrestricted instances the default. We say that query Q is monotonically determined over views $\mathbf{V}, \Sigma$ *over finite instances* if in the definition "for any each instances $\mathcal{I}, \mathcal{I}'$ satisfying ..." is replaced by "for each finite instances $\mathcal{I}, \mathcal{I}'$ satisfying ...". We similarly arrive at the notion of a query R being a rewriting over $\mathbf{V}, \Sigma$

for finite instances by changing "for each $\mathcal{I}$ over $\mathbf{S}$" in the definition of rewriting to "for each finite $\mathcal{I}$ over $\mathbf{S}$". If the finite and unrestricted versions coincide, we say that *monotonic determinacy is finitely controllable* for a class of view, queries, and rules. In (Benedikt et al. 2023) finite controllability was observed in the absence of rules, while Example 2 shows that it fails even for simple rules.

Monotonic determinacy characterized using approximations and the chase. Algorithm 1 gives an abstract procedure for checking monotonic determinacy of Boolean Datalog query Q over Datalog views $\mathbf{V}$ with respect to rules Σ. We universally choose an approximation Q_n of Q, and then chase the canonical database of this query with Σ. One can think of each of the results as a generic instance satisfying Q and Σ. We let $\mathcal{J}_n$ be the view image of such a database: thus $\mathcal{J}_n$ is a set of view facts. We then take each fact $\mathbf{V}_i(\boldsymbol{c})$ in $\mathcal{J}_n$, where $\mathbf{V}_i$ is a view predicate, and *non-deterministically choose witnesses facts for it*. For each $\mathbf{V}_i(\boldsymbol{c})$ we choose an approximation for the query $Q_{\mathbf{V}_i}$ associated with $\mathbf{V}_i$, and then take the canonical database of this with $\boldsymbol{c}$ substituted in for the free variables. For an instance $\mathcal{J}$ of the view schema, we let $\textsc{BackV}_{\mathbf{V}}(\mathcal{J})$ be the instances of the base schema that results from this process. In the case that the views are CQs, we can assume there is only one instance in $\textsc{BackV}_{\mathbf{V}}(\mathcal{J})$, namely the chase of $\mathcal{J}$ with rules of the form $\forall \boldsymbol{x}\, [V_i(\boldsymbol{x}) \rightarrow Q_{\mathbf{V}_i}(\boldsymbol{x})]$. When views are UCQs or Datalog, $\textsc{BackV}_{\mathbf{V}}(\mathcal{J})$ can be thought of as the result of a "disjunctive chase". Next, we chase instances in $\textsc{BackV}_{\mathbf{V}}(\mathcal{J}_n)$ using Σ – thus chasing a second time. We have described a non-deterministic process that generates instances $\mathcal{I}'$ in the base schema, where each $\mathcal{I}'$ satisfies Σ, and has a view image containing one of the $\mathcal{J}_n$. That is, each of these "test instances" has a view image containing the view image of an instance satisfying Q and Σ, and thus monotonic determinacy states they should satisfy Q. The process is non-deterministic because we have guesses for the initial approximation Q_n, and guesses for the approximation witnessing each view fact. If each of the databases $\mathcal{I}'$ resulting from this process satisfies the original query Q, monotonic determinacy holds.

Algorithm 1 Process for checking monotonic determinacy.

$\textsc{MonDet}(Q, \mathbf{V}, \Sigma)$**:**
1: **for** Q_n approximation of Q **do** ▷ unfold the query
2: $C_n := \textsc{Chase}_\Sigma(\textsc{CanonDB}(Q_n))$ ▷ Chase an unfolding
3: $\mathcal{J}_n := \mathbf{V}(C_n)$ ▷ Apply views
4: **for** $Q'_{m,n} \in \textsc{BackV}_{\mathbf{V}}(\mathcal{J}_n)$ **do** ▷ Guess a witness for each view fact
5: $C'_{m,n} := \textsc{Chase}_\Sigma(Q'_{m,n})$ ▷ Chase again
6: If $C'_{m,n} \not\models Q$ ▷ Check if Q holds
7: **return false**
8: **return true**

The fact that this process captures monotonic determinacy with respect to existential rules is straightforward (see also Lemma 5.4 in (Benedikt et al. 2023))

Proposition 1. *Q is monotonically determined over $\mathbf{V}$ w.r.t. Σ if and only if the process of Algorithm 1 returns true.*

The "process" above is *not* an algorithm, even in the absence of rules, since there are infinitely many choices for Q_n and infinitely many choices for the approximations to substitute in Step 4. With rules, there is also the problem that the chase may be infinite. Nevertheless, all of our results on monotonic determinacy will make use of this characterization. Intuitively, one way to analyze monotonic determinacy is by *moving forward* in the process: getting effective representations of the intermediate artifacts produced in each step. A second technique is to *move backward*, rewriting away steps of the process to get a simpler algorithm that does not use these steps. We will make use of both of these approaches in our results.

3 Tools for the positive results

Our undecidability results will be direct reductions, and thus do not require much prior machinery. But as mentioned in the introduction, for our positive results we develop some techniques for analyzing rules, views, and queries involving Datalog, which we then apply to the process of Figure 1.

Review of the forward-backward approach and the bounded tree-width view image property. We now review an idea from (Benedikt et al. 2023): we can "capture the intermediate artifacts in Figure 1 using tree automata". And we can sometimes move from tree automata in the view signature to Datalog over the views (backward mapping).

For a number k a *tree decomposition of adjusted width k* for an instance $\mathcal{I}$ is a pair $\mathcal{TD} = (\tau, \lambda)$ consisting of a rooted directed tree $\tau = (\textsc{vertices}, E)$, either finite or countably infinite, and a map λ associating a tuple of distinct elements $\lambda(v)$ of length at most k (called a *bag*) to each vertex v in $\textsc{vertices}$ such that the following conditions hold: – for every atom $R(\boldsymbol{c})$ in $\mathcal{I}$, there is a vertex $v \in \textsc{vertices}$ with $\boldsymbol{c} \subseteq \lambda(v)$; – for every element c in $\mathcal{I}$, the subgraph of τ induced over the set $\{\, v \in \textsc{vertices} \mid c \in \lambda(v) \,\}$ is connected. A tree decomposition $\mathcal{TD}$ of an instance $\mathcal{I}$ can be associated with a labelled tree T where labels describe the facts holding on the elements associated with a node. We use a standard encoding – see the full version (Benedikt et al. 2024). Any such tree T will be a *k tree code* of the instance $\mathcal{I}$. Given a tree code T, we denote the instance it encodes with $\mathfrak{D}(T)$.

Above we abuse notation slightly by re-using $\lambda(v)$ to indicate the underlying set of elements, as well as the tuple. The *adjusted treewidth* of an instance $\mathcal{I}$, $\textsc{tw}^+(\mathcal{I})$, is the minimum adjusted width of a tree decomposition of $\mathcal{I}$. For a tree decomposition $\mathcal{TD}$ of data instance $\mathcal{I}$ let $\textsc{tspan}(\mathcal{TD})$ (the *treespan* of the decomposition) be the maximum over elements e of $\mathcal{I}$ of the number of bags containing e.

A counterexample to monotonic determinacy consists of instances $\mathcal{I}, \mathcal{I}'$ satisfying Σ such that $\mathbf{V}(\mathcal{I}) \subseteq \mathbf{V}(\mathcal{I}')$, $\mathcal{I} \models Q$, and $\mathcal{I}' \models \neg Q$. We will consider such a pair as a single instance in the signature with two copies of the base schema, one interpreted as in $\mathcal{I}$ while the other is interpreted as in $\mathcal{I}'$, along with one copy of the view predicates, interpreted as in $\mathbf{V}(\mathcal{I})$. A *tree code of a counterexample to monotonic determinacy* is a tree code for the instance formed from

a counterexample in the vocabulary above. The following proposition, an application of Courcelle's theorem (Courcelle 1991), states that for any fixed k, we can recognize counterexamples that are of low treewidth, using automata.

Proposition 2. *[Forward Mapping] For each Q in FGDL, $\mathbf{V}$ in FGDL or FO, Σ existential rules, and each k there is a tree automaton that accepts all finite k tree codes of counterexamples to monotonic determinacy of $Q, \mathbf{V}, \Sigma$. There is a a Büchi automaton over infinite trees that holds exactly when there is an arbitrary (possibly infinite) k tree code of such an instance.*

Of course, for monotonic determinacy, we are interested not just in treelike counterexamples, but arbitrary counterexamples. However, in some cases, low treewidth is enough. A triple $(Q, \mathbf{V}, \Sigma)$ has the *bounded treewidth view image property* (BTVIP) if we can compute a k such that for every approximation Q_n and some chase C of $\textsc{CanonDB}(Q_n)$ under Σ it holds that $\textsc{tw}^+(\mathbf{V}(C)) \leq k$. If we can choose a finite C, we say $(Q, \mathbf{V}, \Sigma)$ has the *finite bounded treewidth view image property* (FBTVIP).

Proposition 3. *If Σ are* FGTGD*s, $(Q, \mathbf{V}, \Sigma)$ has the BTVIP (resp. FBTVIP), $\mathbf{V}$ is in Datalog, then there is k, computable from $(Q, \mathbf{V}, \Sigma)$, such that whenever monotonic determinacy fails, there is some counterexample (resp finite counterexample) of treewidth k.*

In particular, the two propositions tell us that, when the hypotheses of both propositions apply, it suffices to check that the automaton from Proposition 2 is nonempty. This will allow us to get decidability results on monotonic determinacy in the presence of the BTVIP.

For getting results on rewriting, it is useful to move backward from tree automata accepting certain codes to a Datalog query accepting their decodings. Here is one formalization of the backward mapping, a variation of (Benedikt et al. 2023, Proposition 7.1)

Theorem 1. *[Backward Mapping] Let σ be a relational signature, $k \in \omega$ and A a tree automaton over the signature for tree codes of treewidth k structures over σ. Then there is a Datalog program E_A such that for every σ-structure $\mathfrak{M}$: $\mathfrak{M} \models E_A$ iff there is a finite tree code $\mathcal{T}$ over σ accepted by A with a homomorphism from $\mathfrak{D}(\mathcal{T})$ to $\mathfrak{M}$.*

In our rewriting theorems – see Theorems 12 and 14 in Section 4 – the idea is to first apply the forward mapping of Proposition 2 to get an automaton accepting treelike counterexamples. We then project to get an automaton over codes of view instances. We apply backward mapping of Theorem 1 to get a Datalog program. We emphasize that the same process was used in (Benedikt et al. 2023) for the rewriting results in Figure 2. The main difference is that we will need a modification dealing with the fact that the treecodes involved may be infinite. This required us to extend to automata over infinite trees in Proposition 2, and in Theorem 12 it will require us to expand the rewriting language from Datalog to the larger logic POSLFP.

Certain answer rewritings. The alternative to "moving forward" in the process of Figure 1 is to go backwards, eliminating steps in the process via *certain answer rewriting results*. This will require us to obtain new results eliminating entailment steps.

For an instance $\mathcal{I}$, logical sentences Σ, and Boolean query Q, we write $\mathcal{I} \wedge \Sigma \models Q$ if Q returns true on every instance that includes $\mathcal{I}$ and satisfies Σ. This is just the usual logical entailment when $\mathcal{I}$ is considered as a conjunction of facts. We also say that Q is *certain* for $\mathcal{I}, \Sigma$. A *certain-answer rewriting* (C.A. REWRITING) of Q with respect to rules Σ is a query Q_Σ such that running Q_Σ on every $\mathcal{I}$ will tell whether Q is certain for $\mathcal{I}, \Sigma$. For a query language L, we talk about an L-C.A. REWRITING. Informally, Q has an L-C.A. REWRITING over Σ if we can check whether Q is certain w.r.t Σ with an L query.

The following is easy to derive from prior results.

Theorem 2. *UCQs have UCQ* C.A. REWRITING*s over linear TGDs (Calì, Lembo, and Rosati 2003, Thm. 3.3). UCQs have* FGDL C.A. REWRITING*s over* FGTGD*s (Bárány, Benedikt, and ten Cate 2018, Thm 5.6).*

In all cases where we write that something has a C.A. REWRITING, the generation of the rewriting is effective. We omit the precise bounds here. We can refine the argument from (Bárány, Benedikt, and ten Cate 2018) to show that for FR-1 rules, the rewriting is in MDL:

Theorem 3. *CQs have MDL* C.A. REWRITING*s over* FR-1 *TGDs.*

Less is known when Q is in Datalog. For $Q \in$ FGDL we get a result from Thm. 2 and "moving Datalog rules into the existential rules":

Proposition 4. *FGDL queries have FGDL* C.A. REWRITING*s over* FGTGD*s.*

A new finite controllability result. We now consider a subset of Datalog that is less restrictive than MDL or even FGDL. The *Extensional Gaifman graph* of a Datalog rule is the graph whose nodes are the variables in the head and whose edges connect two variables if there is an atom over an extensional relation that connects them. A query or program is *extensionally-connected* (EC) if in each rule the Extensional Gaifman graph is connected. EC Datalog subsumes FGDL and hence MDL. Recall the definition of "entailment is finite controllable" from Section 2. We can show:

Theorem 4. *For the class of* FR-1 *rules and EC-Datalog queries, entailment is finitely controllable.*

This is *the first non-trivial finite-controllability result we know of for Datalog queries with arbitrary arity*. The proof modifies an approach of "finite quotients", stemming from work in finitely controllability for description logics, e.g (Gogacz, Ibáñez-García, and Murlak 2018):

Proof. Fix the database instance $\mathcal{D}$, the set of frontier-one TGDs Σ, and an EC Datalog query Q. Let $\mathcal{C}$ denote the chase of $\mathcal{D}$ by Σ, let n denote the maximal size of any rule in both Σ and Q, and let $N = 4 \cdot n^2$. In this sketch we make some drastic simplifications to convey the idea: 1. The database instance $\mathcal{D}$ is a single unary atom. 2. The EDB relational symbols are at most binary. 3. The rules of Σ have exactly one frontier variable. 4. No atoms of the shape $E(x, x)$ appear in $\mathcal{D}, \Sigma$, and Q 5. Heads of rules of Σ are trees. We

disregard the direction of binary predicates whenever discussing graph-theoretic notions, such as distances, trees, or cycles. From the assumptions we easily see:

Proposition 5. *The chase of $\mathcal{D}$ by Σ forms a regular tree.*

Definition 1 (Unabridged)**.** *Given an instance $\mathcal{I}$ and a cycle C in $\mathcal{I}$ we say that C is* unabridged *if for every pair of elements t, t' of C the shortest path between them in $\mathcal{I}$ goes through C.*

Note that above we treat instances as undirected graphs.

Definition 2 (Trimming)**.** *Consider a tree T with a node v. The process of removing all the children of node v is referred to as* trimming *at node v. Note, that trimming at leaves is allowed, but it has no effect.*

Definition 3 (Unfolding tree)**.** *For a Datalog program P without repeated variables in rule heads, the* unfolding tree *is any tree derived from a CQ-approximation tree T of P through the process of trimming at one or more nodes of T.*

Then, an unfolding *of P is a conjunction of labels from any unfolding tree of P. Note that the resulting CQ may include IDB predicates.*

Definition 4 (Succession)**.** *We say that an unfolding tree T is a* direct successor *of an unfolding T' if T' can be obtained from T by trimming it at a node with only leaves as its children. We define* succession *as the transitive closure of direct succession. These notions naturally extend to unfoldings.*

To prove Theorem 4 we need to show that iff $\mathcal{C}$ does not entail Q then there exists a finite model M of Σ containing $\mathcal{D}$ that does not entail Q as well. We give only the construction of M here, leaving the verification for the full version.

Definition 5 (Ancestor)**.** *Given an infinite tree $\mathcal{T}$, a natural number m, and a node u of $\mathcal{T}$, we define the m-*ancestor *of u as the ancestor of u at a distance of m, if it exists; otherwise, the m-ancestor of u is the root of T..*

Definition 6 (Perspective)**.** *Given a natural number m and a node u of an infinite tree $\mathcal{T}$, we define the m-*perspective *of u as the pair $\langle T', u\rangle$ where T' is the subtree of T that is rooted at the m-parent of u. We consider m-perspectives up to isomorphism.*

Let $type(u)$, for a term u in $\mathcal{C}$, consist of two values:
1. The depth of u in $\mathcal{C}$ modulo N. 2. The N-perspective of u. Note that Proposition 5 indicates there are only a finite number of such perspectives, keeping in mind that we count only up to isomorphism. Define M as a structure that is a quotient of $\mathcal{C}$ using the "is of the same type" relation, where $type$ is defined as above.

Using an analysis of how Q can be satisfied, we can show that M witnesses finite controllability: M extends $\mathcal{D}$, satisfies Σ, and does not satisfy Q. □

Compactness of entailment. Let us go back to the forward approach for analyzing pipelines of chasing and views, such as the process of Figure 1. We start by using an automaton to represent the view images of chases of unfoldings of the query, which we can do when we have the BTVIP and some extra conditions on the views. But usually we need an automaton over infinite trees to do this, as in Proposition 2. This is unfortunate, because our backward mapping result, Theorem 12, requires an ordinary finite tree automaton to map backward into Datalog. What will help us is that we are interested in cases where the entire process succeeds – which means monotonic determinacy holds. We would like to argue that this depends on only a finite part of the view image of the chase, and later conclude that a finite tree automata suffices to capture this part. We give a general result about entailment that will be useful for those purposes.

Fix a set of views $\mathbf{V}$, rules Σ, and a query Q, and let $\mathcal{J}$ be an instance of the view schema. A $\mathbf{V}, \Sigma$-*sound realization of* $\mathcal{J}$ is an instance of the base schema satisfying Σ whose view image includes $\mathcal{J}$. An instance $\mathcal{J}$ is said to be Q-*entailing* (w.r.t $\Sigma, \mathbf{V}$) if every $\mathbf{V}, \Sigma$-sound realization of $\mathcal{J}$ satisfies Q. We also write $\mathcal{J} \models_{\mathbf{V},\Sigma} Q$. Monotonic determinacy can be restated as: for every instance $\mathcal{I}$ satisfying Q and Σ, its view image is Q-entailing w.r.t. Σ,$\mathbf{V}$.

It is easy to see that if Σ consists of existential rules, and views are CQs, then when $\mathcal{J}$ is Q-entailing there is a finite subinstance $\mathcal{J}_0$ that is Q-entailing w.r.t. Σ, $\mathbf{V}$. We only need enough facts from $\mathcal{J}$ to fire the chase rules needed to generate a match of Q. In the case where the views are in Datalog, this is not clear. But it turns out we can often obtain this "compactness property":

Theorem 5. *[Compactness for view and rule entailment] Let Σ consist of FGTGDs, $\mathbf{V}$ a set of views defined by Datalog queries, and Q an* FGDL *query. For every $\mathcal{J}$ such that $\mathcal{J}$ is Q-entailing w.r.t. Σ, $\mathbf{V}$, there is $\mathcal{J}_0$ a finite subinstance of $\mathcal{J}$ such that $\mathcal{J}_0$ is Q-entailing.*

Note that under the hypotheses, if we take a particular *unfolding of* $\mathcal{J}$ – a choice of witness for each view fact – then Q will hold in the chase of the corresponding instance, and the satisfaction of Q will depend on only finitely many facts in the chase, thus on only finitely many facts in $\mathcal{J}$. The issue is that there are infinitely many witness unfoldings, and one worries that more and more facts from $\mathcal{J}$ are required for different witnesses. The proof of Theorem 5 works by observing that the set of witnesses needed will only depend on the j quantifier rank type in guarded second order logic, for sufficiently large j. There are only finitely many such types, and thus we need only finitely many facts. Note that when we move to general Datalog Q, this result fails: see the full version (Benedikt et al. 2024) for details.

Recognizing Q-entailing instances with automata. Now consider a *tree-like* view instance $\mathcal{J}$: one with treewidth k, given by some tree code T. Theorem 5 tells us that there is a finite prefix T_0 of T whose decoding is Q-entailing. It turns out that we can often recognize these Q-entailing finite prefixes by running an ordinary finite tree automaton. This result and Theorem 5 together will allow us to reduce monotonic determinacy to reasoning with finite tree automata, rather than dealing with infinite trees.

Theorem 6. *[Automata for entailing witnesses] Let Σ consist of FGTGDs, $\mathbf{V}$ a set of views defined by* FGDL *queries, and Q a* FGDL *query. Let k be a number. There is a tree automaton $T_{Q,\mathbf{V},\Sigma,k}$ running over k-tree codes in the view*

signature such that for finite $\mathcal{J}_0$ of tree-width k, $T_{Q,\mathbf{V},\Sigma,k}$ accepts a code of $\mathcal{J}_0$ if and only if $\mathcal{J}_0 \models_{\mathbf{V},\Sigma} Q$.

The proof of this theorem involves *elimination of quantification over extensions of a tree in favor of quantification within a tree*, an idea inspired by prior work in embedded finite model theory over trees (Benedikt, Libkin, and Neven 2007). Consider the set E of tree codes of finite instances $\mathcal{J}_0$ such that $\mathcal{J}_0 \models_{\mathbf{V},\Sigma} Q$. The entailment relation quantifies over all extensions of $\mathcal{J}_0$. But in our setting, it equivalently quantifies over all tree-like instances, finite or infinite. Since all the queries involved are well-behaved, we can write a monadic second order sentence ϕ quantifying over infinite trees such that a finite tree is in E if and only if all of its extensions – "suffixes" that add on additional branches, both finite and infinite – satisfy ϕ. But we can argue that quantifying over all infinite extensions in the way does not take us out of regular tree languages.

4 Applying the Tools for Decidability and Rewritability

We first apply the tools from Section 3 to give *decidability results*. We focus on three decidable cases: *FGDL TGDs, FGDL queries, and FGDL views* – that is "everything is guarded"; *Fr-1 TGDs, MDL queries and CQ views* – which we shorten as "Fr-1 rules and queries, CQ views". and finally, *linear TGDs, CQ query, CQ views* – that is, "linearity and CQs"; For brevity we omit complexity bounds in the statements, but elementary upper bounds are easy to derive. What we want to highlight here is that *the conditions are very restrictive, but just afterwards we will show that they are necessary*.

Decidability via bounding treewidth. In the first case mentioned above, we establish decidability via the forward-reasoning approach in the previous section, and more specifically the *Forward mapping tools*:

Theorem 7. *Let Q range over* FGDL *queries, Σ over FGTGDs, and $\mathbf{V}$ over FGDL views. Then monotonic determinacy is decidable.*

Proof. It is easy to show that we have the BTVIP in the case above: the approximations of Q are tree-like, chasing with FGTGDs adds on additional branches to the tree, and applying FGDL preserves the tree structure. We just check non-emptiness of the automaton of Proposition 2. By Proposition 3 this is sufficient □

The same method applies to the *second case above*, MDL queries and CQ views, and FR-1 rules:

Theorem 8. *Let Q range over MDL queries, Σ over* FR-1 *TGDs, and $\mathbf{V}$ over CQ views. Then monotonic determinacy is decidable.*

Proof. When we approximate an MDL query and chase, we get a structure with bounded tree-width and low treespan. That is, a value appears in a fixed number of bags. When we apply CQ views, the treewidth is bounded (Benedikt et al. 2023, Lem 6.5, Thm 8.2). □

Decidability based on certain answer rewriting. We tackle the final decidable case by giving decidability via the "backwards analysis", using *certain answer rewriting techniques from the previous section*. Recall that in the case of CQ views, Step 4 in the process of Figure 1 amounts to chasing with respect to the source-to-target rules $\text{BACKV}_{\mathbf{V}}$ mentioned earlier. Thus the entire process can be considered to consist of chase steps, and we can proceed by a rewriting approach that removes some of these steps: we first find a C.A. REWRITING R_1 of Q with respect to Σ, and then a C.A. REWRITING R_2 of R_1 with respect to $\text{BACKV}_{\mathbf{V}}$. Then we have monotonic determinacy of Q over $\mathbf{V}$ with respect to Σ exactly when Q entails R_2 with respect to Σ. We can apply this straightforwardly in the case of linear TGDs, where Theorem 2 tells us we can find UCQ C.A. REWRITINGs R_1, which will generate another UCQ C.A. REWRITING R_2.

Theorem 9. *If Σ ranges over linear TGDs, then monotonic determinacy of UCQ Q over CQ views $\mathbf{V}$ relative to Σ is decidable.*

Decidability in the finite using the finite controllability result. We have sketched the argument for decidability of monotonic determinacy over all instances for three cases. In these cases, we can show that monotonic determinacy is also decidable in the finite as well. We focus on a special case of the second decidable case, which will make use of the *finite controllability tool* (Theorem 4) from the previous section.

Theorem 10. *Let Q be a CQ query, $\mathbf{V}$ a set of CQ views, and Σ is a set of* FR-1 *TGDs. If Q is monotonically determined by $\mathbf{V}$ with respect to Σ over finite structures, then the same holds over all structures.*

Proof. Using our result on certain-answer rewriting, Theorem 3, we get an MDL certain answer rewriting R_1 of Q w.r.t. Σ. By a direct argument we get an MDL c.a. rewriting R_2 of R_1 with respect to the rules that correspond to view definitions.

Now if monotonic determinacy fails over all instances, we know, by Proposition 1 and the properties of c.a. rewritings, that R_2 is not entailed by Q and Σ. But now we can apply Theorem 4 to conclude there is a finite counterexample $\mathcal{I}_1$, satisfying $Q \wedge \Sigma \wedge \neg R_2$. The view image of $\mathcal{I}_1$, and its chase under the backward mapping rules is likewise a finite instance $\mathcal{I}'_1$. We claim that there is a finite instance $\mathcal{I}_2$ containing $\mathcal{I}'_1$, satisfying $\Sigma \wedge \neg Q$. If this were the case, $\mathcal{I}_1$ and $\mathcal{I}_2$ together would contradict monotonic determinacy in the finite. We obtain $\mathcal{I}_2$ by simply applying Theorem 4 again for $\mathcal{I}'_1, \Sigma$, and R_1. □

We now apply the tools to investigate what kind of rewritings we can obtain.

View rewritings based on certain answer rewritings. One approach to obtaining rewritings is "working backwards" on the pipeline in Fig. 1, applying the tools related to *certain answer rewritings* in Section 3 to remove the last steps in the process.

Theorem 11. *Let Σ be a set of* FGTGD*s, Q be a* FGDL *query, and $\mathbf{V}$ a set of CQ views. Then if Q is monotonically determined over $\mathbf{V}$ with respect to Σ, there is a rewriting of Q in Datalog.*

Proof. We apply Proposition 4 to get a Datalog certain answer rewriting R_1 of Q under Σ, thus reversing the final step in the process. We then get a certain answer rewriting R_2 of R_1 under the rules $\text{BACKV}_{\mathbf{V}}$, which are source-to-target TGDs. We can check that R_2 is the desired rewriting. □

View rewriting based on forward-backward. The certain-answer rewriting approach applies only to CQ views. We now provide an approach based on bounding treewidth.

Theorem 12. *Let Σ be a set of* FGTGD*s, Q a Boolean Datalog query and $\mathbf{V}$ a set of* FGDL *views. Then if Q is monotonically determined by $\mathbf{V}$ $w.r.t.$ Σ there is a* POSLFP *rewriting.*

Proof. We use a modification of the forward-backward approach mentioned after Theorem 1. Using the same argument as Proposition 2 we can get a Büchi automaton for tree-like view images of unfolding of Q. We can project on to the view signature to get a Büchi automaton accepting view images. And because $\Sigma, Q, \mathbf{V}$ has the BTVIP it is sufficient to check such tree-like counterexamples. If we ignore acceptance conditions in this automaton, we get an ordinary finite tree automaton. We apply the backward mapping of Theorem 1 to this to get Datalog rules. We can add nested fixpoints to capture the Büchi acceptance conditions. □

We do not know if Datalog suffices for FGTGDs. But we can show POSLFP is necessary in order to rewrite Q monotonically determined over $\mathbf{V}$ w.r.t. Σ for cases when we have the BTVIP:

Theorem 13. *There are Σ, Datalog Q and $\mathbf{V}$ such that the BTVIP holds, with Q monotonically determined by $\mathbf{V}$ w.r.t. Σ, where there is a* POSLFP *rewriting but no Datalog rewriting over all structures.*

The argument will rely on the *failure* of "compactness of entailment" for this class. We provide an example where compactness fails, and show that this failure implies that a Datalog rewriting is impossible.

Better rewritings using Compactness of Entailment and Automata for Entailing Witnesses. We can also use *Compactness of Entailment* and *Automata for entailing witnesses* from Section 3 to show that if Q is in FGDL, we can do better than POSLFP:

Theorem 14. *Let Σ be a set of* FGTGD*s, Q a Boolean* FGDL *query, $\mathbf{V}$ a set of* FGDL *views. Then if Q is monotonically determined by $\mathbf{V}$ w.r.t.Σ, then there is a Datalog rewriting.*

Proof. Since $Q, \mathbf{V}, \Sigma$ has the BTVIP, there exists natural k such that for every CQ approximation of Q there exists its chase C_n under Σ having $\text{TW}^+(\mathbf{V}(C_n)) \leq k$.

Applying *Compactness of Entailment*, Theorem 5, for any view instance of treewidth k that entails (via BACKV and Σ) Q, there is a finite $\mathcal{J}$ contained in it that entails Q the same way. By the *Automata for entailing witnesses result*, Theorem 6, there is an ordinary tree automaton A that captures the codes of these finite subinstances. We convert A to Datalog, using the backward mapping of Theorem 1. □

Using a similar technique, we can also conclude rewritability when Q is a CQ but $\mathbf{V}$ are arbitrary Datalog.

Theorem 15. *Suppose Q is a Boolean UCQ (resp. CQ), $\mathbf{V}$ a set of Datalog views, with Q monotonically determined by $\mathbf{V}$ w.r.t.Σ. Then there is a UCQ (resp. CQ) rewriting of Q over $\mathbf{V}$ relative to Σ.*

5 Surprising Undecidability

The three cases that give us decidability in Section 4 amount to much stronger hypotheses than in the absence of rules, where all the cases without recursion are decidable. Remarkably, they cannot be loosened. If we just mix the constraint classes from the second and third decidable cases, we get undecidability *even with CQ queries and views*.

Theorem 16. *The problem of monotonic determinacy is undecidable when Σ ranges over combinations of Linear TGDs and Frontier-1 TGDs, Q over Boolean CQs, and $\mathbf{V}$ over CQ views. If we allow MDL queries, we have undecidability with just Linear TGDs.*

The idea is that we can code a cellular automaton in a monotonic determinacy problem.

If we allow UCQ views – adding disjunction, but no recursion – we can even get undecidability for rules in the intersection of the two well-behaved rule classes:

Theorem 17. *The problem of monotonic determinacy is undecidable when Σ ranges over UIDs, which are linear and frontier-1, Q over Boolean UCQs, and $\mathbf{V}$ over UCQ views.*

Here the idea is to code a tiling problem (or a nondeterministic Turing Machine). In the process of Figure 1, chasing the canonical database of one disjunct of the UCQ – the "start disjunct" – with the rules will generate two infinite axes, while one disjunct of a UCQ view will generate the cross product of the axes: all(x, y) co-ordinate pairs. "Reverse chasing" with the other disjuncts of the views will generate grid points for each such pair, with each grid point tagged with a tile predicate. Another disjunct of the CQ – a "verification disjunct" – will return true if the tiles violate one of the forbidden patterns. Note that the use of a UCQ will allow us to code non-deterministic computation, while a CQ view (as in the previous theorem) allows us only to mimic deterministic computation.

With effort we can get undecidability even with Q a CQ:

Theorem 18. *The problem of monotonic determinacy is undecidable when Σ are UIDs, Q is a Boolean CQ, and $\mathbf{V}$ are UCQ views.*

Similarly to the proof of Theorem 17 we code a tiling problem. The main new challenge is we do not have multiple disjuncts in Q to test for different violations. So now *conjuncts* in our CQ will represent different violations of a correct tiling. We arrange that if there is one violation of correctness of a tiling, there are "degenerate ways" to map the remaining conjuncts of the CQ. Details are in the full version (Benedikt et al. 2024).

Thus far we have focused on FGTGDs, where the chase may not terminate. For full TGDs (no existentials in the head) the chase terminates. It follows that monotonic determinacy is finitely controllable. Some of the rewritability

<table>
<tr><th>$Q \backslash \mathbf{V}$</th><th>CQ</th><th>MDL, FGDL</th><th>UCQ, DL</th></tr>
<tr><td>CQ, UCQ</td><td>Fr-1 ∪ Lin - Und., Thm. 16
Lin - Dec., Thm. 9</td><td>FGTGD - Dec., Thm. 7</td><td rowspan="4"></td></tr>
<tr><td>MDL</td><td>Lin - Und., Thm. 16
full TGD - Und., Thm. 19
Fr-1 - Dec., Thm. 8</td><td>full TGD - Und., Thm. 19
FGTGD - Dec., Thm. 7</td></tr>
<tr><td>FGDL</td><td>Lin - Und., Thm. 16</td></tr>
<tr><td>DL</td><td colspan="2">UID - Und., Thm. 18</td></tr>
</table>

Figure 3: Decidability with Rules

<table>
<tr><th>$Q \backslash \mathbf{V}$</th><th>CQ</th><th>MDL, FGDL</th><th>UCQ</th><th>DL</th></tr>
<tr><td>CQ</td><td>CQ, †</td><td rowspan="2">DL, Thm. 14</td><td rowspan="2">UCQ, Thm 2.8 †</td><td>CQ Thm. 15</td></tr>
<tr><td>UCQ</td><td>UCQ, Thm 2.8 †</td><td>UCQ Thm. 15</td></tr>
<tr><td>MDL, FGDL</td><td>DL, Thm. 11</td><td rowspan="2">n.n. DL</td></tr>
<tr><td>DL</td><td>Open</td><td>PosLFP, Thm. 12</td></tr>
</table>

Figure 4: Rewritability with Rules; †(Benedikt et al. 2016)

results easily carry over from the case without rules. For example, for the case of CQ views, we can infer that monotonically determined Datalog queries are Datalog rewritable, using the inverse rules algorithm (Duschka, Genesereth, and Levy 2000). But we can show that *even for full TGDs, some of the decidable cases from Fig. 1 become undecidable*. In particular:

Theorem 19. *The problem of monotonic determinacy is undecidable when Q ranges over MDL, $\mathbf{V}$ over unary atomic views, and Σ over full TGDs. For such $Q, \mathbf{V}, \Sigma$ the problems of Datalog rewritability, CQ rewritability, and the variants of these problems over finite instances are also undecidable.*

We proceed here by constructing a family of $Q, \Sigma, \mathbf{V}$ where chasing the approximations of Q allows us to build an arbitrarily large deterministic computation graph. The views will just indicate whether this computation is accepting. This will allow us to encode a deterministic machine acceptance problem as monotonic determinacy. Because $\mathbf{V}$ are so restricted (unary atomic!), monotonic determinacy coincides with CQ rewritability for this family.

6 Conclusions

We investigated reasoning problems involving the interaction of views, queries, and background knowledge. We developed tools which can be applied to get positive results about monotonic determinacy. And we show that even when you combine simple queries, views, and rules, static analysis problems involving all of them can become undecidable.

The goal in this paper was *not* to discuss all we know about monotonic determinacy with rules. Many results follow easily from prior work. But for completeness, Figures 3 and 4 summarize our knowledge. The results apply to FGTGDs, except where we indicate restrictions (e.g. LIN for Linear rules). In the tables **bold font** highlights a significant difference from the case without rules. In Figure 4 Blue indicates that results from prior work (Benedikt et al. 2016) can be used out of the box. Red indicates use of certain answer-rewriting approaches, while Black indicates an approach based on bounding treewidth. For readability, we omit some results for full existential rules in the table. As shown in the table, a number of cases are left open, both for decidability and rewritability.

Acknowledgements

Piotr Ostropolski-Nalewaja was supported by the European Research Council (ERC) Consolidator Grant 771779 (DeciGUT).

References

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

Afrati, F., and Chirkova, R. 2019. *Answering Queries Using Views*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

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

Bárány, V.; Benedikt, M.; and ten Cate, B. 2018. Some model theory of guarded negation. *J. Symb. Log.* 83(4):1307–1344.

Benedikt, M.; ten Cate, B.; Leblay, J.; and Tsamoura, E. 2016. *Generating Plans from Proofs: the Interpolation-based Approach to Query Reformulation*. Morgan Claypool.

Benedikt, M.; Kikot, S.; Nalewaja-Ostropolski, P.; and Romero, M. 2023. On monotonic determinacy and rewritability for recursive queries. *ACM TOCL*.

Benedikt, M.; Kikot, S.; Marti, J.; and Ostropolski-Nalewaja, P. 2024. Monotone rewritability and the analysis of queries, views, and rules. *CoRR* abs/2407.14907.

Benedikt, M.; Libkin, L.; and Neven, F. 2007. Logical definability and query languages over ranked and unranked trees. *ACM Trans. Comput. Log.* 8(2):11.

Calì, A.; Gottlob, G.; Lukasiewicz, T.; and Pieris, A. 2011. A logical toolbox for ontological reasoning. *SIGMOD Record* 40(3):5–14.

Calì, A.; Lembo, D.; and Rosati, R. 2003. Query rewriting and answering under constraints in data integration systems. In *IJCAI*.

Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Vardi, M. Y. 2002. Lossless regular views. In *PODS*.

Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Vardi, M. Y. 2007. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theoretical Computer Science* 371(3):169–182.

Courcelle, B. 1991. Recursive queries and context-free graph grammars. *Theoretical Computer Science* 78(1).

Duschka, O. M.; Genesereth, M. R.; and Levy, A. Y. 2000. Recursive query plans for data integration. *J. Log. Prog.* 43(1):49 – 73.

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

Francis, N.; Segoufin, L.; and Sirangelo, C. 2015. Datalog rewritings of regular path queries using views. *LMCS* 11(4).

Gogacz, T., and Marcinkowski, J. 2015. The hunt for a red spider: Conjunctive query determinacy is undecidable. In *LICS*.

Gogacz, T., and Marcinkowski, J. 2016. Red spider meets a rainworm: Conjunctive query finite determinacy is undecidable. In *PODS*.

Gogacz, T.; Ibáñez-García, Y. A.; and Murlak, F. 2018. Finite query answering in expressive description logics with transitive roles. In *KR*.

Levy, A. Y.; Mendelzon, A. O.; Sagiv, Y.; and Srivastava, D. 1995. Answering queries using views. In *PODS*.

Lukasiewicz, T.; Calì, A.; and Gottlob, G. 2012. A general datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics* 14(0):57–83.

Maier, D.; Mendelzon, A. O.; and Sagiv, Y. 1979. Testing implications of data dependencies. *TODS* 4(4):455–469.

Nash, A.; Segoufin, L.; and Vianu, V. 2010. Views and queries: Determinacy and rewriting. *TODS* 35(3).

Perez, J. 2011. *Schema Mapping Management in Data Exchange Systems*. Ph.D. Dissertation, University of Chile.

Salvat, E., and Mugnier, M.-L. 1996. Sound and complete forward and backward chainings of graph rules. In *ICCS*.

Thomas, W. 1997. Languages, Automata, and Logic. In Rozenberg, G., and Salomaa, A., eds., *Handbook of Formal Languages*.