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

Michael Benedikt[1] , **Stanislav Kikot** , **Johannes Marti** , **Piotr Ostropolski-Nalewaja**[4]
[1]University of Oxford [4]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 mechanism for defining an interface to a set of datasources. 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 we have that $\mathbf{V}(\mathcal{I}) \subseteq \mathbf{V}(\mathcal{I}')$ 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 we have that $R(\mathbf{V}(\mathcal{I})) = \mathsf{Output}(Q, \mathcal{I})$ 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

[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. 2011b) in the literature.

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, 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 an (unguarded) CQ or even 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

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

logic, and thus in polynomial time. In certain cases we can conclude rewritability in Datalog.

Contributions. Our first contribution is a set of tools for reasoning with queries, views, and rules:

- We extend the "forward-backward" method for obtaining rewritings to the presence of rules.
- We provide new results on *certain-answer rewritability* of Datalog queries with rules, and on *finite controllability of recursive queries with rules*.
- We present a new kind of "compactness property" for entailments involving Datalog queries, rules, and views. 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 hope they 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, monotonic determinacy for CQ queries and CQ views is undecidable even for reasonably simple rules: combinations of frontier-one and linear.

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 appendix.

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 con-

sider 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 with head and body as above. The notion of dependency 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 $\textsc{Chase}_\Sigma(\mathcal{I})$ to denote any instance (possibly infinite) formed from $\mathcal{I}$ by iteratively applying chase steps, where Σ holds. We say that $\textsc{Chase}_\Sigma(\mathcal{I})$ *is finite* if we can perform finitely many chase steps and obtain an instance satisfying Σ. We say that Σ *has terminating chase* if $\textsc{Chase}_\Sigma(\mathcal{I})$ 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. Figure 3 gives an abstract proce-

MONDET$(Q, \mathbf{V}, \Sigma)$:

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

Figure 3: Process for checking monotonic determinacy.

dure 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 $\text{BACKV}_\mathbf{V}(\mathcal{J})$ 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 $\text{BACKV}_\mathbf{V}(\mathcal{J})$, 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, $\text{BACKV}_\mathbf{V}(\mathcal{J})$ can be thought of as the result of a "disjunctive chase". Next, we chase instances in $\text{BACKV}_\mathbf{V}(\mathcal{J}_n)$ 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.

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 Figure 3 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 3.

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 3 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 $\tau = (\text{VERTICES}, E)$, 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 VERTICES such that the following conditions hold: – for every atom $R(\boldsymbol{c})$ in $\mathcal{I}$, there is a vertex $v \in$ VERTICES with $\boldsymbol{c} \subseteq \lambda(v)$; – for every element c in $\mathcal{I}$, the subgraph of τ induced over the set $\{ v \in \text{VERTICES} \mid c \in \lambda(v) \}$ 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 Appendix A. 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 $\mathcal{I}$, $\text{TW}^+(\mathcal{I})$, is the minimum adjusted width of a tree decomposition of $\mathcal{I}$. For a tree decomposition $\mathcal{TD}$ of data instance $\mathcal{I}$ let $\text{TSPAN}(\mathcal{TD})$ (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 $\text{CANONDB}(Q_n)$ under Σ it holds that $\text{TW}^+(\mathbf{V}(C)) \leq k$. 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 3 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 proceeds along the general lines used in finitely controllability for description logics, see in particular (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 shall 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 in the 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.

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 3. 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 appendix 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 3 amounts to chasing with respect to the source-to-target rules $\text{BACK}\mathbf{V}_{\mathbf{V}}$ 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 $\text{BACK}\mathbf{V}_{\mathbf{V}}$. 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. 3, 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 $\text{BACK}\mathbf{V}_{\mathbf{V}}$, 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 $\mathrm{TW}^+(\mathbf{V}(C_n)) \leq k$.

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 hypotheses that give us decidability in Section 4 are much stronger 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 case above, 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 atomic 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 3, 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 more 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 appendix.

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 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*

<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 4: 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 5: Rewritability with Rules; †(Benedikt et al. 2016)

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 4 and 5 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 5 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.; Mugnier, M.-L.; Rudolph, S.; and Thomazo, M. 2011a. Walking the Complexity Lines for Generalized Guarded Existential Rules. In *IJCAI*.

Baget, J.-F.; Leclère, M.; Mugnier, M.-L.; and Salvat, E. 2011b. 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.

Bárány, V.; Cate, B. t.; and Segoufin, L. 2015. Guarded negation. *J. ACM* 62(3).

Bekić, H. 1984. Definable operations in general algebras, and the theory of automata and flowcharts. In *Programming Languages and Their Definition*.

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.; Bourhis, P.; Cate, B. T.; Puppis, G.; and Boom, M. V. 2021. Inference from visible information and background knowledge. *ACM Transacations on Computational Logic* 22(2).

Benedikt, M.; Buron, M.; Germano, S.; Kappelmann, K.; and Motik, B. 2022. Rewriting the infinite chase. *Proc. VLDB Endow.* 15(11).

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

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.

Benedikt, M. 2020. Lecture notes on guarded logic and automata. www.cs.ox.ac.uk/michael.benedikt/readingcourse/coursenotes.pdf.

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 relation-

ship between rewriting, answering and losslessness. *Theoretical Computer Science* 371(3):169–182.

Chaudhuri, S., and Vardi, M. Y. 1997. On the equivalence of recursive and nonrecursive datalog programs. *JCSS* 54(1):61–78.

Courcelle, B. 1990. The monadic second order theory of graphs i: Recognisable sets of finite graphs. *Information and Computation* 85:12—75.

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*.

Grädel, E.; Hirsch, C.; and Otto, M. 2002. Back and forth between guarded and modal logics. *TOCL* 3(3):418–463.

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.

Rabin, M. O. 1970. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory*, volume 59. 1–23.

Rosati, R. 2011. On the finite controllability of conjunctive query answering in databases under open-world assumption. *Journal of Computer and Systems Science* 77(3):572–594.

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*.

A Additional details for codes, automata, Datalog, and the characterization of monotonic determinacy, Section 2

A.1 Basics of tree codes, tree automata, and Monadic Second Order Logic

A key component of our arguments involves coding bounded treewidth structures by labelled trees with a fixed signature – that is, the notion of tree code, referred to in the body of the paper. We now review the formal details, as well as the notation that we will use throughout the appendix. We emphasize that the encoding we are using is quite standard: see, for example, the appendix of (Benedikt et al. 2021).

Fix a number $k > 0$ and fix a relational schema σ. We define a schema σ_k^{code} of unary predicates such that it contains the following predicates:

1. For every two numbers $l, l' \in \{1, \ldots, k\}$ a predicate $T^{=}_{l,l'}$.
2. For every m-ary relation symbol R from σ and m-tuple $\mathbf{n} = (l_1, \ldots, l_m) \in \{1, \ldots, k\}^m$ a predicate $T^R_{\mathbf{n}}$. In the case where $m = 0$ this means that we add just one predicate T^R_{ε} for the empty tuple $\varepsilon = ()$.
3. For every partial injective map g from $\{1, \ldots, k\}$ to $\{1, \ldots, k\}$ a predicate T_g.

Informally, the predicates allow a single vertex of a tree to encode the σ predicates holding on up to k elements of a structure. A number in $1 \ldots k$ occurring in a predicate of a vertex is a *local name* of the vertex. The predicates $T^{=}_{l,l'}$ tell which pairs of local names code the same element. The predicates $T^R_{\mathbf{n}}$ indicate which tuples of elements coded in a vertex satisfy relation R. The predicates T_g describe how the elements coded in a vertex relate to the elements coded in its parent.

A *finite tree* is a finite directed graph with no cycles, in which every element has at most one predecessor and exactly one element, denoted the root of the tree, has no predecessor. An *ω-tree* is as above, but where the set of vertices is countably infinite. When we refer to a *tree* we mean either a finite tree or an ω-tree.

A σ_k^{code}-tree $\mathcal{T}$ is a tree whose vertices are labelled with predicates from σ_k^{code}. We assume that there is an r such that every node that is not a leaf has exactly r children which are ordered. If we want to emphasize the number of children, we call $\mathcal{T}$ a tree with *branching width* r.

A σ_k^{code}-tree $\mathcal{T}$ is *coherent* if

1. For every vertex v of $\mathcal{T}$ the assignment of the predicates $T^{=}_{l,l'}$ to v satisfies the axioms of equality. Precisely, this means that for all $l, l', l'' \in \{1, \ldots, k\}$
   (a) the predicate $T^{=}_{l,l}$ is true of v,
   (b) if $T^{=}_{l,l'}$ is true of v then $T^{=}_{l',l}$ is true of v, and
   (c) if $T^{=}_{l,l'}$ and $T^{=}_{l',l''}$ are both true of v then $T^{=}_{l,l''}$ is also true of v.
2. For every vertex v of $\mathcal{T}$ the assignment of predicates is closed under substitution of equal local names. This means that if $T_{l,l'}$ is true at v and $T^R_{\mathbf{n}}$ is true at v for some m-ary relation symbol R and M tuple $\mathbf{n} = (l_1, \ldots, l_k)$ such that $l_i = l$ then $T^T_{\mathbf{m}}$ is also true v, where $\mathbf{m} = (h_1, \ldots, h_m)$ is such that $h_i = l'$ and $h_j = l_j$ for all $j \neq i$.
3. If v is a child of w, T_g is true at v and $T^{=}_{l,l'}$ is true at w then $l \in \text{DOM}(g)$ iff $l' \in \text{DOM}(g)$. Moreover, if $l \in \text{DOM}(g)$ and $l' \in \text{DOM}(g)$ then $T^{=}_{g(l),g(l')}$ is true at v.
4. For every nullary predicate P in σ and vertices v and w of $\mathcal{T}$ we have that T^P_{ε} is true of v iff T^P_{ε} is true of w.
5. For every vertex v of $\mathcal{T}$ there is exactly one partial injection g such that T_g is true of v.

Coherent σ_k^{code} trees are also called *tree codes*. The *decoding* of a tree code $\mathcal{T}$ is the structure $\mathfrak{D}(\mathcal{T})$ that is defined as follows:

- The domain of $\mathfrak{D}(\mathcal{T})$ consists of all the pairs (v, l), where v is a vertex in $\mathcal{T}$ and $l \in \{1, \ldots, k\}$, quotiented by the smallest equivalence relation $\equiv$ such that
  - if $T^{=}_{l,l'}$ is true at v then $(v, l) \equiv (v, l')$, and
  - if w is the parent of v, T_g is true at v and $l \in \text{DOM}(g)$ then $(w, l) \equiv (v, g(l))$.

  We write $[v, l]$ for the equivalence class of the pair (v, l) under $\equiv$.
- A tuple $(c_1, \ldots, c_m)$ of equivalence classes is put in the interpretation of the m-ary predicate R from σ iff there are $l_1, \ldots, l_m$ and a vertex v such that $(v, l_i) \in c_i$ holds for all $i = 1, \ldots, m$.

We call a tree code $\mathcal{T}$ *active* if there is at least one element in the active domain of $\mathfrak{D}(\mathcal{T})$.

We stress again that this is the standard definition. In the standard setting, we would assume also that *tree codes, and the instances they define via decoding, are finite*. But the definitions also make sense for infinite trees (say, with countable depth). The following proposition connecting codings and decodings is routine.

Proposition 6. *For every σ-structure $\mathfrak{M}$ of adjusted treewidth $k - 1$ and number $r \geq 2$ there is a tree code $\mathcal{T}$ with branching width r such that $\mathfrak{D}(\mathcal{T})$ is isomorphic to $\mathfrak{M}$.*

Tree automata and MSO. We deal with two kinds of tree automata in this work. Finite tree automata are associated with a label alphabet Σ and a branching factor r. They are given by a finite set of states Q and an initial state $q_0 \in Q$, a transition relation that is a subset of $Q^r \times \Sigma \times Q$, and a subset of Q, the final states. The notion of an accepting run of such an automaton on a tree with branching r is the usual one. We annotate each node of the tree with a state, such that the root gets an initial state and leaves get a final state. The states assigned in the run to a parent and child must be consistent with the transition relation. An automaton accepts a tree if it has an accepting run, and the language of an automaton is the set of trees it accepts.

Recall that *MSO* is *Monadic Second Order Logic* over trees. We have first and second order variables, a binary relation symbol for the parent child relationship, and unary relations for each alphabet symbol. Atomic formulas include $S(x)$ for S a second order variable, and the usual atomic formula involving the parent-child relation and the unary predicates. We build up using Boolean operations, first order quantifiers, and second order quantifiers. The notion of a tree satisfying an MSO sentence is the standard one, and the language of an MSO sentence is the set of trees satisfying it.

We will use the well-known fact (Thomas 1997):

Proposition 7. *Tree automata and MSO express the same set of languages of finite trees.*

These are the *regular tree languages*.

We also make use of automata over *infinite trees*, such as *Büchi tree automata*. The latter are specified in the same way as finite tree automata. They accept an r-branching tree whose depth is infinite. An accepting run associates each node of the tree with a state. The root must be labelled with the initial state, and the transition function must be respected as with finite tree automata. But now the accepting states must occur infinitely often down every branch. Consider *Weak Monadic Second Order Logic* (WMSO), where second order quantification is only over finite subsets of the vertices. It is known (Rabin 1970) that a WMSO collection of trees is recognizable by a Büchi tree automaton. Indeed the WMSO definable sets are exactly those such that both the set and its complement are recognizable by Büchi automata.

More on automata and tree codes. Naturally, we can run tree automata on tree codes. For this purpose we make the assumption that the tree codes have a constant branching width of r for all nodes that are not leafs and that the children of any parent node are ordered. We also think of σ_k^{code}-trees as trees over the alphabet Σ that contains letters for all sets of predicates in σ_k^{code}. If $\mathcal{T}$ is coherent then any such set contains only one predicate of the from T_g. In this case we write $\textsc{Code}_B^g$ for the letter $\{T_g\} \cup B$, where B is a set of predicates of the form $T_{l,l'}^{=}$ or $T_{\mathbf{n}}^R$. We call B a *bag*.

Given a subsignature σ_V of σ we can define for every σ_k^{code}-tree $\mathcal{T}$ its restriction $\mathcal{T}\restriction_{\sigma_V}$ to be a $(\sigma_V)_k^{\mathrm{code}}$-tree in which from every letter we drop all predicates $T_{\mathbf{n}}^R$ such that R is not in σ_V. It is easy to see that this is equal to the restriction of $\mathfrak{D}(\mathcal{T})$, which is defined using the full signature σ, to a smaller label set.

When we restrict to codes that are finite, we will run finite tree automata. We will use standard results about closure of regular tree languages under *projection*. That is, for every σ_k^{code}-tree automaton A there is a $(\sigma_V)_k^{\mathrm{code}}$-tree automaton $A\restriction_{\sigma_V}$ such that for all $(\sigma_V)_k^{\mathrm{code}}$-trees $\mathcal{T}$

$$A\restriction_{\sigma_V} \text{ accepts } \mathcal{T} \quad \text{iff} \quad \text{there exists some } \sigma_k^{\mathrm{code}}\text{-tree } \mathcal{T}^+ \text{ accepted by } A \text{ such that } \mathcal{T}^+\restriction_{\sigma_V} = \mathcal{T}.$$

A.2 More details on approximating Datalog with CQs

In the body of the paper we referred to CQ approximations of a Datalog query. This is a well-known notion that originates in (Chaudhuri and Vardi 1997). We give the formalization that we use in this work, which is taken from (Benedikt et al. 2023), see in particular Appendix C.

We will deal with Head-Unconstrained (HU) Datalog queries, which do not allow repeated variables in the head, except in the goal predicate, which cannot occur in the body of rules. Every Datalog query can be rewritten to an HU Datalog query (Benedikt et al. 2023, Proposition 3.1). HU Datalog is easier to deal with, since we can always unify an intensional fact in a rule body with any rule head using the same predicate.

Definition 7 (CQ approximation)**.** *A* CQ approximation tree *for a HU Datalog query* $(\Pi, \textsc{Goal})$ *will be a tree with each node labelled with an atom, with that atom using only variables. Non-leaf nodes will always be labelled with atoms over intensional relations of* Π*, while leaf nodes will be labelled with extensional atoms of the input schema. Approximation trees will never have the root as a leaf. We require that the root label of a CQ approximation tree does not contain repeated variables, unless it involves the* Goal *predicate. For example, using* $\textsc{Goal}(x, y, x, z)$ *as a root label is allowed, but* $A(x, y, x, z)$ *for* A *a different predicate is not. Each non-leaf node* n *will also be associated with a rule of* Π*, denoted* $\textsc{RuleOf}(n)$*.*

The idea of the repeated variable restriction is: we do not need repeated variables in head in intermediate predicates, because we can push the repetition into the rule body atoms that will unify with the heads. The only reason we need repeated variables in heads is if the final query we want to perform involves repetition.

We define the depth n *CQ approximation trees for a Datalog query, rooted at an arbitrary intensional fact* A *with no repeated variables, by induction on* n*. The base case is a tree consisting of a single root node labelled with an intensional atom* A *having all variables distinct, whole children are all extensional atoms, with the children matching the rule bodies of a rule for* A*.*

For the inductive case, consider the case of a non-goal rule of Π*,* $\rho := U(\boldsymbol{x}) := \beta_1, \ldots \beta_k$ *and let* $\beta_{n_1} \ldots \beta_{n_j}$ *be intensional atoms in the rule body. Let* $\mathbb{T} = T_{n_1} \ldots T_{n_j}$ *be a tuple of CQ approximation trees such that the relation of* β_{n_i} *is the same as the relation of the root label of* T_{n_i}*, for every* i*. Then the tree* T *built in the following way is a CQ approximation tree:*

- *Set $U(\boldsymbol{x})$ to be the label of the root node n of T and set* $\text{RULEOF}(n) = \rho$.
- *For each n_i Let h_{n_i} be a homomorphism taking the atom A_{n_i} in the root label of T_{n_i} to β_{n_i}. Such a homomorphism always exists if the relations match, since the variables in the intensional atom A_{n_i} are distinct.*
- *Let T'_{n_i} be obtained from T_{n_i} by replacing y in every label of T_{n_i} with $h_{n_i}(y)$ for every variable y of the root label of T_{n_i}, and by replacing other variables of T_{n_i} with fresh ones.*
- *For the extensional atoms β_i in the rule body, we let T_i be a tree consisting only of the atom.*
- *The nodes of T will be the root node defined above along with the nodes of T'_i. We set the children of the root node to be the root of every T'_i. The edge relations and labels outside of the root are those inherited from T'_i.*

Every CQ approximation tree rooted at the goal predicate of (Π, GOAL) defines a CQ, by conjoining the leaf atoms. This is a *CQ approximation of the Datalog query*. It is well-known and straightforward to see that:

Proposition 8. *A Datalog query is the union of its approximations.*

We now claim that approximations of a Datalog program can be captured with tree automata. The following is a variation of Proposition 6.9 of (Benedikt et al. 2023), proven in Appendix C. But the idea goes back to Chaudhuri and Vardi, see (Chaudhuri and Vardi 1997):

Proposition 9. *If P is a Datalog query and k is a bound on the size of the bodies of rules in P then every CQ approximation of P has a cannonical tree decomposition, which witnesses that it has tree-width k. And there is a finite tree automaton that accepts exactly these tree decompositions.*

Note that we do *not* require that Q is in FGDL. The approximations are well-behaved for any Datalog query. This result will be used as a component of showing that certain examples have the BTVIP – e.g. in Theorem 7. It will also be used in some of our rewriting results via the forward-backward method.

Of course, we will generally need Q to be in Datalog to apply this result to obtain decidability, since without that we cannot effectively check whether $\neg Q'$ holds on a tree, which is part of being a counterexample to monotonic determinacy.

A.3 Tree-like structure for the chase of instances, and the chase of approximations

In the previous subsection, we argued that the tree codes of approximations of a Datalog program form a regular set of trees. This shows that the first step in the process of Figure 3 is always treelike and regular. Recall that the BTVIP property states that the first 3 steps of the process of the figure can be captured with a regular tree language, in the sense that we can always find some representation that is treelike and regular.

We will make use of the fact that for FGTGDs, the chase can be given a tree-like structure, a result that is well-known since the discovery of the $Datalog^{\pm}$ family (Lukasiewicz, Calì, and Gottlob 2012) and of FGTGDs (Baget et al. 2011a). If we have a finite instance $\mathcal{I}$ and FGTGDs Σ, the chase can be given the structure of a tree with the root containing all facts of $\mathcal{I}$. We will ensure that for every trigger for FGTGD τ, it *fires locally in the tree* – its image consists of facts in a single node n. The facts generated by that chase step will be in a child c of n, and c will inherit all facts from n that are guarded in c – that is, facts of n whose are arguments are contained in some fact of c. In order to ensure that triggers always fire locally, acts will be propagated from children to parents but only facts that are guarded in the parent. In particular, we have the following two properties:

- Bounded treewidth of the chase. Although the chase can be infinite, the tree structure described above forms a tree decomposition in the usual sense, with the width bounded by the size of $\mathcal{I}$ and the maximal size of a rule head in Σ.
- Guarded interfaces. For each subtree of the root, the elements of $\mathcal{I}$ that appear in the subtree form a guarded set in $\mathcal{I}$.

In the case that $\mathcal{I}$ is a CQ approximation tree Q_n of a Datalog query Q, it already comes with a tree decomposition whose width is independent of n, by Proposition 9. When we chase such an approximation with FGTGDs, we will do so preserving this tree structure, rather than putting all of the elements of Q_n in the root note. When we make the first non-full chase steps, we will be creating children of nodes that may be in the interior of the approximation tree: that is, we grow the chase tree off of each node in the approximation. Although the chase generates new facts on the domain of the canonical database of Q_n, by the guarded interface property above, each such fact will be guarded by a fact in the canonical database of Q_n. In particular, the following result, will be used later in the appendix, see for example the proof of Theorem 7.

Proposition 10. *For any Datalog query, there is a uniform bound on the treewidth of* $\text{CHASE}_\Sigma(Q_n)$. *We can effectively form an automaton on infinite trees that accepts the canonical tree structures associated with chasing the approximation trees of Q_n in the way described above, or a WMSO sentence that describes these trees.*

A.4 Proof of Proposition 1: the process of Figure 3 is correct

Recall the process of Figure 3:

Many of the results in the submission rest on Proposition 1 in the body, stating that this process is correct for monotonic determinacy with rules over all instances. We recall the statement of the proposition:

Q is monotonically determined over $\mathbf{V}$ w.r.t. Σ if and only if the process of Figure 3 returns true.

MONDET$(Q, \mathbf{V}, \Sigma)$**:**

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

Figure 6: Process for checking monotonic determinacy.

We now overview the proof. We stress that this it follows the same argument as Lemma 5.4 in (Benedikt et al. 2023). There are two extra steps involved in the process, involving chasing, and these are reflected in the proof.

In one direction, suppose that the process in the figure fails (returns false). Let C_n and $C'_{m,n}$ as above witness this. Note that $\mathbf{V}(C_n) \subseteq \mathbf{V}(C'_{m,n})$ as $\mathbf{V}(Q'_{m,n}) = \mathbf{V}(Q_n)$ and $Q'_{m,n} \subseteq C'_{m,n}$. Moreover $C_m \models Q$ and $C'_{m,n} \not\models Q$. Thus C_n and $C'_{m,n}$ serve as a counterexample to monotonic determinacy.

Before we prove the other direction, note the following propositions:

Proposition 11. *For any instance $\mathcal{I}$ there exists an instance $\mathcal{I}' \in \text{BACKV}_\mathbf{V}(\mathbf{V}(\mathcal{I}))$ such that $\mathcal{I}'$ homomorphically maps to $\mathcal{I}$.*

Proposition 12. *Let $\mathcal{I}$ and $\mathcal{I}'$ be two instances such that there is a homomorphism from $\mathcal{I}$ to $\mathcal{I}'$. Then for every instance $B \in \text{BACKV}_\mathbf{V}(\mathbf{V}(\mathcal{I}'))$ there exists an instance $A \in \text{BACKV}_\mathbf{V}(\mathbf{V}(\mathcal{I}))$ such that A homomorphically maps to B.*

Beginning the argument for the other direction, suppose that monotonic determinacy fails with witness instances $\mathcal{I}_1$ and $\mathcal{I}_2$. That is, $\mathcal{I}_1 \models Q$, $\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy Σ, the view image $\mathbf{V}(\mathcal{I}_2)$ of $\mathcal{I}_2$ contains all facts in the view image $\mathbf{V}(\mathcal{I}_1)$ of $\mathcal{I}_1$, but $\mathcal{I}_2$ does not satisfy Q. We shall show, that the above procedure returns false. Let Q_n be an unfolding of Q that holds in $\mathcal{I}_1$. Such a Q_n exists because $\mathcal{I}_1 \models Q$. We let C_n and $\mathcal{J}_n$ be as in Figure 3. Note that there exists a homomorphism h from C_n to $\mathcal{I}_1$. Consider $\mathbf{V}(h(C_n))$ and $\text{BACKV}_\mathbf{V}(\mathbf{V}(h(C_n)))$. From Proposition 11 there exists an instance W of $\text{BACKV}_\mathbf{V}(\mathbf{V}(h(C_n)))$ that homomorphically maps to $\mathcal{I}_2$ as $\mathbf{V}(h(C_n)) \subseteq \mathbf{V}(\mathcal{I}_1) \subseteq \mathbf{V}(\mathcal{I}_2)$. Finally note, from Proposition 12, that there exists an instance, call in $Q'_{m,n}$, that is in $\text{BACKV}_\mathbf{V}(\mathcal{J}_n)$ and is mappable to W. Note that $C'_{m,n} = \text{CHASE}_\Sigma(Q'_{m,n})$ maps to $\mathcal{I}_2$, thus $C'_{m,n} \not\models Q$. Therefore, the procedure returns false.

B Additional material related to tools for positive results, Section 3

B.1 More details concerning the forward-backward method

Forward mapping results. We recall the results on forward mapping, beginning with Proposition 2:

For each Q in FGDL, $\mathbf{V}$ in FGDL or FO, Σ TGDs, and each k there is a tree automaton that accepts all finite k tree codes of countereexample to monotonic determinacy of $Q, \mathbf{V}, \Sigma$. There is a nondeterministic Büchi automaton over infinite trees that holds exactly when there is an arbitrary (possibly infinite) k tree code of such an instance.

The argument works for a larger logic – *Guarded Second Order Logic (GSO)*. A tuple is said to be *guarded* in an instance if there is a fact $F(c_1 \dots c_n)$ in the instance such that every component of the tuple is one of the c_i. A set of k-tuples (a k-ary relation) is said to be *guarded* in an instance if every tuple in it is guarded. Guarded Second Order logic is defined, following (Grädel, Hirsch, and Otto 2002), as a semantic restriction of SO requiring that the second order quantifiers range over guarded relations. For example, when the base schema has only a binary relation $R(x, y)$, GSO allows the usual MSO quantification and also quantification over sets of edges: thus GSO can express the existence of paths. The following is a variant of *Courcelle's theorem* (Courcelle 1990):

Theorem 20. *For every guarded second-order sentence ϕ over the signature σ and $k \in \omega$ there is a tree automaton A_ϕ over σ_k^{code} such that for all finite σ_k^{code}-trees $\mathcal{T}$ we have that A_ϕ accepts $\mathcal{T}$ iff $\mathcal{T}$ is coherent and $\mathfrak{D}(\mathcal{T}) \models \phi$.*

Proofs can be found many places. For example, see Theorem 10 of (Benedikt 2020) for an argument that matches the phrasing above.

Proof. The first part of the proposition follows immediately from Theorem 20. The statement that an instance is a counterexample involves statements that:

- The rules are satisfied on both instances. Since the rules are TGDs, they are FO, hence in GSO.
- The views predicates match their definitions. FGDL and FO are both subsets of GSO, and GSO is closed under Boolean operations and universal quantification. Thus this is expressible in GSO.
- The query holds in one instance and does not hold in another: this again is in GSO if Q is in FGDL.

- The view facts in one instance are contained in the view facts of the other instance: this is a simple universal containment, hence in FO, hence in GSO.

Thus being a counterexample can be expressed in GSO and Theorem 20 applies to it.

The second part of the theorem can be shown either by redoing the proof of Courcelle's theorem for infinite trees, or by showing definability in Weak Monadic Second Order Logic, and using the result mentioned earlier in the appendix that WMSO definability implies recognizabiilty by a nondeterministic Büchi Automata (Rabin 1970). □

We now recall Proposition 3:

If $(Q, \mathbf{V}, \Sigma)$ has the BTVIP, $\mathbf{V}$ in Datalog, then there is k, computable from $(Q, \mathbf{V}, \Sigma)$, such that whenever monotonic determinacy fails, there is e some counterexample of treewidth k.

Note that the BTVIP is only defined when Q is in Datalog and Σ are TGDs, since we need to talk about the chase of approximations.

Proof. We make use of the correctness of the process in Figure 3. By the BTVIP assumption there is k such that for each approximation fo Q, there is a finite chase of the view images with treewidth k. Let A_0 be this set of instances.

We then consider the instances A_1 obtained by adding on fresh witnesses for each view fact. Since we can use new trees for these witnesses, the treewidth of A_1 is also bounded by the maximum of k and the size of each rule body.

We now let A_2 be the result of chasing these instances chasing the view facts of these instances with the backward views. Since Σ are FGTGDs, we can perform the usual tree-like chase on it – see, for example, (Benedikt et al. 2022) or Section A.3 of this submission for a description. Note that when we chase an $\mathcal{I}_2 \in A_2$ we are adding on facts over values in $\mathcal{I}_2$, but only facts that are already guarded in $\mathcal{I}_2$. So we do not break the running intersection property of the existing decomposition of $\mathcal{I}_2$. In addition our chasing adds on new structure, possibly infinite. But the structure consists of components of bounded treewidth, interfacing with $\mathcal{I}_2$ in a guarded set (Lukasiewicz, Calì, and Gottlob 2012; Bárány, Benedikt, and ten Cate 2018). Thus we can simply union the tree decomposition of the components with the decomposition of $\mathcal{I}_2$. □

Backward mapping results. We now recall Theorem 1 from the body, which deals again with the case of finite trees. It says that if we start with a standard finite tree automaton, we can go backward to Datalog:

Let σ be a relational signature, $k \in \omega$ and A an automaton for the tree signature σ_k^{code} of 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 active tree code $\mathcal{T}$ over σ with a homomorphism from $\mathfrak{D}(\mathcal{T})$ to $\mathfrak{M}$ such that A accepts $\mathcal{T}$.

The proof of this will be more involved, but we emphasize that it is essentially the same argument as in Proposition 7.1 of (Benedikt et al. 2023).

Definition 8. *Given the σ_k^{code}-tree automaton A we define a Datalog program E_A as follows: The extensional predicates are all the predicates from σ. As intensional predicates we have the 0-ary goal predicate* GOAL*, the 1-ary predicate* ADOM*, for every bag L the k-ary predicate LOCAL_L and k-ary predicates $P_{q,g}$ for every state q of A and partial injective function $g : \{1, \ldots, k\} \to \{1, \ldots, k\}$. The program contains the following rules: For every n-ary extensional predicate R in σ it contains the rule*

$$\text{ADOM}(x_i) := R(x_1, \ldots, x_n), \quad \textit{where } i \in \{x_1, \ldots, x_n\}.$$

For every accepting state q of A and partial injective g it contains the rule

$$\text{GOAL} := P_{q,g}(x_1, \ldots, x_k).$$

For every bag L we have the rule

$$\mathsf{LOCAL}_L(x_1, \ldots, x_n) := \bigwedge_{i=1}^{n} \text{ADOM}(x_i) \wedge \bigwedge_{T^{=}_{l_0,l_1} \in L} x_{l_0} = x_{l_1} \wedge \bigwedge_{T^{R}_{\mathbf{n}} \in L} R(x_{\mathbf{m}}),$$

where $x_{\mathbf{m}} = x_{l_1}, \ldots, x_{l_m}$ for $\mathbf{m} = (l_1, \ldots, l_m)$. For every transition of the form $\text{CODE}^g_L \to q$ in A the program E_A contains the rule

$$P_{q,g}(x_1, \ldots, x_k) := \mathsf{LOCAL}_L(x_1, \ldots, x_k).$$

For every transition of the form $q_1, \ldots, q_r, \text{CODE}^g_L \to q$ in A and for all partial injections $g^1, \ldots, g^r$ it contains the rule

$$P_{q,g}(x_1, \ldots, x_k) := \bigwedge_{j=1}^{r} \left[P_{q_j,g^j}(x^j_1, \ldots, x^j_k) \wedge \bigwedge_{l \in \text{DOM}(g^j)} x_l = x^j_{g^j(l)} \right] \wedge \mathsf{LOCAL}_L(x_1, \ldots, x_k). \tag{1}$$

We refer to the Datalog query E_A as the *backward mapping Datalog query*, and the rules as *backward mapped rules*.

Lemma 1. *If $\mathfrak{M} \models E_A$ then there is an active tree code $\mathcal{T}$ with a homomorphism from $\mathfrak{D}(\mathcal{T})$ to $\mathfrak{M}$ such that A accepts $\mathcal{T}$.*

Proof. Assume that $\mathfrak{M} \models E_A$. We define the tree $\mathcal{T}$ by induction on the evaluation tree that captures that witnesses $\mathfrak{M} \models E_A$. We show that every intensional predicate of the form $P_{q,g}(x_1, \ldots, x_k)$ that holds of some tuple $(s_1, \ldots, s_k)$ of elements of $\mathfrak{M}$ there is a σ_k^{code}-tree $\mathcal{T}$ together with a run of A that ends at the root v of $\mathcal{T}$ in state q and a homomorphism h from $\mathfrak{D}(\mathcal{T})$ to $\mathfrak{M}$ such that $h([v, l]) = s_i$ for all $l \in \{1, \ldots, k\}$. This yields the desired result, because GOAL can only be true if $P_{q,g}(x_1, \ldots, x_k)$ for an accepting state q is true of some elements in $\mathfrak{M}$.

In the base case $P_{q,g}(x_1, \ldots, x_k)$ holds of the tuple $(s_1, \ldots, s_k)$ because of the rule $P_{q,g}(x_1, \ldots, x_k) := \mathsf{LOCAL}_L(x_1, \ldots, x_k)$ for some leaf transition $\text{CODE}_L^q \to q$. In this case we define $\mathcal{T}$ to consist of just one leaf v that is labelled with CODE_L^q. Because of the transition $\text{CODE}_L^q \to q$ there is a run of A on $\mathcal{T}$ that ends in q. This definition of $\mathcal{T}$ means that the structure $\mathfrak{D}(\mathcal{T})$ has as its domain the elements $[v, 1], \ldots, [v, k]$ and a predicate R from σ is defined to be true of $([v, l_1], \ldots, [v, l_m])$ iff $T_{\mathbf{n}}^L \in L$ for $\mathbf{n} = (l_1, \ldots, l_m)$. We define the homomorphism h such that $h([v, l]) = s_l$ for all $l \in \{1, \ldots, k\}$. This is well-defined as a function because whenever $[v, l] = [v, l']$ then this is because there is the predicate $T_{l,l'}^{=}$ in the label CODE_L^q of v and hence $s_l = s_{l'}$ holds, because $\mathsf{LOCAL}_L(x_1, \ldots, x_k)$ is true of $s_1, \ldots, s_k$. The function h preserves all the facts in $\mathfrak{D}(\mathcal{T})$ because the last conjunct of LOCAL_L requires the facts from L to hold at $(s_1, \ldots, s_k)$ in $\mathfrak{M}$.

In the inductive case $P_{q,g}(x_1, \ldots, x_k)$ holds of the tuple $(s_1, \ldots, s_k)$ because of the rule from (1), which is in the program E_A because of the transition $q_1, \ldots, q_r, \text{CODE}_L^g \to q$ in A. We then have that for every $j \in \{1, \ldots, r\}$ there is some partial injection g^j such that the predicate $P_{q_j,g^j}(x_1^j, \ldots, x_k^j)$ holds of some elements $s_1^j, \ldots, s_k^j$ in $\mathfrak{M}$. By the inductive hypothesis we obtain for every j a tree code $\mathcal{T}_j$ and a homomorphism h_j from $\mathfrak{D}(\mathcal{T}_j)$ to $\mathfrak{M}$ such that A has a run in $\mathcal{T}_j$ ending with q_j and $h_j([v^j, l]) = s_l^j$ for all $l \in \{1, \ldots, k\}$. We define $\mathcal{T}$ such that it consists of the disjoint union of the trees $\mathcal{T}_j$ for all $j \in \{1, \ldots, r\}$ together with a new root v that is labelled with CODE_L^g and has all the roots v_j of the $\mathcal{T}_j$ as its children. Note that with this construction we have that $[v, l] = [v_j, g^j(l)]$ holds in $\mathfrak{D}(\mathcal{T})$ for all $j \in \{1, \ldots, r\}$ and $l \in \text{DOM}(g^j)$. There is a run of A in $\mathcal{T}$ ending at q because we can combine all the runs in the $\mathcal{T}_j$ ending in q_j and use the transition $q_1, \ldots, q_r, \text{CODE}_L^g \to q$. The homomorphism h from $\mathfrak{D}(\mathcal{T})$ to $\mathfrak{M}$ is defined such that $h([v, l]) = s_l$ for all $l \in \{1, \ldots, k\}$ and for $w \in \mathcal{T}_j$ we set $h([w, l]) = h_j([w, l])$. To show that this is well-defined one uses the observation that whenever some $[w, l]$ for $w \in \mathcal{T}_j$ is equal to either some $[v, l']$ or some $[w', l']$ for $w' \in \mathcal{T}_{j'}$ with $j \neq j'$ then $[w, l] = [v_j, l'']$ for some local name l'' such that $l'' = g^j(l')$ for some $l' \in \text{DOM}(g^j)$. Then one exploits the equality $x_{l'} = x^j_{g^j(l')}$ in (1). For all equality that are enforced on local names at v we use the corresponding conjunction in the body of $\mathsf{LOCAL}_L(x_1, \ldots, x_k)$, which holds of $s_1, \ldots, s_k$ Our definition yields a homomorphism for the facts in the bag of v because of the corresponding conjunct in the body of $\mathsf{LOCAL}_l(x_1, \ldots, x_k)$. □

Definition 9. *Let $\mathcal{T}$ be an active tree code and fix an element b in the active domain of $\mathfrak{D}(\mathcal{T})$. For every v in the tree $\mathcal{T}$ we define a k-tuple $a^v = (a_1, \ldots, a_k)$ of elements of $\mathfrak{D}(\mathcal{T})$ such that $a_l = [v, l]$, if $[v, l]$ is in the active domain of $\mathfrak{D}(\mathcal{T})$, and $a_l = b$ otherwise.*

Lemma 2. *For every node v in an active tree code $\mathcal{T}$ and bag L the predicate $\mathsf{LOCAL}_L(x_1, \ldots, x_k)$ is true of a^v in $\mathfrak{D}(\mathcal{T})$.*

Proof. We show that the body of

$$\mathsf{LOCAL}_L(x_1, \ldots, x_n) := \bigwedge_{i=1}^{n} \text{ADOM}(x_i) \wedge \bigwedge_{T_{l_0,l_1}^{=} \in L} x_{l_0} = x_{l_1} \wedge \bigwedge_{T_{\mathbf{n}}^R \in L} R(x_{\mathbf{n}}),$$

is true of $a^v = (a_0, \ldots, a_k)$. By the statement of the lemma it is guaranteed that every a_l for $l \in \{1, \ldots, k\}$ is in the active domain of $\mathfrak{D}(\mathcal{T})$. Therefore, the first conjunct of the body holds. That the other conjuncts about equality and the atomic relation symbols hold follows immediately from the definition of $\mathfrak{D}(\mathcal{T})$. □

Lemma 3. *If the active tree code $\mathcal{T}$ is accepted by A then $\mathfrak{D}(\mathcal{T}) \models E_A$.*

Proof. To prove that $\mathfrak{D}(\mathcal{T}) \models E_A$, we show with an induction on the depth of a run that if there is a run of A ending at some node v of $\mathcal{T}$ in a state q of the automaton then for every partial injection g the predicate $P_{q,g}(x_1, \ldots, x_k)$ is true of a^v in $\mathfrak{D}(\mathcal{T})$. Because of the rule $\text{GOAL} := P_{q,g}(x_1, \ldots, x_k)$, for accepting q, this then means that if there is an accepting run of A at the root of $\mathcal{T}$ then E_A is true in $\mathfrak{D}(\mathcal{T})$.

In the base case of the induction we have a run that ends at a leaf v of $\mathcal{T}$ and corresponds to a transition of the form $\text{CODE}_L^g \to q$ in A. It then follows immediately that $P_{q,g}(x_1, \ldots, x_k)$ is true of a^v because by Lemma 2 the predicate $\mathsf{LOCAL}_L(x_1, \ldots, x_k)$ is true of a_v and the program E_A contains the rule $P_{q,g}(x_1, \ldots, x_k) := \mathsf{LOCAL}_L(x_1, \ldots, x_k)$.

In the inductive case we have a run ending in a state q and at a node v of $\mathcal{T}$ that is not a leaf. We need to establish that for all partial injections g the predicate $P_{g,q}(x_1, \ldots, x_k)$ is true of a^v in $\mathfrak{D}(\mathcal{T})$. Consider the last transition of the run that ends at v, which must be of the form $q_1, \ldots, q_r, \text{CODE}_L^g \to q$. Thus, for each of the children $v_1, \ldots, v_r$ of v in $\mathcal{T}$ we have that there are accepting runs of A at v_j ending in state q_j. By the induction hypothesis we have that for each j and any partial injection g' that the predicate $P_{q_j,g'}(x_1^j, \ldots, x_k^j)$ is true of a^{v_j} in $\mathfrak{D}(\mathcal{T})$. In particular this holds for the partial injection g^j that is at the label

$\text{CODE}_{L_j}^{g^j}$ of the child v_j, meaning that the predicate $P_{q_j,g'}(x_1^j, \ldots, x_k^j)$ is true of a^{v_j}. Moreover, by the construction of $\mathfrak{D}(\mathcal{T})$ we have that $[v, l] = [v_j, g^j(l)]$ for all $l \in \text{DOM}(g^j)$. Therefore, if $[v, l]$ is in the active domain of $\mathfrak{D}(\mathcal{T})$ then so is $[v^j, g^j(l)]$ because it is the same element. It follows that in this case the equality $x_l = x^j_{g^j(l)}$ holds. If on the other hand $[v, l]$ is not in the active domain of $\mathfrak{D}(\mathcal{T})$ then $[v_j, g^j(l)]$ is also not in the active domain and the equality $x_l = x^j_{g^j(l)}$ holds because both variables are interpreted as b. Our reasoning so far shows that for every j the first conjunct in the body of (1) holds. For the second conjunct, $\mathsf{LOCAL}_L(x_1, \ldots, x_k)$, we can use Lemma 2. □

Proof of Theorem 1. Let E_A be defined from A as in Definition 8.

The direction from left to right is proven in Lemma 1.

For the other direction assume that there is a tree code $\mathcal{T}$ such that A accepts $\mathcal{T}$ and there is a homomorphism from $\mathfrak{D}(\mathcal{T})$ to $\mathfrak{M}$. By Lemma 3 it follows that $\mathfrak{D}(\mathcal{T}) \models E_A$. Because Datalog is preserved under homomorphisms it follows that $\mathfrak{M} \models E_A$. □

B.2 Proofs of Certain Answer Rewriting results

A note on finite and infinite. We note that in the notion of certain answer rewriting, we are quantifying twice over instances or finite instances. In the usual definition of rewriting, we say R is a certain answer rewriting of Q w.r.t. Σ if:

for all finite instances $\mathcal{I}$, $R(\mathcal{I})$ holds if and only if for every $\mathcal{I}'$ extending $\mathcal{I}$ with $\mathcal{I}'$ satisfying Σ, $\mathcal{I}'$ satisfies Q.

The second quantification over instances $\mathcal{I}'$ cannot in general be exchanged with a quantification over finite instances: this exchange is valid when the rules are finitely controllable.

But in the first quantification, the distinction between finite instances and infinite instances will usually not be important to us:

Proposition 13. *If Σ is a set of existential rules, and Q and R are in Datalog, "finite instances" can be exchanged with "instances" without impacting the definition.*

Proof. Clearly if we have the equivalence for all instances, we have it for finite instances. Conversely, suppose we have an infinite $\mathcal{I}$ where it fails. One possibility for failure is that R holds on $\mathcal{I}$ but the right hand implication does not hold. But then the is some finite $\mathcal{I}_0$ subinstance of $\mathcal{I}$ where R holds, and the right hand implication follows,. The other possibility for failure is that the implication on the right holds for some infinite instance $\mathcal{I}$, but R fails to hold on $\mathcal{I}$. But then $\text{CHASE}_\Sigma(\mathcal{I})$ satisfies Q, since the chase characterization still holds for Datalog Q and infinite instances. Some approximation Q_n of Q witnesses this, and only a finite set of facts $\mathcal{I}_0$ from $\mathcal{I}$ where needed to generate the match of Q_n in the chase. Thus $\mathcal{I}_0$ is a counterexample of the statement for finite instances. □

We will make use of this propositions in many of the arguments that follow, since we will apply the certain answer rewritings within pipelines of transformations that produce infinite instances.

We now turn to proving the results on certain answer rewriting from Section 3 in the body of the paper.

Recall the statement of Theorem 3:

CQs have MDL C.A. REWRITINGs over frontier-one TGDs.

Sketch. The rewriting algorithm follows the approach in Theorem 5.5 and 5.6 of (Bárány, Benedikt, and ten Cate 2018), where the same result is shown for guarded and frontier-guarded TGDs, with the resulting rewriting being in frontier-guarded Datalog.

Let Σ be the rules. We can assume that our Boolean query Q consists of the atomic 0-ary predicate GOAL: if it is not, we add a new TGD to Σ of the form $Q \rightarrow \text{GOAL}$

For simplicity we deal with the case where there are no constants in the query or in the rules. Let us refer to a fact over our initial instance $\mathcal{I}$ as a "ground fact". We can see, arguing from the chase, that for any $\mathcal{I}$, ground facts that are inferred by Σ are "unary": that is, they have only one element in them, like $U(c)$ or $R(c, c)$.

Say that an instance $\mathcal{I}$ is *fact saturated with respect to* Σ if every ground fact that is derived from $\mathcal{I}$ and Σ is already in $\mathcal{I}$.

In deriving a new ground fact, from Σ, we may make use of an unbounded number of facts in $\mathcal{I}$: think of a graph $G(x, y)$ and Σ with reachability rules, of the form s $Reaches(x) \wedge G(x, y) \rightarrow Reaches(y)$. To derive $Reach(a)$ we may use many G facts. However, to achieve fact saturation with respect to frontier-guarded rules, it suffices to ensure saturation for small sets. In the above example, once we know that the facts on each pair of nodes cannot generate any new facts, we know we are fact-saturated. This is capture in the "bounded base lemma", Lemma 5.8 of (Bárány, Benedikt, and ten Cate 2018)

Lemma 4. *(Bárány, Benedikt, and ten Cate 2018) Letting k be the maximal number of variables in a TGD of Σ, and let A be an instance such that for each subset X of the active domain of A with $|X| \leq k$, the induced substructure $A \restriction$ is fact-saturated with respect to Σ Then A is fact-saturated with respect to Σ.*

We now create a Datalog program P_Σ that infers all ground facts. We include:

- intensional predicates U_A for each atom with a single free variable x, along with "copy rules" of the form $U_A(x) := A(x)$. Similarly for every 0-ary predicate in the signature.
- all Datalog rules over the signature with the extensional predicates and the intensional predicates above, with at most k atoms in the body and a unary or 0-ary predicate U_A in the head, such that the rules follow from Σ and the copy rules. We can decide whether such a rule is derived using a decision procedure for the appropriate guarded logic, for example the guarded negation fragment of first order logic (GNFO).
- U_{GOAL} as the goal predicate

Clearly, the program only derives $U_A(x)$ when $A(x)$ is derivable from Σ. To show the converse, it suffice to show that for any instance $\mathcal{I}$ after running the Datalog program until a fixpoint is reached, and applying the "reverse copy rule" $U_A(x) \rightarrow A(x)$, the resulting instance is fact-saturated with respect to Σ. By the lemma, it suffices to show that for each induced subinstance $A_k = a_1 \dots a_k$ with domain of size k, A_k is fact-saturated with respect to Σ and the reverse rules. So fix A_k and a fact F that is derived from A_k using Σ and the reverse rules. Clearly, the reverse rules are only useful to derive F at the end, since we can replace any use of A by U_A in a derivation. But $A_k \rightarrow F$ is a consequence of Σ. Thus $A_k \rightarrow U_F$ will be in P_Σ, and thus F will be derived from P_Σ and the reverse rules. We conclude A_k is fact-saturated, and we are done. □

Recall the statement of Proposition 4:

For any set Σ be a set of frontier-guarded TGDs and any Boolean FGDL query Q there is a Boolean FGDL query that is a certain answer rewriting of Q $w.r.t.$ Σ.

Proof. As mentioned earlier, the corresponding assertion for a CQ query is in (Bárány, Benedikt, and ten Cate 2018). We reduce to this case, by moving the rules of FGDL Q into additional rules of Σ, arriving at set of rules Σ', and letting Q' be just GOAL. Note that since the rules of Q are frontier-guarded, the rules Σ' are all frontier-guarded TGDs. We claim that for any instance $\mathcal{I}$,

$$\mathcal{I} \wedge \Sigma \models Q \leftrightarrow \mathcal{I} \wedge \Sigma' \models \text{GOAL}$$

Where $\mathcal{I}$ refers to the conjunction of facts in the instance.

The equivalence is a standard observation about entailment in Datalog and entailment in Horn clause logic. Note that the right hand entailment asserts that for any $\mathcal{I}$, and an arbitrary superinstance of $\mathcal{I}$ satisfying Σ and satisfying the rules of Q as Horn clauses, GOAL holds. The left hand entailment asserts that for every instance $\mathcal{I}$ satisfying Σ, the instance $\mathcal{I}'$ that is the least fixed point of the rules satisfies GOAL. Thus for the right to left direction, we just note that $\mathcal{I}'$ is a superinstance of a structure satisfying Σ which satisfies the rules, and thus the right hand entailment implies to conclude GOAL. For the other direction, suppose we have a counterexample to the right hand entailment, consisting of $\mathcal{I}$ satisfying Σ and a superinstance $\mathcal{I}^*$ satisfying the rules of Q, where $\neg$GOAL holds in $\mathcal{I}^*$. Since $\mathcal{I}^*$ is a fixpoint, setting $\mathcal{I}'$ to be the least fixed point of $\mathcal{I}$ under the rules. Since we have only removed facts, $\neg$GOAL still holds. But then the left is contradicted. □

B.3 Proof of Theorem 4: Finite controllability of EC Datalog under Frontier One TGDs

In this section we present the proof of Theorem 4:

For the class of frontier-one TGDs Σ and EC-Datalog queries $Q = (\Pi, \text{GOAL})$ entailment is finitely controllable.

For the remainder of this section, 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 $N = 4 \cdot n^2$.[2]

In the upcoming section, we will introduce a few minor assumptions. Near the conclusion of the proof, we will discuss how to remove these assumptions.

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.

In this section, we shall disregard the direction of binary predicates whenever discussing graph-theoretic notions, such as distances, trees, or cycles.

Proposition 14. *The chase of $\mathcal{D}$ by Σ forms a regular tree.*

[2]The constant 4 is chosen here for convenience and is not in any way considered to be a *tight* value.

Proof. From Assumptions 1 to 5. □

Definition 10 (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 in the above we treat instances as undirected graphs.

Definition 11 (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 12 (Unfolding tree)**.** *For a Datalog program P that lacks repeated variables in its head, the* unfolding tree *is defined as 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 13 (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.*

Proposition 15. *Let U be an unfolding of Q. Let x and y be distinct variables of U. Let p be a path in U using binary EDB predicates, and suppose p connects x and y. Then*

$$|p| \leq n \cdot dist_U(x, y), \tag{♣}$$

Proof. First, note the following:

($\diamondsuit$) For any three variables x, y, and z, and two paths p (from x to y) and p' (from y to z), if both x, y, p and y, z, p' satisfy Eq. (♣), then x, y, and the concatenation of p and p' also satisfy Eq. (♣).

($\heartsuit$) Bodies of rules of Q satisfy Eq. (♣).

We will prove Proposition 15 by a simple inductive argument.

(base) Let U be a single-atom unfolding - it trivially satisfies Eq. (♣).

(ind. step) Let U' be any direct predecessor of U and let $U \wedge \alpha = U'$ where α is a conjunction of atoms. Let $\boldsymbol{x}$ be the tuple of variables shared between U and U'. Note that α satisfies Eq. (♣) from ($\heartsuit$) and U' satisfies Eq. (♣) by ind. hypothesis.

1) If $\boldsymbol{x}$ is empty then U trivially satisfies Eq. (♣).

2) If $\boldsymbol{x}$ consists of one or two variables x then U satisfies Eq. (♣) from ($\diamondsuit$)

□

Lemma 5. *Every unfolding of Q contains no unabridged cycles of length greater than or equal to N when restricted to EDB relations.*

Proof. We will use Proposition 15 to prove the lemma. Assuming, for the sake of contradiction, that the statement above does not hold, let U be the earliest counterexample — meaning no unfolding that is a predecessor of U is a counterexample.

Let U' be any direct predecessor of U and let $U' \wedge \alpha = U$ where α is a conjunction of atoms. Note, as U' is not a counterexample, it contains no large unabridged cycle over EDB relations. However, U does contain such a cycle C. Thus, there exist two distinct variables x and y shared between U' and α such that C passes through x and y in U. Let p be the path over EDB relations connecting x and y in U'. Note that $|p| \geq N - |\alpha|$ and $dist_U(x, y) \leq |\alpha| \leq n$. Together with Proposition 15 we get a contradiction: $4 \cdot n^2 - n \leq |p| \leq n \cdot dist_U(x, y) \leq n^2$. □

In order to prove Theorem 4 we shall show the following:

Lemma 6. *If $\mathcal{C}$ does not entail Q then there exists a finite model M of Σ that does not entail Q as well.*

The proof of the above lemma is the heart of the argument, and will require some machinery.

Definition 14 (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 15 (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:

- The depth of u in $\mathcal{C}$ modulo N.
- The N-perspective of u. Note that Proposition 14 indicates there is only a finite number of such, 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.

Proposition 16. *M is a model of Σ.*

Proof. Note that N-distance neighborhoods of elements of $\mathcal{C}$ before and after identification are homomorphically equivalent to each other. Thus for any element of $\mathcal{C}$ the satisfaction of unary CQs of size smaller than N is identical before and after taking the quotient. Thus the result follows from the fact that Σ is frontier-one, as trigger activation relies on satisfaction of unary CQs of size no greater than n. □

Proposition 17. *Any instance that can be mapped to M but not to $\mathcal{C}$ contains an unabridged cycle of length at least N.*

Proof. Claim: M does not contain an unabridged cycle of length smaller than N.

Assume, towards contradiction, that M contains an unabridged cycle of length smaller than N. Let C be the shortest such cycle with a length of m, and let $\mathcal{P}$ be a sequence of simple paths in $\mathcal{C}$ that demonstrates the existence of C. Note that each vertex appears only once among the paths in $\mathcal{P}$.

Assume that $\mathcal{P}$ contains a single element, P. Let u and u' be, respectively, the first and last vertices of P. Note that, as P is simple and shorter than N, and the depth of u and u' modulo N is equal, there exists another vertex, v, in P that is the common ancestor of both u and u'. Let w and w' be children of v on paths toward u and u'. Observe that w and w' have equal types, and thus are identified in M. Therefore, C is not a simple cycle, leading to a contradiction.

Assume that $\mathcal{P}$ contains more than one path, and let P and P' be its two first elements. We will show that there exists $\mathcal{P}'$ containing one less path than $\mathcal{P}$ which is witnessing the existence of another unabridged cycle of length m. Let u be the last element of P, and let u' be the first element of P'. As u and u' are identified, one can find a path equivalent to P' in the N-perspective of u that can be used instead. Thus, using a simple inductive reasoning, one can reduce this case to the above.

Finally, note that any instance that contains no cycle and can be mapped to M can also be mapped to $\mathcal{C}$. Using the above claim, one can conclude the proposition. □

Now, we are ready to prove Lemma 6. Assume, for the sake of contradiction, that M entails Q, and let U be an unfolding of Q that serves as a witness. From Proposition 17, we know that $U|_{EDB}$ contains an unabridged cycle of length at least N. From Lemma 5 we get a contradiction.

On lifting assumptions. The assumptions can be lifted using the following pre-processing steps.

1. The database instance $\mathcal{D}$ is a single unary atom:
The database can be saturated $\mathcal{D}$ under Datalog rules first. Then, each constant can be placed into a unique unary atom. With some trivial but necessary adjustments to Σ, one can consider each database constant separately.

2. The EDB relational symbols are at most binary:
This can be achieved through a simple reification. For example, the atom $R(x, y, z)$ can be transformed into the following conjunction:

$$R_1(w, x) \wedge R_2(w, y) \wedge R_3(w, z).$$

3. The rules of Σ have exactly one frontier term:
Trees built over atoms resulting from the application of frontierless rules can be considered separately, as in step 1.

4. No atoms of the shape $E(x, x)$ appear in $\mathcal{D}, \Sigma$, and Q:
Similarly to step 2, one can replace $E(x, x)$ with $E_{loop}(x)$ in every atom of the input. Again, a trivial but necessary surgery of the ruleset and the Datalog query might be required.

5. Heads of rules of Σ are trees:
This can be ensured during reification in step 2.

B.4 Proof of Theorem 5: compactness of entailment for views and rules

Recall the statement of Theorem 5 [Compactness for view and rule entailment]:

Let Σ consist of frontier-guarded TGDs, $\mathbf{V}$ a set of views defined by Datalog queries, and Q a FGDL query.

For every $\mathcal{J}$ such that $\mathcal{J}$ is Q-entailing with respect to Σ, $\mathbf{V}$, there is $\mathcal{J}_0$ a finite subinstance of $\mathcal{J}$ that is Q-entailing with respect to Σ, $\mathbf{V}$.

We write $\mathcal{I} \equiv_k \mathcal{I}'$ if Duplicator has a winning strategy in the EF-game over k rounds for guarded second order logic. In every round of this game Spoiler can choose to either pick an element or a guarded set in one of the instances. A guarded set is a subset R of D^n for some $n \geq 1$, where D is the domain of the instance and every tuple in R is already in the interpretation of one of the relations symbols. Duplicator has to reply with an choice of the same type in the other instance. After k rounds Duplicator wins if the pairs of elements that have been picked in the rounds where Spoiler choice to play an element are a partial isomorphism. We clarify the notion of partial isomorphism we use. F Assume that in the match the pairs $(a_1, a'_1), \ldots, (a_h, a'_h)$ of elements have been played. We write $c^{\mathcal{I}}$ for the interpretation of some constant $c \in \mathcal{L}$ in $\mathcal{I}$. We require that the relation

$$\{(a_1, a'_1), \ldots, (a_h, a'_h)\} \cup \{(c^{\mathcal{I}}, c^{\mathcal{I}'}) \mid \text{ for some constant } c \in \mathcal{L}\}$$

is a partial isomorphism in the usual sense. We also need that the partial isomorphism preserves the guarded relations that have been chosen in any of the rounds, where Spoiler chose a guarded subset of D^n.

A basic result is that $\mathcal{I} \equiv_k \mathcal{I}'$ iff $\mathcal{I}$ and $\mathcal{I}'$ cannot be distinguished by formulas of quantifier depth at most k. Here, we do not distinguish between first and second order quantifiers when computing the quantifier depth of a formula. If the language is finite then one can show with an induction on k that, up-to logical equivalence, there are only finitely many formulas of quantifier depth k. Thus, $\equiv_k$ has a finite index.

Given two instances $\mathcal{I}$ and $\mathcal{I}'$ for $\mathcal{L}$ their *disjoint sum* $\mathcal{I} + \mathcal{I}'$ is defined as follows: Its domain is the disjoint union of the domains of $\mathcal{I}$ and $\mathcal{I}'$ quotiented by the smallest equivalence relation $\sim$ that is generated by all pairs $a_1 \sim a_2$ such that for some constant c of $\mathcal{L}$ the element a_1 is the interpretation of c in $\mathcal{I}$, and a_2 is the interpretation of c in $\mathcal{I}'$. A tuple $(u_1, \ldots, u_n)$ is in the interpretation of an n-ary relation symbol P if there representatives $a_1, \ldots, a_n$ such that a_i is in the equivalence class u_i for all i, and the elements $a_1, \ldots, a_n$ either all belong to the domain of $\mathcal{I}$ and are in the interpretation of P in $\mathcal{I}$ or they all belong to the domain of $\mathcal{I}'$ and are in the interpretation of P in $\mathcal{I}'$.

The following is a useful observation about guarded sets in the disjoint sum:

Lemma 7. *For every guarded relation R in $\mathcal{I} + \mathcal{I}'$ there are guarded relations A in $\mathcal{I}$ and B in $\mathcal{I}'$ such that $R = A \uplus B$.*

The following "composition lemma", also straightforward, says that $\equiv_k$ behaves well under disjoint union:

Lemma 8. *If $\mathcal{I} \equiv_k \mathcal{I}'$ and $\mathcal{J} \equiv_k \mathcal{J}'$ then $(\mathcal{I} + \mathcal{J}) \equiv_k (\mathcal{I}' + \mathcal{J}')$.*

We now show that we can without loss of generality consider UCQ views rather than Datalog views:

Lemma 9. *Let $V = \{V_1, \ldots, V_n\}$ be a set of Datalog views and ϕ a GSO-sentence. Then there is a set $V' = \{V_1', \ldots, V_n'\}$ of UCQ views for the same view schema as V such that for every instance $\mathcal{J}$ over the view schema*

$$\mathcal{I} \models \phi \text{ holds for all } \mathcal{I} \in \mathrm{Back}_V(\mathcal{J}) \quad \textit{iff} \quad \mathcal{I} \models \phi \text{ holds for all } \mathcal{I} \in \mathrm{Back}_{V'}(\mathcal{J}). \tag{2}$$

Proof. Let k be the quantifier depth of Q, thought of as a GSO formula. For every view V_i with answer variables $x_1, \ldots, x_m$ we let $\mathbf{S}_i$ be the same as the schema $\mathbf{S}$ of ϕ, but with added constants $c_1, \ldots, c_m$. Every approximation $\mathcal{U}$ of the Datalog program for V_i can then be considered as an instance for the schema $\mathcal{L}_i$ in which the constants $c_1, \ldots, c_n$ denote answer variables $x_1, \ldots, x_m$ at the root of $\mathcal{U}$. We partition the set of all approximations of V_j into its $\equiv_k$-equivalence classes for the schema $\mathcal{L}_i$. Because the index of $\equiv_k$ is finite we can choose finitely many representatives $\mathcal{R}_{i,1}, \ldots, \mathcal{R}_{i,l_i}$ of all the $\equiv_k$-equivalence classes of approximants of V_i. Recall that the *canonical CQ* of an instance is the CQ whose variables are the elements of $\mathcal{I}$, with atoms for each fact of the instance. We define the UCQ view V_i' to be the UCQ that is the disjunction of all the canonical CQs of the instances $\mathcal{R}_{i,1}, \ldots, \mathcal{R}_{i,l_i}$, where the constants $c_1, \ldots, c_m$ become the answer variables.

The right-to-left direction of (2) holds because every $\mathcal{I} \in \mathrm{Back}_{V'}(\mathcal{J})$ also exists in $\mathrm{Back}_V(\mathcal{J})$. The reason is that every CQ component of the UCQ view V_i' is also a CQ approximation of the Datalog view V_i.

For the direction from left to right we show that for every $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J})$ there is an instance $\mathcal{I}' \in \mathrm{Back}_{V'}(\mathcal{J})$ such that $\mathcal{I} \equiv_k \mathcal{I}'$ holds for the schema $\mathbf{S}$. For a fact $F = V_i(\boldsymbol{c})$ over the view schema where the views are defined using Datalog query Q_V, we talk about an *approximation* of the fact, meaning $Q_i(\boldsymbol{c})$ for Q_i a CQ approximation of Q_V. The idea is to replace the approximation of any view fact in $\mathcal{I}$ with the corresponding representative, and then use Lemma 8 to argue that this replacement preserves equivalence. To make this precise we use an induction over an enumeration $F_1, \ldots, F_h$ of all the facts in $\mathcal{J}$. We show by induction over $j = 0, 1, \ldots, h$ that there is an $\mathcal{I}_j$ such that

1. $\mathcal{I}_j \equiv_k \mathcal{I}$,
2. $\mathcal{I}_j$ contains approximations of the facts $F_1, \ldots, F_j$ of $\mathcal{J}$ according to the views in V', and
3. $\mathcal{I}_j$ contains approximations of the facts $F_{j+1}, \ldots, F_h$ from $\mathcal{J}$ according to the views in V.
4. $\mathcal{I}_j$ does not contain any facts that are not required by the previous two items.

From the second and third item it is clear that $\mathcal{I}_h \in \mathrm{Back}_{V'}(\mathcal{J})$ and thus we can then take $\mathcal{I}' = \mathcal{I}_h$. In the base case of the induction we let $\mathcal{I}_0 = \mathcal{I}$.

In the inductive step we assume that we are given the instance $\mathcal{I}_j$, satisfying the three items above. We need to construct $\mathcal{I}_{j+1}$, that satisfies the same items with $j+1$ in place of j. Thus, consider the fact F_{j+1} in $\mathcal{J}$. It corresponds to one of the views V_i in V and by the third item $\mathcal{I}_j$ contains an approximation $\mathcal{U}$ of the Datalog program V_i as the approximation of F_{j+1}. Consider $\mathcal{I}_j$ as an instance over the extended schema $\mathbf{S}_i$, where the constants $c_1, \ldots, c_m$ denote the elements at the root of $\mathcal{U}$ that correspond to the answer variables in V_i and to the constants in the fact F_{j+1}. It is clear that we can then view $\mathcal{I}_j$ as a sum $\mathcal{U} + \mathcal{I}^-$ for the schema that contains these additional constants. Here, $\mathcal{I}^-$ contains all the facts of $\mathcal{I}_j$ that are not in the approximation $\mathcal{U}$ of F_{j+1}. Let $\mathcal{U}_{i,u}$ be the representative of the $\equiv_k$ equivalence class of $\mathcal{U}$ that was chose above for the definition of the view V_i'. We let $\mathcal{I}_{j+1} = \mathcal{U}_{i,u} + \mathcal{I}^-$. Thus $\mathcal{I}_{j+1}$ is just like $\mathcal{I}_j$, but the fact F_{j+1} has been unfolded to $\mathcal{U}_{i,u}$ instead of $\mathcal{U}$. By the choice of $\mathcal{U}_{i,u}$ we have that $\mathcal{U}_{i,u} \equiv_k \mathcal{U}$ and thus it follows by Lemma 8 that

$$\mathcal{I}_{j+1} = (\mathcal{U}_{i,u} + \mathcal{I}^-) \equiv_k (\mathcal{U} + \mathcal{I}^-) = \mathcal{I}_j \equiv_k \mathcal{I}.$$

Note that here by Lemma 8 the first equivalence $\equiv_k$ holds for the schema $\mathbf{S}_i$ of the view fact F_{j+1}. But since $\mathbf{S}_i$ extends $\mathbf{S}$ with constants, the equivalence also holds if we consider the instances as instances for the reduced schema $\mathbf{S}$ without these constants. □

We now finish the proof of Theorem 5 by handling the case of UCQ views:

Lemma 10. *Assume that V consists of UCQ views. Let Q be some Datalog query and consider a countably infinite instance $\mathcal{J}$ over view schema such that $\mathcal{I} \models Q$ for all $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J})$. Then there is a finite $\mathcal{J}^- \subseteq \mathcal{J}$ such that $\mathcal{I} \models Q$ for all $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J}^-)$.*

Proof. Let $F_0, F_1, \ldots$ be an enumeration of all the atomic facts in $\mathcal{J}$. For every $n \in \omega$ define the finite subinstance $\mathcal{J}_n \subseteq \mathcal{J}$ to consist of just the facts $F_0, F_1, \ldots, F_{n-1}$. We claim that there is some n such that $\mathcal{I} \models Q$ holds for all $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J}_n)$. Thus, we can then set $\mathcal{J}^- = \mathcal{J}_n$.

Define an infinite, but finitely branching, tree T, whose vertices are pairs $(n, \mathcal{I})$ such that $n \in \omega$ and $\mathcal{I}$ is a finite instance over the original schema. The root is $(0, \emptyset)$, and for every node $(n, \mathcal{I})$ that is already in the tree we add children as follows: Consider the fact $F_n = V_i(c_1, \ldots, c_m)$ in $\mathcal{I}$. For every CQ U in UCQ defining the view V_i we add a child $(n, \mathcal{I}')$ to the tree, where $\mathcal{I}'$ is the extensions of $\mathcal{I}$ with the canonical database of U, identifying the constants $c_1, \ldots, c_m$ in $\mathcal{I}$ with the answer variables of U.

It follows from our definition of the tree T that

$$\mathrm{Back}_V(\mathcal{J}_n) = \{\mathcal{I} \mid (n, \mathcal{I}) \in T\}.$$

Moreover, it also follows from the definitions that

$$\mathrm{Back}_V(\mathcal{J}) = \{\mathcal{I}_\beta \mid \beta \text{ infinite branch in } T\},$$

where $\mathcal{I}_\beta = \bigcup\{\mathcal{I} \mid \beta_n = (n, \mathcal{I}) \text{ for some } n\}$.

To prove the claim assume for a contradiction that for every n there is some $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J}_n)$ such that $\mathcal{I} \not\models Q$. Consider the subtree T' of T consisting only of nodes $(n, \mathcal{I})$ such that $\mathcal{I} \not\models Q$. This is a connected subtree because if $(n, \mathcal{I}_p)$ is a parent of $(n+1, \mathcal{I}_c)$ then $\mathcal{I}_p \subseteq \mathcal{I}_c$ and thus $\mathcal{I}_c \not\models Q$ implies $\mathcal{I}_p \not\models Q$. By our assumption that for every n there is some $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J}_n)$ such that $\mathcal{I} \not\models Q$ it follows that T' is infinite. It follows from König's Lemma that T' contains an infinite branch β. Since $\mathcal{I}_\beta \in \mathrm{Back}_V(\mathcal{J})$ we have that $\mathcal{I}_\beta \models Q$. Because Q is a Datalog query there are finitely many facts in $\mathcal{I}_\beta$ that are sufficient for Q to hold in $\mathcal{I}_\beta$. This entails that $\mathcal{I} \models Q$ holds already for a finite $\mathcal{I}$ such that $\beta_n = (n, \mathcal{I})$ for some n. This contradicts our definition of T'. □

Because every FGDL query is expressible in GSO we can use Lemma 9 to strengthen the result from the previous lemma.

Corollary 1. *Assume that V consists of Datalog views and that Q is some* FGDL *query. Consider a countably infinite instance $\mathcal{J}$ such that $\mathcal{I} \models Q$ for all $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J})$. Then there is a finite $\mathcal{J}^- \subseteq \mathcal{J}$ such that $\mathcal{I} \models Q$ for all $\mathcal{I} \in \mathrm{Back}_V(\mathcal{J}^-)$.*

B.5 Proofs related to Theorem 6: automata for entailment with views and rules over bounded treewidth structures

Recall the statement of Theorem 6 [Automata for entailing witnesses]

Let Σ consist of frontier-guarded TGDs, $\mathbf{V}$ a set of views defined by FGDL Datalog queries, and Q a FGDL query. Let k be a number.

There is a tree automaton $T_{Q,\mathbf{V},\Sigma,k}$ that runs over k-tree codes in the view signature such that for every finite $\mathcal{J}_0$ of treewidth k,

$T_{Q,\mathbf{V},\Sigma,k}$ accepts a code of $\mathcal{J}_0$ if and only if $\mathcal{J}_0 \models_{\mathbf{V},\Sigma} Q$.

Note that here we refer to an ordinary tree automaton over finite trees.

Proof. By Lemma 9 it suffices to consider the case where the views in V are UCQs.

Recall that GSO refers to Guarded Second Order Logic, where we have first order quantification and also quantification over guarded sets of tuples. We are going to show a stronger statement:

For any ϕ in GSO, we can define a MSO formula ψ for the schema of $(\sigma_V)_k^{\mathrm{code}}$-trees such that for every $(\sigma_V)_k^{\mathrm{code}}$-tree $\mathcal{T}$ we have that $\mathcal{T} \models \psi$ iff $\mathcal{T}$ is a tree code and $\mathcal{I} \models \phi$ for all $\mathcal{I} \in \mathrm{Back}_V(\mathfrak{D}(\mathcal{T}))$.

To obtain the proof of Theorem 6, we apply the above to the GSO sentence $\phi = \Sigma \to Q$. We get an MSO sentence τ that checks whether all instances obtained by backward chasing the views, $\Sigma \to Q$ holds. That is:

$$\mathcal{T} \models \beta \quad \text{iff} \quad \mathcal{I} \models \phi \text{ for all } \mathcal{I} \in \mathrm{Back}_V(\mathfrak{D}(\mathcal{T})), \tag{3}$$

It is clear that there is a MSO formula τ (or an automaton) that expresses that $\mathcal{T}$ is a tree code. We then just convert $\psi = \tau \wedge \beta$ to a tree automaton, which we can do since over finite trees, finite tree automata capture MSO properties.

We now turn to proving the stronger statement. Given an instance of the view schema, we refer to an instance of the base schema in $\mathrm{Back}_V(\mathfrak{D}(\mathcal{T})$ as a *realization*. We call an instance of the The main difficulty is to express the quantification over all realizations in the right hand side of (3) with monadic quantifiers over the treecode $\mathcal{T}$. The proof strategy is to extend the

bags that are encoded at the nodes of $\mathcal{T}$ with additional local names that hold of the realizations of any of the view facts in the bag. To this aim we define a notion of extended tree codes, which are tree codes for the schema $\mathbf{S}' = \sigma_V \uplus \mathbf{S}$ that are supposed to represent the view instance $\mathfrak{D}(\mathcal{T})$, together with one of its realizations $\mathcal{I} \in \mathrm{Back}_V(\mathfrak{D}(\mathcal{T}))$. We provide more details in the paragraphs below.

For every UCQ V_i in V let a_i denote its arity. For every UCQ V_i in V let R_i be the finite set of all canonical instances of the CQs in the UCQ V_i. Each of the instances $\mathcal{R} \in R_i$ can be seen as an instance for the schema $\mathbf{S}_i$, which extends the schema $\mathbf{S}$ of ϕ with additional constants $c_1, \ldots, c_{a_i}$, that mark the positions of the answer variables of V_i.

Let b be the maximal size of any of the $\mathcal{R} \in R_i$ for any i and set $k' = k + r \cdot (k+1)^m \cdot b$, where r is the number of views in V and m is the maximal arity of any of the views in V. It is clear that for every view V_i in V, any realization instance $\mathcal{R} \in R_i$, and any tuple $\mathbf{n} = (l_1, \ldots, l_{a_i})$, there is an injective function $e^{\mathcal{R}}_{i,\mathbf{n}}$ from the domain of $\mathcal{R}$ to $\{1, \ldots, k'\}$ such that $e^{\mathcal{R}}_{i,\mathbf{n}}(c_j) = l_i$, meaning that the element corresponding to the j^{th} answer variable in $\mathcal{R}$ is mapped to the j-th element in the tuple $\mathbf{n}$. The number k' is chosen large enough to ensure that the realizations of any view facts overlap only at their answer variables. Formally, this means that for all views V_i and $V_{i'}$ in V, $\mathbf{n} = (l_1, \ldots, l_{a_i})$ and $\mathbf{n}' = (l'_1, \ldots, l'_{a_{i'}})$ such that either $V_i \neq V_{i'}$ or $\mathbf{n} \neq \mathbf{n}'$, and moreover, for all $\mathcal{R} \in R_i$, $\mathcal{R}' \in R_{i'}$, with elements a from $\mathcal{R}$ and a' from $\mathcal{R}'$ such that a is not answer variable of $\mathcal{R}$ and a' is not an answer variable of $\mathcal{R}'$, it holds that $e^{\mathcal{R}}_{i,\mathbf{n}}(a) \neq e^{\mathcal{R}'}_{i',\mathbf{n}'}(a')$.

Define an *extended tree code* to be a $(\mathbf{S}')^{\mathrm{code}}_{k'}$-tree such that:

1. The tree $\mathcal{T}$ is a $(\mathbf{S}')^{\mathrm{code}}_{k'}$-tree code
2. For every T_g that is true at a vertex v of $\mathcal{T}$ the partial injection g is such that its domain and range are both contained in $\{1, \ldots, k\}$.
3. If $T^{=}_{l,l'}$ is true at a vertex v of $\mathcal{T}$ then $l, l' \leq k$.
4. If $T^{V_i}_{\mathbf{n}}$ is true at a vertex v of $\mathcal{T}$ for some view symbol V_i from σ_V then $\mathbf{n} \in \{1, \ldots, k\}^{a_i}$, that is, $\mathbf{n}$ contains only local names from $\{1, \ldots, k\}$.
5. For every vertex v of $\mathcal{T}$ there is a function υ_v that maps any pair $(V_i, \mathbf{n})$, where $T^{V_i}_{\mathbf{n}}$ is true at a vertex v, to an element $\upsilon_v(V_i, \mathbf{n}) \in R_i$ such that
   (a) for every fact $R(b_1, \ldots, b_h)$ in $\mathcal{R} = \upsilon_v(V_i, \mathbf{n})$ such that $T^{V_i}_{\mathbf{n}}$ is true at v the predicate $T^R_{\mathbf{m}}$ is true at v for $\mathbf{m} = (e^{\mathcal{R}}_{i,\mathbf{n}}(b_1), \ldots, e^{\mathcal{R}}_{i,\mathbf{n}}(b_h))$, and
   (b) if $T^R_{\mathbf{m}}$ is true at v then there is some $T^{V_i}_{\mathbf{n}}$ that is true at v and a fact $R(b_1, \ldots, b_h)$ in $\mathcal{R} = \upsilon_v(V_i, \mathbf{n})$ such that $\mathbf{m} = (e^{\mathcal{R}}_{i,\mathbf{n}}(b_1), \ldots, e^{\mathcal{R}}_{i,\mathbf{n}}(b_h))$.

One can check that there is an MSO formula η such that $\mathcal{T} \models \eta$ iff $\mathcal{T}$ is an extended tree code.

Every $(\mathbf{S}')^{\mathrm{code}}_{k'}$-tree $\mathcal{T}$ projects to a $\mathbf{S}^{\mathrm{code}}_{k'}$-tree $\mathcal{T}\restriction_{\mathbf{S},k'}$ if we just forget all the predicates of the form $T^{V_i}_{\mathbf{n}}$ for some view relation V_i in V. Moreover, it is obvious that if $\mathcal{T}$ is a $(\mathbf{S}')^{\mathrm{code}}_{k'}$-tree code then its projection $\mathcal{T}\restriction_{\mathbf{S},k'}$ is a $\mathbf{S}^{\mathrm{code}}_{k'}$-tree code.

Similarly, every $(\mathbf{S}')^{\mathrm{code}}_{k'}$-tree $\mathcal{T}$ projects to a $(\sigma_V)^{\mathrm{code}}_{k}$-tree $\mathcal{T}\restriction_{\sigma_V,k}$. We just forget all the predicates that mention any local name $l > k$ or that are of the form $T^R_{\mathbf{n}}$ for some relation symbols R from $\mathbf{S}$. Because of items 1, 2 and 3 above we have that if $\mathcal{T}$ is an extended tree code then $\mathcal{T}\restriction_{\sigma_V,k}$ is a $\sigma_V{}^{\mathrm{code}}_{k}$-tree code.

Extended tree codes capture the backwards mapping of views. This is made precise by the following two properties of extended tree codes:

Soundness: For every extended tree code $\mathcal{T}$ there is an $\mathcal{I} \in \mathrm{Back}_V(\mathfrak{D}(\mathcal{T}\restriction_{\sigma_V,k}))$ such that $\mathcal{I} \cong \mathfrak{D}(\mathcal{T}\restriction_{\mathbf{S},k'})$.

Completeness: For every $(\sigma_V)^{\mathrm{code}}_{k}$-tree code $\mathcal{T}'$ and $\mathcal{I} \in \mathrm{Back}_V(\mathfrak{D}(\mathcal{T}'))$ there is an extended tree code $\mathcal{T}$ such that $\mathcal{T}\restriction_{\sigma_V,k} = \mathcal{T}'$ and $\mathfrak{D}(\mathcal{T}\restriction_{\mathbf{S},k'}) \cong \mathcal{I}$.

Using items 4 and 5 from the definition of extended tree codes, one can check that these properties hold.

We observe that in MSO we can evaluate GSO formulas over the models that are encoded over tree codes. Thus, there is an MSO formula ϕ' such that for all $\mathbf{S}^{\mathrm{code}}_{k'}$-tree codes $\mathcal{T}$

$$\mathcal{T} \models \phi' \quad \text{iff} \quad \mathfrak{D}(\mathcal{T}) \models \phi.$$

See the argument for the "Courcelle-like results" - Proposition 2 and Theorem 20 – elsewhere in the appendix.

Also note that the only difference between $\mathbf{S}^{\mathrm{code}}_{k'}$ and $(\mathbf{S}')^{\mathrm{code}}_{k'}$ is that the latter contains additional predicate symbols. Hence, the formula ϕ' can also be evaluated over extended tree codes $\mathcal{T}$ for which we have that

$$\mathcal{T} \models \phi' \quad \text{iff} \quad \mathfrak{D}(\mathcal{T}\restriction_{\mathbf{S},k'}) \models \phi.$$

Define the MSO formula β as follows:

$$\beta \equiv \forall Q_1 \ldots \forall Q_s(\eta \rightarrow \phi'),$$

where $Q_1, \ldots, Q_s$ are all the predicate symbols that are in $\mathbf{S}^{\mathrm{code}}_{k'}$. The idea on a $(\sigma_V)^{\mathrm{code}}_k$ tree is to consider all extensions to a $(\mathbf{S}')^{\mathrm{code}}_{k'}$-tree, which need to extended tree codes because of η, and then make sure that they all satisfy ϕ. By the soundness and completeness properties of extended tree codes it follows that this definition of β satisfies (3). □

C Additional material, related to applying the tools from Section 3 to obtain decidability and rewritability results, Section 4

C.1 More details on the proof of Theorems 7 and 8: decidability of monotonic determinacy via the BTVIP

Recall the statement of Theorem 7:

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

Proof. We give more details here. Note that similar arguments for the final applications of this theorem – to CQ views and FGDL views – are given in Theorems 9.1 and 9.2 of (Benedikt et al. 2023). The main difference here relates to the fact that infinite structures are involved.

The heart of the argument is to show that we have the BTVIP. The fact that the approximations are treelike is Proposition 9. The fact that the chase with FGTGDs gives a low treewidth structure is well-known: see, for example Proposition 10 in Section A.3, or (Benedikt et al. 2022). As mentioned earlier, the idea goes back to (Lukasiewicz, Calì, and Gottlob 2012). Applying the FGDL views simply adds facts compatible with the same tree decomposition, since the views are guarded.

Proposition 3 tells us that we can compute a k' such that it suffices to look for a k' treelike counterexample – possibly infinite – to monotonic determinacy.

Proposition 2 tells us that we can compute a Büchi Automaton (or a WMSO sentence) that represents these counterexamples. So it suffices to check whether this sentence is satisfiable, which we can do by decidability of nonemptyiness of these automata, or appealing to decidability of satisfiability of Weak MSO over trees, which follow from Rabin's theorem: see, for example, Theorem 11.3 of (Thomas 1997). □

The argument for Theorem 8 is similar, except the confirmation of the BTVIP requires the analysis of CQ views as in Lemma 6.5 and Theorem 8.2 of (Benedikt et al. 2023).

C.2 Proof of Theorem 9: decidability of monotone determinacy for CQ views and queries, for linear TGDs

Recall the statement of Theorem 9:

Suppose Σ ranges over linear TGDs, Then monotonic determinacy of UCQ Q over CQ views $\mathbf{V}$ relative to Σ is decidable.

Proof. We make use of the certain answer rewriting approach outlined in the body of the paper. That is, we will use certain answer rewriting to eliminate steps moving backward in the process of Figure 3.

We can produce a UCQ R_1 which is a certain answer rewriting of Q with respect to Σ.

Consider a variation of the process of Figure 3, where the chase steps in line 4 produce facts in a primed signature, and the further steps after that are also in the primed signature. This change of signature does not impact the correctness of the procedure. Now consider the steps where we apply the views and then "chase backwards" with the view-to-definition rules. We can consolidate these into a single step that chases with rules of the form:

$$\phi_v(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \exists \boldsymbol{y}\ \phi'_v(,\boldsymbol{y})$$

where ϕ_q is the view definition of a view $v(\boldsymbol{x})$. These are source-to-target rules, which are always first-order rewritable. That is, we can get a UCQ R_2 that is certain answer rewriting of R_1 for these rules.

Now monotonic determinacy, the success of the whole process, is equivalent to the entailment

$$Q \models_\Sigma R_2$$

This is an entailment of UCQs with linear TGDs, decidable in PSPACE. □

C.3 More details on decidability and monotonic determinacy in the finite

In Section 4 in the body we mentioned that in the three decidable cases for monotone determinacy, we have decidability in the finite.

The *second decidable case* was about Fr-1 TGDs, MDL queries and CQ views. For UCQ queries and CQ views, finite controllability was proven in Theorem 10 in the paper. We can use the same ideas to handle the MDL case: In order to extend to MDL queries, we need to use the trick of "moving MDL rules into Fr-1 TGDs" that is applied in the proof of Theorem 3.

The *third decidable case* had to do with *CQ views and queries, linear TGDs*. We state the result here:

Proposition 18. *Suppose Σ consists of Linear TGDs. Then monotonic determinacy of a CQ query Q over CQ views $\mathbf{V}$ with respect to Σ is finitely controllable.*

We give the proof of the proposition. It also relies on finite controllability results for certain answers, but in this case we use an old result. Recall that a *k-universal model* for an instance $\mathcal{I}$ and set of dependencies Σ is a superinstance $\mathcal{I}_k$ of $\mathcal{I}$ that satisfies Σ with the property that a CQ of size at most k holds in $\mathcal{I}_k$ if and only if it is entailed by $\mathcal{I}$ and Σ.

For any TGDs, The chase is a k-universal model for any k, but in general it is infinite. However for linear TGDs it is known that we can do better:

Theorem 21. *(Rosati 2011) Linear TGDs have finite k-universal models for each instance $\mathcal{I}$ and each k.*

We can easily use this to prove Proposition 18.

Proof. Compute a UCQ Q^* that is a rewriting of Q' (the primed version of Q) with respect to Σ'. Such a Q^* exists by the BDDP. Let q be the size of Q, k be the maximal number of atoms in Q^*, and let w be the maximal number of atoms in a view.

Choose a $k \cdot w$-universal model $\mathcal{I}_1$ for the canonical database of Q and Σ. Let $\mathcal{I}_2$ be the result of applying $\Sigma_{\mathbf{V}}$ to $\mathcal{I}_1$, where $\Sigma_{\mathbf{V}}$ are the source-to-target TGDs corresponding to $\mathbf{V}$, going from the unprimed relations to the primed ones. Let $\mathcal{I}_3$ be a q-universal model for $\mathcal{I}_2$ and Σ.

We claim that if the pair $\mathcal{I}_1, \mathcal{I}_3$ is not a finite counterexample to monotonic determinacy of Q over $\mathbf{V}$, then Q is monotonically-determined over $\mathbf{V}$. This would imply monotonic determinacy is finitely controllable.

Clearly $\mathcal{I}_1, \mathcal{I}_3$ form a counterexample to monotonic determinacy as long as $\mathcal{I}_3$ does not satisfy Q', so suppose $\mathcal{I}_3$ satisfies Q'. We know by q-universality of $\mathcal{I}_3$ over Σ' and $\mathcal{I}_2$ that Q' must be entailed by Σ' over $\mathcal{I}_2$. Hence the rewriting Q^* must hold in $\mathcal{I}_2$. Thus Q^* must be entailed by $\Sigma_{\mathbf{V}}$ over $\mathcal{I}_1$, by universality of the usual chase. Letting Q'' be a rewriting of Q^* with respect to $\Sigma_{\mathbf{V}}$, we know Q'' holds in $\mathcal{I}_1$. The maximal size of disjuncts in Q'' is at most $k \cdot w$, and thus by $k \cdot w$-universality of $\mathcal{I}_1$, Q'' is entailed by Σ over the canonical database of Q.

We argue that Q' is entailed by $\Sigma, \Sigma_{\mathbf{V}}, \Sigma'$ over canonical database of Q, which would imply monotonic determinacy. This follows using the properties of rewritings and the chase. Since Q'' is entailed by Σ it will hold in $\mathcal{I}_1^\infty$, the chase of the canonical database under Σ. Since Q'' is a rewriting of Q^*, Q^* will hold in $\mathcal{I}_2^\infty$, the chase of $\mathcal{I}_1^\infty$ by $\Sigma_{\mathbf{V}}$. And since Q^* is a rewriting of Q', we infer Q' must hold in the chase of $\mathcal{I}_2^\infty$ by Σ'. Thus the "pipeline" described in Figure 3 returns true, and using Proposition 1 we conclude that Q is monotonically determined over $\mathbf{V}$ relative to the rules. □

Thie leaves the *first decidable case*. Recall that this case is *"everything is guarded"*: the views and queries are both guarded, and rules are FGTGDs. Here we do not claim finite controllability of monotonic determinacy. Instead we argue for decidability of monotone determinacy in the finite directly.

Theorem 22. *Let Q be a UCQ query, $\mathbf{V}$ a set of CQ views, and Σ a set of linear TGDs If Q is monotonically determined by $\mathbf{V}$ with respect to Σ over finite structures, then the same holds over all structures.*

We provide the proof of this theorem. The *Guarded Negation Fixpoint Logic* (GNFP) is a fragment of LFP with the restrictions:

- Only existential quantification
- Free variables in a negated formula are guarded. negation is of the form: $R(\boldsymbol{x}) \wedge \neg\phi(\boldsymbol{x})$, where R is an input relation, not a free second order variable that gets bound by a fixpoint.
- Fixpoints are guarded. To take a least fixpoint over predicate U of a formula $\phi(U, \boldsymbol{x})$, $\boldsymbol{x}$ must be guarded by an input relation.

If we disallow fixpoints, we get the *Guarded Negation Fragment* GNF.

Monotone determinacy of Q over $\mathbf{V}$ with respect to Σ can be rephrased as the satisfiability of: $Q \wedge \Sigma \wedge \Sigma_{\mathbf{V}} \wedge \Sigma' \wedge \neg Q'$ where Q' is a copy of the query on a signature where each predicate R is replaced by a primed copy R', and similarly for the rules Σ. $\Sigma_{\mathbf{V}}$ contains axioms:

$$\forall \boldsymbol{x}\ \phi_V(\boldsymbol{x}) \rightarrow \phi'_V(\boldsymbol{x})$$

where ϕ_V is the definition of view V, and ϕ'_V is a copy in a primed signature.

Q and $\neg Q'$ are in GNF, hence in GNFP. In the latter case, this is because there are no free free variables, recalling that Q is a Boolean CQ. Σ and Σ' are likewise in GNF. Q is clearly in GNFP, since the rules are assumed guarded. Similarly $\Sigma_{\mathbf{V}}$ can be expressed in GNFP, since the views are guarded by a base relation.

Now we can use a result from (Bárány, Cate, and Segoufin 2015) that entailment for GNFP is decidable in the finite.

C.4 Proof of Theorem 10

Recall the statement of 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. Fix Q, $\mathbf{V}$, and Σ. Let σ be the signature of Q and Σ. Let σ' denote primed the copy of σ. Below we shall use $\cdot'$ to denote copies of queries, rules, and instances over σ to their respective versions in the primed signature σ'.

We show that the general and finite cases of monotonic determinacy coincide. To this end assume that a there exists a counterexample to monotonic determinacy. We shall show that there exists a finite counterexample. Note that the other direction is trivial.

As stated above, we let Q' be the query Q copied over to to σ', the copy of the base signature, and similarly let Σ' denote the copy of the rules on a primed signature. Using our results on certain-answer rewriting, Theorem 3 , we can get an MDL certain answer rewriting R' of Q' with respect to Σ'. Note that R' works over the copied signature σ'. Note that since the views are CQs, we can treat view definitions and their "inverses" (view-to-definition rules) as TGDs. One can rewrite each rule body of R' against these rules, ignoring the intensional predicates. Let Θ denote the set of such TGDs, We let R denote the resulting Datalog program, with the notation indicating that it is in the unprimed signature σ. Note that in the process of rewriting R' we did not change the heads of R', meaning that R is also MDL.

We shall state the properties of R and R' that follow from their respective definitions. We have that:

- ♠) for any instance $\mathcal{I}$ over σ if $\mathcal{I} \models R$ then $\mathcal{I} \wedge \Theta \models R'$;
- $\diamondsuit$) for any instance $\mathcal{I}'$ over σ' if $\mathcal{I}' \models R'$ then $\mathcal{I}' \wedge \Sigma' \models Q$.

Consequently we have $\heartsuit$) $\mathcal{I} \models R$ implies $\mathcal{I} \wedge \Theta \wedge \Sigma \models Q$.

As there exists a counterexample to monotonic determinacy, from Proposition 1 we know that there exists one produced by the process of Fig. 3. Note that when we apply this process in the case of CQ query and views, it is deterministic. Let C_n be as in Fig. 3. Note, that by the definition of R and C_n we have $C_n \models Q \wedge \Sigma \wedge \neg R$, otherwise, from ($\heartsuit$) the process of Fig. 3 would return true. From Theorem 4, we obtain a finite model of Q and Σ that does not entail R. Let C_f denote it. Note that $\textsc{BackV}_{\mathbf{V}}(\mathbf{V}(C_f))$ is a singleton, again since the views $\mathbf{V}$ are CQs. Take the instance and copy it to the σ' signature, letting M'_f denote the result. Thus M'_f is finite and is isomorphic to $\textsc{Chase}_{\Theta}(C_f)$. Therefore, from (♠), we have that M'_f does not entail R'. From ($\diamondsuit$), we have that M'_f and Σ' do not entail Q. From a second application of Theorem 4 we obtain a finite model of M'_f and Σ' that does not entail Q. Let C'_f denote that model. Finally observe that $\mathbf{V}(C_f) \subseteq \mathbf{V}(C'_f)$, $C_f \models Q$, and $C'_f \not\models Q'$. Therefore, as both C_f and C'_f are finite $C_f \cup C'_f \cup \mathbf{V}(C_f)$ form a finite counterexample to monotonic determinacy. □

C.5 More details on Theorem 12: rewritability via the forward-backward method

Recall Theorem 12:

Let Σ be a set of FGTGDs, 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 of Q in terms of $\mathbf{V}$ w.r.t Σ.

Proof. For this proof we provide a sketch of the argument.

We proceed by first using Proposition 9 to get a tree automaton A_0 that represents approximations of Q. We proceed by first using Proposition 9 to get a tree automaton A_0 that represents approximations. We can also use definability in WMSO to argue that the set of instances that contain the approximations, satisfy the rules Σ, and have their view instances correct are recognized by a non-deterministic Büchi automata.

Example 3. *Consider a base schema with binary predicate R, S and unary predicates W and Z. Consider a Datalog query Q that asserts that there is an R path from a W node to a U node:*

$$\begin{aligned} \textsc{Goal} &:= W(x), I(x) \\ I(x) &:= Z(x) \\ I(x) &:= R(x,y), I(y) \end{aligned}$$

The canonical expansions are just words that begin with W and traverse R edges until reaching Z.

Consider Σ consisting of the rules:

$$\begin{aligned} Z(x) &\rightarrow \exists y\, S(x,y) \\ S(x,y) &\rightarrow \exists z\, S(y,z) \end{aligned}$$

Thus the chase with Σ will append to the final Z node an infinite chain of S edges. Note that the adjusted treewidth is 2.

Consider $\mathbf{V}$ that copy the S and R edges, omitting the unary predicates. For brevity we call the view predicates R and S.

The Büchi tree automaton that we want will accept exactly infinite chains consisting of an initial chain of R edges and then an infinite chain of S edges. It will have states q_R, q_S, with q_R initial and q_S accepting (i.e. the Büchi acceptance condition is $F = \{q_S\}$). The label alphabet will be sets of facts over two elements.

The transitions will be:

- t_1*: if we have a parent with label including* $R(p, q)$ *and the parent is in state* q_R*, then in the unique child we can transition to* q_R
- t_2*: if we have a parent with label including* $S(p, q)$ *and the parent is in state* q_R*, then in the unique child we can transition to* q_S
- t_3*: if we have a parent with label including* $S(p, q)$ *and the parent is in state* q_S*, then in the unique child we can transition to* q_S

The transition function of the automaton is transformed into Datalog rules similar to the backward mapping rules defined for finite tree automata in the algorithm of Theorem 1 mentioned in the body. Recall that this refers back to Section 7 of (Benedikt et al. 2023), with the algorithm given in the beginning of that section and the correctness proven in Proposition 7.1 (Benedikt et al. 2023). We now review the construction, which is quite intuitive, and then describe how we modify it. The algorithm translates each transition in the automaton into a Datalog rule. Ignoring some additional parameters needed for bookkeeping, in this translation each state q of the automaton over tree codes with k local names will translate into an intensional predicate $P_q(x_1 \dots x_k)$ on the instance that is coded. Each transition between parent state q and child states q_1, q_2 in the automaton will correspond to a rule with P_q in the head and P_{q_1}, P_{q_2} in the rule body.

The distinction from Theorem 1, which dealt with standard tree automaton, is that we have to deal with the acceptance conditions of the Büchi automaton. So now we create not a Datalog query, but a POSLFP sentence. We change the finitely many Datalog rule bodies associated to any intensional predicate I_q into a disjunction of CQs. So now we have a vector consisting of intensional predicates and associated UCQs.

Example 4. *Let us continue the examination of Example 3. The backward mapping for the example will have binary intensional predicates for each state,* $I_{q_R}(p, c)$*,* $I_{q_S}(p, c)$*, and extensional predicates for the binary predicates* R *and* S*, recalling that these represent view predicates.*

If we were doing the backward mapping for finite tree automata, we would produce a Datalog program with rules

$$I_{q_R}(p, c) := R(p, c) \wedge I_{q_R}(c, c')$$
$$I_{q_R}(p, c) := S(p, c) \wedge I_{q_S}(c, c')$$

corresponding to transitions t_1, t_2 *above, and similarly for* I_{q_S} *and* t_3*.*

In our construction, we will proceed as above for the predicates corresponding to non-accepting states – $I_{q_R}(x, y)$ *above. But we will proceed differently for the predicates corresponding to accepting states, like* $I_{q_S}(x, y)$ *above.*

We illustrate the construction of the POSLFP formula for the case when A is *shallow*, in the sense that when a parent is in an accepting state, it can only transition to children that are also in accepting states. The example above has this property: the intuition for this example is that the accepting states correspond to tree code vertices generated by view images of the chase, a suffix, while non-accepting states will correspond to treecode vertices that come from unfolding the approximation, a prefix.

Instead of creating Datalog rules as in the usual backward mapping, we will create corresponding fixpoint formulas, where the non-accepting states are treated using least fixpoints, as in traditional Datalog, while the accepting states are treating using greatest fixpoints. We can consider these as vectorized fixpoints, binding a collection of predicates at once. It is known by the "Bekić principle" (Bekić 1984) that such vectorized fixpoints can be replaced by sequences of usual fixpoints. The predicates corresponding to accepting states (informally, those corresponding to nodes generated in the chase, or witnesses to views), are put into an inner vector of ν operators. Predicates corresponding to non-accepting states – informally, those corresponding to the CQ approximation – are bound with a μ operator. This ordering here – accepting states within the scope of non-accepting state – corresponds in our case to the fact that we first choose the approximation, then chase it and take the view image.

We explain the construction by continuing the example, leaving some details to the full version.

Example 5. *Let us continue the examination of Example 3. The backward mapping for the example will have binary intensional predicates for each state,* $I_{q_R}(p, c)$*,* $I_{q_S}(p, c)$*, and extensional predicates for the binary predicates* R *and* S*, recalling that these represent view predicates.*

We produce the system of fixpoints:

$$\mu_{I_{q_R}} : I_{q_R}(p, c) := ((\exists c' \; R(p, c) \wedge I_{q_R}(c, c')) \vee \exists c' \; (R(p, c) \wedge I_{q_S}(c, c')))$$
$$\nu_{I_{q_S}} : I_{q_S}(p, c) := (S(p, c) \wedge \exists c' \; I_{q_S}(c, c'))$$

This asserts that we do an outer least fixpoint for I_{q_R} *and in each iteration do a greatest fixpoint on* $I_{q,S}$*. And then we add the quantification* $\exists pc \; I_{q_R}(p, c)$*.*

That is, our POSLFP *formula states that:*

- *there is an* I_{q_R} *pair*
- I_{q_R} *pairs are labelled correctly and have* R *paths to an* I_{q_S} *pair.*
- *every* I_{q_s} *pair is* S*-labelled correctly and links to another* I_{q_s} *pair*

Thus the formula asserts exactly the structure we want within the view instance.

Now we explain how this example generalizes. Recall that for FGTGDs, the instance can be given a tree structure, where each tree vertex is associated with a collection of facts, as in a tree decomposition. When a FGTGD τ is fired, the trigger image of the body is contained in some vertex v of the tree. For a non-full TGD, we create a new child of v which includes the facts in the head along with any facts from v that are guarded by the facts in the head. For a full TGD we create new facts in the same node, and we may propagate a fact back from a child to its parent, if the fact is guarded in the parent. We say that an FGTGD τ is *speedily witnessed* in a tree code if whenever the body of the TGD matches via a homomorphism h into a tree vertex v, the head facts are found in either in v or in the child created when the the chase step for h is fired. We say a tree code t' extends another tree codes t if there is an injective mapping f from the nodes of t to the nodes of t' such that the labels of node v are a subset of the labels of the nodes of $f(v)$.

Claim 1. *For* FGTGD*s* Σ *with* k *the maximum number of elements in a frontier, and any finite* k *tree code* t *with width the maximum number of elements in a rule , there is a* k *tree code* t' *that extends* t *where each rule in* Σ *is speedily witnessed.*

Proof. This can be proven using standard results about the treelike chase for FGTGDs, see for example (Benedikt et al. 2022): when we need a new speedy witness we create a child. In order to ensure that witness triggers for Σ always have frontier within a single node, we always propagate a new fact from child to parent if it is guarded in the parent. But since all such facts are guarded in the parent, this will not violate the bound k. □

The *hybrid envelope trees* for a Datalog query Q, views $\mathbf{V}$, and FGTGDs Σ are the tree codes t' for the combined base and view signature, such that:

- t' extends an approximation tree corresponding to the canonical database of some approximation Q_n
- every TGD is speedily witnessed in t'
- for each $\boldsymbol{d}$ coded in a single node of t', for each view definition $\phi(\boldsymbol{x})$, the corresponding view fact $V_\phi(\boldsymbol{d})$ is present in the tree node

Using the proof of Claim 1 above, we can easily show:

Claim 2. *For FGDL views* $\mathbf{V}$ *and a Datalog query* Q*, and any number* n*, there is a hybrid envelope tree* t *whose decoding is a universal model for* $\textsc{CanonDB}(Q_n) \wedge \Sigma$.

Proof. We perform the treelike chase to obtain a tree satisfying the first two items: that is, we repeatedly extend as in Claim 1. And then we simply evaluate each view definition to add the corresponding facts. Since the views are FGDL we do not break the tree decompositon when we do this. □

A Büchi automaton is *shallow* if for every accept state q and every transition whose parent state is q, the associate child states are also accepting. That is, in an accepting run, accepting states form a suffix of the tree.

Lemma 11. *For Datalog* Q*, FGDL* $\mathbf{V}$*, and* FGTGD*s* Σ*, there is a shallow automaton that accepts exactly the hybrid envelop trees for* Q*,* $\mathbf{V}$*,* Σ.

Proof. The first item in the definition can be enforced with a shallow Büchi automaton easily: we have non-accept states for the intermediate intentionsal predicates and then accept states when we reach extensional facts. The second item can be enforced with an automaton having a single accept state that is initial and a sink non-accepting state The third item can be enforced with a shallow automaton that guesses the intermediate intensional relations holding at each node.

The automaton we want then is the product, and the product of shallow automaton is shallow. □

We note that the projection of a shallow automaton is shallow. Let A_V be the automaton obtained from Lemma 11 by projecting away everything but the view facts.

We let $\phi_{\mathbf{V},\Sigma,Q}$ be the POSLFP formula obtained by performing the backward mapping of the shallow automaton A_V above. Our main claim is the following:

Theorem 23. $\phi_{\mathbf{V},\Sigma,Q}$ *is a rewriting of* Q *over* $\mathbf{V}$ *with respect to* Σ.

Proof. In one direction suppose Q holds in $\mathcal{I}$ and let $\mathcal{J}$ be the view image of $\mathcal{I}$. There is some approximaton Q_n with a homomorphism h sending Q_n into $\mathcal{I}$. We apply Claim 2 to get a hybrid envelope tree t embedding Q_n that is a universal model for $\textsc{CanonDB}(Q_n) \wedge \Sigma$. We let t' be the corresponding projected tree onto view facts. Then h extends to a homomorphism h_t from the decoding of t into $\mathcal{I}$. By definition of the automaton A_V, t' is accepted by A_V. Since A_V is shallow, there is a run r of A_V on t' in which the domain of accepting states form a suffix. Let p' be the corresponding prefix. We first give a strategy for unfolding the outer least fixpoint of $\phi_{\mathbf{V},\Sigma,Q}$: we unfold until we reach the boundary of $h_t(p')$. At that point, we can unfold the inner fixpoints infinitely often. This witnesses that $\mathcal{J}$ satisfies $\phi_{\mathbf{V},\Sigma,Q}$.

In the other direction, suppose $\phi_{\mathbf{V},\Sigma,Q}$ holds in $\mathcal{J}$.

There is an unfolding of the outer fixpoint that witnesses this, and a finite tree p' where nodes are labelled with the inner fixpoint, such that there is a homomorphism from $\mathfrak{D}(p')$ into $\mathcal{J}$. Repeatedly expanding the inner fixpoint at each node of p', we will obtain a tree with p' as a prefix, thus yielding an infinite tree t' accepted by A_V (filled out with self loops or dummy nodes), along with a homomorphism h' of $\mathfrak{D}(t')$ into $\mathcal{J}$. Since A_V is a projection of the automaton obtained from Lemma 11, there is an expansion of t' to a tree t_0 with codes for base facts that embeds the canonical tree for Q_n into $\mathcal{I}$, where the decoding of t_0 satisfies Σ, and where the view facts are contained in those given by the view definitions for $\mathfrak{D}(t')$. Let $\mathcal{I}_0$ be the decoding of t_0.

$\mathcal{I}_0$ satisfies Q since it contains a homomorphic image of Q_n. Using monotonic determinacy, we can see that Q holds in $\mathcal{I}$ as well. □

□

C.6 Proof of Theorem 13: Non-rewritability in Datalog

Recall that if we have frontier-guarded TGDs, an FGDL query Q, and Datalog views $\mathbf{V}$, with Q monotonically determined over $\mathbf{V}$ with respect to Σ, then there is a rewriting of Q over $\mathbf{V}$ in Datalog. This is Theorem 14. On the other hand, if Q is in Datalog, our result only give a rewriting in POSLFP. As mentioned in the body, n the case of frontier-guarded TGDs, or even linear TGDs, we do not know if POSLFP is needed.

We now show that there are TGDs with the BTVIP, where we cannot obtain rewritability in Datalog, we really need POSLFP. This stated in Theorem 13 in the body:

There are rules Σ, a Datalog query Q such that the BTVIP holds, and FGDL views with Q monotonically determined by $\mathbf{V}$ w.r.t. Σ, where there is no Datalog rewriting.

Here we emphasize that, as in our default for the paper, we deal with all instances: so the example will be monotonically determined over all instances, and *there is no Datalog rewriting over all instances*. For the example we give, there will be a Datalog rewriting over finite instances. This is a case where the argument is specific to the setting of unrestricted instances.

Recall, the *compactness of entailment* property discussed in the context of our positive results on rewriting. It deals with the situation where we have an infinite instance of the view schema, and it is Q-entailing with respect to $\Sigma, \mathbf{V}$ – that is, it implies, using the view definitions and Σ, that Q holds. Then there is a finite subinstance if the view image that is Q-entailing with respect to $\Sigma, \mathbf{V}$ well. It is easy to see that this is a necessary condition to have a Datalog rewriting. Theorem 5 proved compactness of entailment in the setting where views and queries are frontier-guarded and the rules are tame. We give an example, where the rules still have the BTVIP, although they are not FGTGDs, where this property is violated. And using this failure, we will easily conclude that Datalog-rewritability fails.

Proposition 19. *There is a set of TGDs Σ, a Boolean CQ Q a set of* FGDL *views $\mathbf{V}$ with the BTVIP such that Q is monotonically determined by $\mathbf{V}$ relative to Σ but Q has no Datalog rewriting over the views in $\mathbf{V}$.*

In fact, the argument will show that *there is no rewriting as an infinite disjunction of CQs*.

Proof. Let the schema $\mathbf{S}$ contain the nullary symbol GOAL, the unary relation symbol U and the binary relation symbols A, L and R. Define the following set of rules for $\mathbf{S}$

$$\begin{aligned}\Sigma = \{&\textsc{Goal} \wedge A(x,x) \rightarrow \exists y\, R(x,y),\\ &\textsc{Goal} \wedge R(x,y) \rightarrow \exists z\, R(y,z),\\ &R(r,r') \wedge L(l,l') \wedge A(r,l) \rightarrow A(r',l')\\ &A(r,l) \wedge U(l) \wedge L(x,y) \rightarrow \textsc{Goal} \wedge A(l,l)\}.\end{aligned}$$

Let the view schema contain binary relations V_A and v_R along with a unary predicate $\textsc{Reach}_U$. The views $\mathbf{V}$ are defined such that V_A and V_R have copy views, meaning the trivial view definitions $V_A(x,y) \rightarrow A(x,y)$ and $V_R(x,y) \rightarrow R(x,y)$. For the unary predicate $\textsc{Reach}_U$ we have the following Datalog view, which returns the nodes that can reach a U node via an L-path:

$$\begin{aligned}&\textsc{Reach}_U(x) := U(x)\\ &\textsc{Reach}_U(x) := L(x,y) \wedge \textsc{Reach}_U(y)\end{aligned}$$

Define the Boolean CQ $Q = \exists x\, \textsc{Goal} \wedge A(x,x) \wedge U(x)$.

Claim 3. *Q is monotonically determined by the views in $\mathbf{V}$ relative to rules Σ.*

Proof. Consider the process for checking monotonic determinacy with rules. We start with the canonical database of Q, which has the goal predicate, along with a single element x_0 satisfying U, with a self-loop $A(x_0,x_0)$. Then we consider the result of chasing this instance with the rules. Chasing with the first two rules we create an infinite R chain rooted at x_0, say $x_0, x_1 \ldots$. The two last rules do not fire, since there are no L atoms present.

Now applying the views, the copy of R reveals the R path within the view image. The copy of A contains only the self-loop on x_0. Finally, we know that $\text{REACH}_U(x_0)$ holds. As x_0 can reach a U node via a path of L edges. That is, $\text{REACH}_U(x_0)$ is generated from a degenerate, empty R-path.

Note that even though the rules are not frontier-guarded the BTVIP holds. Indeed, on the approximations of Q, the non-guarded rules never fire.

Thus in the "reverse views" step, we have instances $\mathcal{I}_n$ each containing the A self-loop on x_0, the R-facts, and a path $x_0 = v_0 \dots v_n$ of size n connected by L edges, leading from x_0 to a U node v_n.

Consider now chasing the instance $\mathcal{I}_n$ with the rules. The first two rules do not fire initially, since GOAL does not hold. But now the second-to-last rule does fire, propagating A-facts $A(x_i, v_i)$. When we get $A(x_n, v_n)$, we deduce whole Q using the final rule. Thus Q holds in every "test", and the algorithm returns true. Therefore we have monotonic determinacy. □

We now claim that there is no Datalog rewriting of Q over V that works over finite and infinite Σ-instances.

Assume for a contradiction that a Datalog rewriting Π of Q exists. We first argue that Π must be true in the following infinite instance of the view schema

$$\begin{aligned}\mathcal{J}_\infty = \{&\text{REACH}_U(x_0),\\ &V_R(x_0, r_1), V_R(r_1, r_2), \dots, V_R(r_n, r_{n+1}), \dots,\\ &V_A(x_0, x_0), V_A(r_1, l_1), V_A(r_2, l_2), \dots, V_A(r_n, l_n), \dots\}.\end{aligned}$$

The reason is that Q is true in the following instance that satisfies Σ:

$$\begin{aligned}\mathcal{I}_\infty = \{&\text{GOAL}, U(x_0),\\ &R(x_0, r_1), R(r_1, r_2), \dots, R(r_n, r_{n+1}), \dots,\\ &A(x_0, x_0), A(r_1, l_1), A(r_2, l_2), \dots, A(r_n, l_n), \dots\}.\end{aligned}$$

and one easily checks that the view image of these two instances are the same. Because Π is a rewriting of Q it follows that Π is true in $\mathcal{J}_\infty$.

Because Π is in Datalog, if it is true in some instance then it is already true in a finite subinstance of that instance. Thus, there exists an n such that Π is true in the instance:

$$\begin{aligned}\mathcal{J}_n = \{&\text{REACH}_U(x_0),\\ &V_R(x_0, r_1), V_R(r_1, r_2), \dots, V_R(r_{n-1}, r_n),\\ &V_A(x_0, x_0), V_A(r_1, l_1), V_A(r_2, l_2), \dots, V_A(r_n, l_n)\}.\end{aligned}$$

But then consider the following instance:

$$\begin{aligned}\mathcal{I}_n = \{&U(l_{n+1}),\\ &R(x_0, r_1), R(r_1, r_2), \dots, R(r_{n-1}, r_n),\\ &A(x_0, x_0), A(r_1, l_1), A(r_2, l_2), \dots, A(r_n, l_n),\\ &L(x_0, l_1), L(l_1, l_2), \dots, L(l_{n-1}, l_n), L(l_n, l_{n+1})\}.\end{aligned}$$

One can easily check that $\mathcal{J}_n \subseteq \mathbf{V}(\mathcal{I}_n)$. To see that $\mathcal{I}_n$ satisfies Σ, observe that the absence of the GOAL fact means that the first two rules hold vacuously. The third rule can easily be seen to hold. And since the unique node satisfying U, l_{n+1}, is not the target of an A-edge, the last rule holds vacuously.

Note that Q is false in $\mathcal{I}_n$, because GOAL is not true there. But then Π can not be a rewriting of Q, because $\mathcal{J}_n \subseteq \mathbf{V}(\mathcal{I}_n)$, Π is true in $\mathcal{J}_n$, but Q is false in $\mathcal{I}_n$. □

C.7 More detail on Proof of Theorem 15: better rewriting using Compactness of Entailment and Automata for Entailing Witnesses

Recall the forward-backward method for generating rewritings for monotonically determined queries over views in the presence of rules. It is available when $(Q, \Sigma, \mathbf{V})$ has the BTVIP. We form a tree automaton for the chase of the view images of unfoldings, and then change them into a relational query. If we have the FBTVIP, we can make due with a finite tree automata, and generate a Datalog query. If we have only the BTVIP – as is the case when the chase does not terminate – then we need a tree automaton over infinite trees to capture the view images, and thus get only a POSLFP rewriting, as in Theorem 12. But if we can argue for compactness of entailment, we can use a finite tree automata to capture a sufficiently large portion of the view image, and use the backward mapping to get a Datalog rewriting again. This is what happens in Theorem 14.

We can use compactness of entailment in a more straightforward way to prove the Datalog rewriting result of Theorem 15. We restate the result here:

Suppose Q a Boolean UCQ, $\mathbf{V}$ a set of Datalog views, and Q is monotonically determined by $\mathbf{V}$ w.r.t.Σ. Then there is a UCQ rewriting of Q over $\mathbf{V}$ relative to Σ. When Q is a CQ, the rewriting can be taken to be a CQ as well.

Note that when we apply the Datalog views to the chase, we may not get bounded treewidth.

Proof. For simplicity, we give the argument only when Q is a CQ. Let V_0 be the view image of $\text{CHASE}_\Sigma(\text{CANONDB}(Q))$: that is, V_0 is the instance formed in the first few steps of the process in Figure 3. Since Q is monotonically determined over $\mathbf{V}$ relative to Σ, V_0 must entail Q using the view-reversing process. Thus by the compactness result, Theorem 5, there is a finite subinstance V_0^- of V_0 that entails this. Changing V_0^- into a CQ gives the desired rewriting. □

D Additional material related to the surprising undecidability results, Section 5

D.1 Proof of Theorem 16: the undecidability result for CQs and combinations of Linear and Frontier-1:

Recall 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.

Proof. We first focus on the first part of the theorem. The second will follow by replacing the use of Frontier-one TGDs by MDL rules.

We prove undecidability of monotonic determinacy by reduction from *state reachability problem for deterministic one-dimensional cellular automata.* A cellular automaton is a pair $\mathbb{A} = (\mathcal{Q}, \Delta)$ where $\mathcal{Q} = \{T_0, T_1, \ldots, T_n\}$ is a set of cell states with T_0 a blank state, and Δ is a set of transitions of the form $T_iT_jT_k \rightarrow T_l$ and $T_jT_k \rightarrow T_l$. All transitions should be deterministic. This means that for each tuple of natural numbers (i, j, k) there is at most one l with $T_iT_jT_k \rightarrow T_l \in \Delta$ and for each pair of numbers (j, k) there is at most one l with $T_jT_k \rightarrow T_l \in \Delta$. A *tape state* is an infinite sequence $\sigma_0\sigma_1\sigma_2\ldots$ of elements of $\mathcal{Q}$ such that there is $i \in \mathbb{N}$ such for all $j \geq i$ $\sigma_j = T_0$. We say that a tape state $\varrho = \varrho_0\varrho_1\varrho_2\ldots$ $\mathbb{A}$-*follows* a tape state $\sigma = \sigma_0\sigma_1\sigma_2\ldots$ if Δ contains transitions $\sigma_0\sigma_1 \rightarrow \varrho_0$ and $\sigma_{i-1}\sigma_i\sigma_{i+1} \rightarrow \varrho_i$ for $i \geq 1$. We say that T_n is reachable in $\mathbb{A}$ if there is a tape state ϱ containing T_n such that the pair $(T_0T_0T_0T_0\ldots, \rho)$ is in the transitive closure of the $\mathbb{A}$-follows relation. Since a deterministic 1-dimensional cellular automata can model an arbitrary deterministic Turing machine, the problem of determining given $\mathbb{A}$ and T_n whether T_n is reachable in $\mathbb{A}$, is undecidable.

Given a cellular automaton $\mathbb{A}$, we define a Frontier-Guarded Datalog query Q and a set of CQ views $\mathbf{V}$ such that Q is monotonically determined by $\mathbf{V}$ iff T_n is reachable in $\mathbb{A}$.

Our schema will use a relational representation of the two-dimensional grid shown in Figure 7. We have a vertical axis y which uses YSUCC for the successor relation and a horizontal axis x which uses XSUCC for the successor relation. Then we have the "grid points" in the quadrant between the axes which are linked via YPROJ and XPROJ relations to their projections on the axes. Unary predicates XZERO and YZERO mark the origins of the axes. We will also have ternary relations G and G′.

The following CQs will be useful in the sequel:

$$\text{HA}(z_1, z_2, y, x_1, x_2) = \text{YPROJ}(y, z_1) \wedge \text{YPROJ}(y, z_2) \wedge \text{XPROJ}(x_1, z_1) \wedge \text{XPROJ}(x_2, z_2) \wedge \text{XSUCC}(x_1, x_2)$$

This says that z_1 and z_2 have the same y-projection, while the x-projection of z_2 is next to the x-projection of z_1. The CQ VA is defined similarly.

It should be clear that $\text{RIGHTOF}(z_1, z_2) = \exists y\ x_1\ x_2 \text{HA}(z_1, z_2, y, x_1, x_2)$ holds for grid points z_1 and z_2 iff z_2 is the right neighbour of z_1. Similarly we can define $\text{VA}(z_1, z_2, y_1, y_2, x)$ checking vertical adjacency and $\text{DOWNTO}(z_1, z_2)$ by existentially quantifying VA indicates that z_2 is the downward neighbor of z_1. Note that we can define the axes and the origin:

- $\text{BOTTOMEDGE}(z) = \exists y\ \text{YPROJ}(y, z) \wedge \text{YZERO}(y)$
- $\text{AXISY}(z) = \exists x\ \text{XPROJ}(x, z) \wedge \text{XZERO}(x)$
- $\text{ATORIGIN}(z) = \text{BOTTOMEDGE}(z) \wedge \text{AXISY}(z)$

Consider the CQ Query:

$$Q = \exists z_0\ x_0\ x_1\ y_0\ y_1\ \text{G}(z_0, x_0, x_1) \wedge \text{XZERO}(x_0) \wedge \text{G}'(z_0, y_0, y_1) \wedge \text{YZERO}(y_0)$$

Along with the CQ views:

$$S(x, y) = \text{XPROJ}(x, z) \wedge \text{YPROJ}(y, z)$$

and the views $V_{\text{XZERO}}, V_{\text{YZERO}}, V_{\text{XSUCC}}, V_{\text{YSUCC}}$.

Our rules Σ consist of several groups:

(I) Linear TGDs that "build a grid":
- $\text{G}(z_0, x, x') \rightarrow \exists x''\ \text{G}(z_0, x', x'')$
- $\text{G}(z_0, x, x') \rightarrow \text{XPROJ}(x, z_0) \wedge \text{XSUCC}(x, x')$
- $\text{G}'(z_0, y, y') \rightarrow \exists y''\ \text{G}'(z_0, y', y'')$
- $\text{G}'(z_0, y, y') \rightarrow \text{YPROJ}(y, z_0) \wedge \text{YSUCC}(y, y')$

When chasing Q with these rules we will get an unbounded XSUCC chain of elements, with "grid point" z_0 projecting to all of them, and also a YSUCC chain of elements with z_0 projecting to all of them.

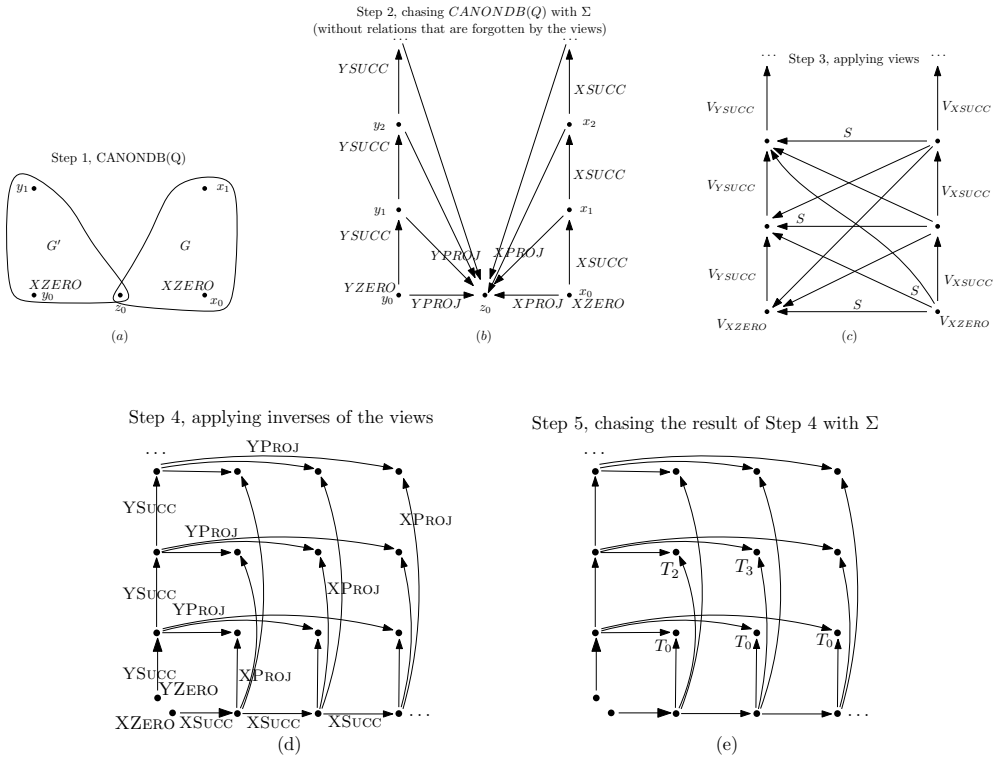

Figure 7: Applying the monotonic determinacy process of Figure 3 to Q, Σ and views from Theorem 16.

(II) Full frontier-one TGDs that simulate the run of the automaton:
- $\textsc{AtOrigin}(x) \to A(x)$
- $A(x) \to T_0(x)$
- $\textsc{AxisY}(y) \wedge T_j(y) \wedge \textsc{RightOf}(y, z) \wedge T_k(z) \wedge \textsc{DownTo}(y', y) \to T_l(y')$, for every transition $T_jT_k \to T_l \in \Delta$,
- $T_i(x) \wedge \textsc{RightOf}(x, y) \wedge T_j(y) \wedge \textsc{RightOf}(y, z) \wedge T_k(z) \wedge \textsc{DownTo}(y', y) \to T_l(y')$, for every transition $T_iT_jT_k \to T_l \in \Delta$,

(III) Linear TGDs that mark acceptance:

$$T_n(v) \to \exists x_0\ x_1\ y_0\ y_1\ z_0\ \mathbf{G}(z_0, x_0, x_1) \wedge \textsc{XZero}(x_0) \wedge \mathbf{G}'(z_0, y_0, y_1) \wedge \textsc{YZero}(y_0)$$

for each final state T_n

Claim 4. *T_n is reachable in $\mathbb{A}$ iff Q is monotonically determined over $\mathbf{V}$ with respect to Σ.*

Proof. The idea of the proof is shown graphically in Figure 7. It shows how the monotonic determinacy checking procedure from Figure 3 works on Q, Σ and views constructed in the proof of Theorem 16. In particular, (a) shows the canonical database of Q, (b) shows the result of chasing it with Σ omitting the symbols that are "forgotten" by the views, (c) shows the result of applying the views to (b), (d) shows the result of applying the "view inverting rules" (choosing a body for each view fact) to the result of (c), yielding a grid-like structure. Then (e) shows how chasing (d) with Σ simulates the work of the cellular automaton $\mathbb{A}$ with transitions $T_0T_0 \to T_2$ and $T_0T_0T_0 \to T_3$. It should be clear that due to Part 3 of Σ, Q holds in the ultimate result of (e) iff T_n is reachable in $\mathbb{A}$. □

The first part of Theorem 16 follows immediately from the claim and the undecidability of the state reachability problem for deterministic one-dimensional cellular automata.

For the second part of Theorem 16, we set Σ' to consist only of TGDs of group (I). TGDs of groups (II) and (III) we convert into a MDL query Q_{MDL} by interpreting TGDs of group (II) as rules of a Datalog program and adding the rule $\textsc{Goal} := T_n(x)$. We set $Q' = Q \vee Q_{MDL}$ and argue that

Claim 5. *T_n is reachable in $\mathbb{A}$ iff Q' is monotonically determined over $\mathbf{V}$ with respect to Σ'.*

To see this, note that every CQ-approximation of Q' is either CQ-approximation of Q or CQ-approximation of Q_{MDL} and that CQ-approximations of Q_{MDL} do not change after chasing by Σ' and are preserved after applying views followed by "view-inverting rules". Thus the monotonic determinacy checking procedure from Figure 3 always succeeds on CQ-approximation of Q_{MDL}. On the other hand, arguing similar to Claim 1, we can show that this procedure, when executed on Q, simulates $\mathbb{A}$, and succeeds if and only if T_n is reachable in $\mathbb{A}$. □

D.2 Proof of Theorem 17: Undecidability with UCQ views with only UIDs

Recall the statement of Theorem 17

The problem of monotonic determinacy is undecidable when Σ ranges over Unary Inclusion Dependencies (which are Frontier-1 and linear), Q over Boolean UCQs, and $\mathbf{V}$ over UCQ views.

We start with a warm-up, in which we use a richer class of rules.

Proposition 20. *The problem of monotonic determinacy is undecidable when Σ ranges over Frontier-1 TGDs, Q over Boolean atomic CQs, and $\mathbf{V}$ over UCQ views.*

Note that this is the same as our goal theorem, but we allow frontier-1 TGDs, rather than only linear frontier-1 TGDs.

Proof. We consider a tiling problem consisting of a set of tiles and horizontal and vertical constraints. The goal is to find a tiling of the infinite quarter-plane satisfying the constraints.

We take $Q = \exists x\ \textsc{Init}(x)$. We set Σ to be the union of $\Sigma_{\textsc{start}}$ and $\Sigma_{\textsc{verify}}$ where $\Sigma_{\textsc{start}}$ is

$$\begin{aligned} \textsc{Init}(x) &\rightarrow A_1(x) \wedge A_2(x) \\ A_1(x) &\rightarrow \exists y\ \textsc{XSucc}(x,y) \wedge A_1(y) \\ A_2(x) &\rightarrow \exists y\ \textsc{YSucc}(x,y) \wedge A_2(y) \end{aligned}$$

and $\Sigma_{\textsc{verify}}$ consists of the following TGDs:

(R1) $A_1(x) \rightarrow \exists y\ \textsc{Init}(y)$ and $A_2(x) \rightarrow \exists y\ \textsc{Init}(y)$

(R2) $\mathrm{HA}(z,z') \wedge T_i(z) \wedge T_j(z') \rightarrow \exists x\ \textsc{Init}(x)$

where $\mathrm{HA}(z,z')$ is $\exists x x' y\ \textsc{XProj}(z,x) \wedge \textsc{YProj}(z,y) \wedge \textsc{XProj}(z',x') \wedge \textsc{YProj}(z',y) \wedge \textsc{XSucc}(x,x')$, and i,j range over pairs violating the horizontal constraint of the tiling;

(R3) and similar rules for the vertically incompatible tiles.

The set of views $\mathbf{V}$ consists of only the *grid-generating view*, which is a UCQ:

$$\begin{aligned} S(x,y) &:= A_1(x), A_2(y) \\ S(x,y) &:= \textsc{XProj}(x,z), T_i(z), \textsc{YProj}(y,z) \text{ for all } T_i \text{ in } Tiles; \end{aligned}$$

We claim that a tiling exists exactly when Q is not monotonically determined over $\mathbf{V}$ with respect to Σ.

In one direction, suppose a tiling exists, and assume that its underlying domain is the set of pairs of positive integers. We let $\mathcal{I}$ be obtained from chasing $\textsc{Init}(x_0)$ with Σ, and let $\mathcal{J}$ be the view image of $\mathcal{I}$. We can identify the domain of $\mathcal{J}$ with pairs of integers, and let $\mathcal{I}'$ be obtained by taking the tiling and changing its domain to be the domain of $\mathcal{J}$. We see that $\mathcal{I}, \mathcal{I}'$ represent a counterexample to monotonic determinacy.

In the other direction, suppose monotonic determinacy fails. Then some iteration of the loop in Figure 3 will fail, where an iteration consists of:

(stage 1) taking the canonical database of Q

(stage 2) chasing with Σ its canonical database

(stage 3) applying the views

(stage 4) choosing disjuncts in chasing the inverse of the view rules

(stage 5) chasing again with Σ

The instance formed in the failing iteration $\mathcal{I}'$ must be based on $\{\textsc{Init}(x_0)\}$. Note that after choosing disjuncts but prior to the final chase step, $\mathcal{I}'$ could not have any A_1 or A_2 atoms, because otherwise $\mathcal{I}'$ after the final stage would satisfy Q because of (R1). Therefore $\mathcal{I}'$, prior to the last step, must be a proper grid with tiles given as T_i-atoms. As we know that $\mathcal{I}'$ does not satisfy Q, it follows that it yields a proper tiling of the quarter-plane.

□

With the warm up over, we are now ready for the proof the theorem:

Proof. We consider a tiling problem consisting of a set of tiles, an initial tile, vertical constraints and horizontal constraints. The goal is to find an infinite tiling satisfying the constraints.

We set Σ to be as follows:

$$\begin{array}{rcl} \textsc{Init}(x) & \rightarrow & A_1(x) \wedge A_2(x) \\ A_1(x) & \rightarrow & \exists y\ \textsc{XSucc}(x,y) \wedge A_1(y) \\ A_2(x) & \rightarrow & \exists y\ \textsc{YSucc}(x,y) \wedge A_2(y) \\ A_1(x) & \rightarrow & \exists y\ \textsc{Init}(y) \wedge \textsc{Origin}(y) \\ A_2(x) & \rightarrow & \exists y\ \textsc{Init}(y) \wedge \textsc{Origin}(y) \end{array}$$

These can clearly be converted to UIDs.

Q_{VERIFY} is the disjunction of the following CQs:

(CQ1) $\text{HA}(z,z') \wedge T_i(z) \wedge T_j(z')$ where $\text{HA}(z,z')$ is $\exists x x' y\ \textsc{XProj}(z,x) \wedge \textsc{YProj}(z,y) \wedge \textsc{XProj}(z',x') \wedge \textsc{YProj}(z',y) \wedge \textsc{XSucc}(x,x')$, and i,j range over pairs violating the horizontal constraint of the tiling;

(CQ2) similar CQs for the vertically incompatible tiles;

(CQ3) $\textsc{Origin}(x) \wedge T_i(x)$ for any i not equal to the (index of the) initial tile;

The set of views $\mathbf{V}_{TP}$ consists of

– the *grid-generating view*

$$\begin{array}{rcl} S(x,y) & := & A_1(x), A_2(y) \\ S(x,y) & := & \textsc{XProj}(x,z), T_i(z), \textsc{YProj}(y,z) \text{ for all } T_i \text{ in } Tiles; \end{array}$$

– the *atomic views* V_{YSUCC}, V_{XSUCC}, V_{ORIGIN}, V_{T_i} for EDBs $\textsc{YSucc}, \textsc{XSucc}, \textsc{Origin}$ and each T_i in $Tiles$;

– the following *special* views

$$\begin{array}{rcl} (SP3) \quad V_{\text{HA}}(z_1,z_2,y,x_1,x_2) & := & \text{HA}(z_1,z_2) \\ (SP4) \quad V_{\text{VA}}(z_1,z_2,y_1,y_2,x) & := & \text{VA}(z_1,z_2) \end{array}$$

Now we set $Q_{\text{START}} = \textsc{Init}(x) \wedge \textsc{Origin}(x)$ and $Q = Q_{\text{START}} \vee Q_{\text{VERIFY}}$.

We claim that a tiling exists exactly when Q is not monotonically determined over $\mathbf{V}$ with respect to Σ.

In one direction, suppose a tiling exists, and assume that its underlying domain is the set of pairs of positive integers. We let $\mathcal{I}$ be obtained from chasing $\textsc{Init}(x_0)$ with Σ, and let V be the view image of $\mathcal{I}$. We can identify the domain of V with pairs of integers, and let $\mathcal{I}'$ be obtained by taking the tiling and changing its domain to be the domain of V. We see that $\mathcal{I}, \mathcal{I}'$ represent a counterexample to monotonic determinacy.

In the other direction, suppose monotonic determinacy fails. Then some iteration of the loop in Figure 3 will fail, where an iteration is based on:

(stage 1) a choice of one of the disjuncts of Q

(stage 2) chasing with Σ its canonical database

(stage 3) applying the views

(stage 4) choosing disjuncts in chasing the inverse of the view rules

(stage 5) chasing again with Σ

Failing means that Q does not hold after applying these choices. In the first item, we cannot make a choice of Q_{VERIFY}, since due to atomic and special views all such choices succeed.

So we assume that the failing iteration is based on Q_{START}. Note that prior to the final chasing stage, the instance formed, call it $\mathcal{I}'$, could not have any A_1 or A_2 atoms, because otherwise $\mathcal{I}'$ after the final stage would satisfy Q_{START}. Therefore $\mathcal{I}'$ prior to the final chasing stage must be a proper grid with tiles, and the final chase oes not introduce any new atoms. As we know that $\mathcal{I}'$ does not satisfy Q_{VERIFY}, it follows that it yields a proper tiling of the quarter-plane.

□

D.3 Proof of Theorem 18: undecidability with only UIDs, where the query is a CQ and the views are UCQs

Recall the statement of Theorem 18:

The problem of monotonic determinacy is undecidable when Σ ranges over Unary Inclusion Dependencies Q over Boolean CQs, and $\mathbf{V}$ over UCQ views.

The Undecidable Problem. We will use a reduction from the problem of coloring a quarter plane. Formally we define the instance of this problem as an pair $\langle \mathcal{T}, \mathcal{F} \rangle$ where $\mathcal{T}$ is a set of tiles and $\mathcal{F}$ is a set of forbidden vertical and horizontal pairs, encoded as a subset of $\mathcal{T} \times \mathcal{T} \times \{\texttt{horizontal}, \texttt{vertical}\}$. We say that tiling admits $p \in \mathcal{F}$ at coordinates (n,m) if and only if tiles at $\langle (n,m), (n+1,m) \rangle$ form p if it is horizontal or tiles at $\langle (n,m), (n,m+1) \rangle$ form p if it is vertical. Given an instance

$\langle \mathcal{T}, \mathcal{F} \rangle$ of the tiling problem we ask if there exists a tiling of a plane that admits no forbidden pair, if that is the case we call the tiling *valid*.

Let $\mathcal{I} = \langle \mathcal{T}, \mathcal{F} \rangle$ an instance of the tiling problem. Having an instance $\mathcal{I}$ of this problem, we show how to construct an instance of the monotonic determinacy problem.

Challenge. A key challenge in coding any determinacy problem is that the query Q plays two roles, highlighted in Fig. 3. On the one hand the query needs to serve as a starting point on which (using the rules) we can generate something like a large grid. On the other hand, the same query is used at the end of the process, usually to check for some violation of a correctness property in a tiling or a Turing Machine run. We have seen two ways of achieving this "dual role" for the query, which we review below.

In the proof of Theorem 16, the query is designed to play only the first role. To allow it to play the second role, we use a rule: when a violation is found a special rule is fired that makes Q hold. But this rule is not a UID, so this technique will not be available.

A second technique would be to let Q be a UCQ, with one disjunct playing the "initialization" role and the others representing detection of various violations. But this will also not be available to us, because we need to use a CQ.

The main challenge will be to code verification of a disjunction of the different tiling violations using a CQ.

The Reduction. We now present the reduction.

Signature. Our signature σ will consist of:

- Unary relations $GridSource$ and T_i for each $i \in \mathcal{T}$.
- Binary relations Net, $Axis_x$, and $Axis_y$.
- Ternary relations $Grid_x$ and $Grid_y$.

Query construction.

Define $Q_{\texttt{start}}$ as the following CQ (see Fig. 8):

$$Q_{\texttt{start}}(v_0, x_0, x_1, y_0, y_1) = Axis_x(x_0, x_1) \wedge Axis_y(y_0, y_1) \wedge GridSource(v_0).$$

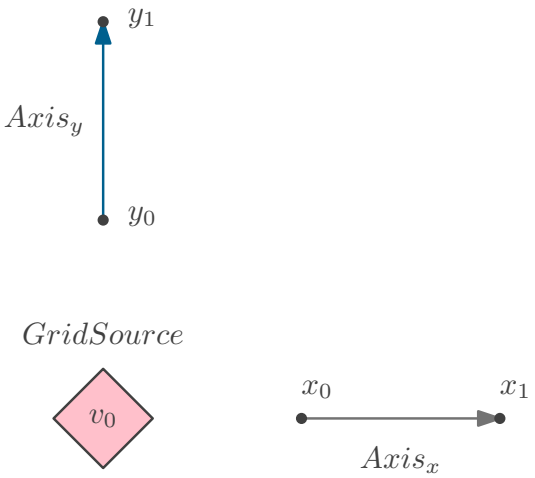

Figure 8: Visualization of $Q_{\texttt{start}}$.

For $p \in \mathcal{F}$, $p = \langle i, j, \texttt{vertical} \rangle$, define $Q^p_{\texttt{bad}}$ as the CQ:

$$Q^p_{\texttt{bad}}(v_p) = \exists v'_p, x, x', y, y', y'' \; T_i(v_p) \wedge T_j(v'_p) \wedge Grid_x(v_p, x, x') \wedge Grid_x(v'_p, x, x') \wedge Grid_y(v_p, y, y') \wedge Grid_y(v'_p, y', y'').$$

We refer to such CQs as "vertical $Q^p_{\texttt{bad}}$ queries": See Fig. 9.

For $p \in \mathcal{F}$, $p = \langle i, j, \texttt{horizontal} \rangle$ define $Q^p_{\texttt{bad}}$ as the CQ:

$$Q^p_{\texttt{bad}}(v_p) = \exists v'_p, x, x', x'', y, y' \wedge T_i(v_p) \wedge T_j(v'_p), \wedge Grid_x(v_p, x, x') \wedge Grid_x(v'_p, x', x'') \wedge Grid_y(v_p, y, y') \wedge Grid_y(v'_p, y, y)'.$$

These are "horizontal $Q^p_{\texttt{bad}}$" queries, illustrated in Fig. 10.

Both queries are visualized in the following figures:

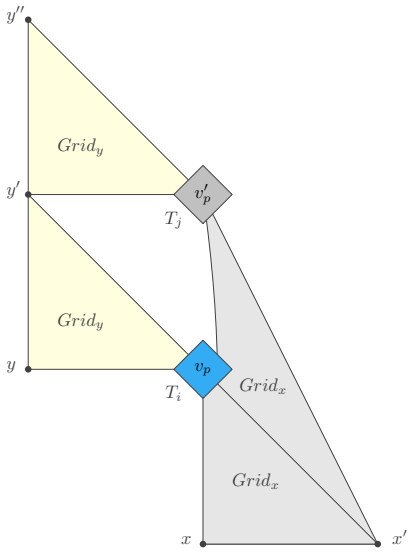

Figure 9: Visualization of a "vertical $Q^p_{\texttt{bad}}$" query.

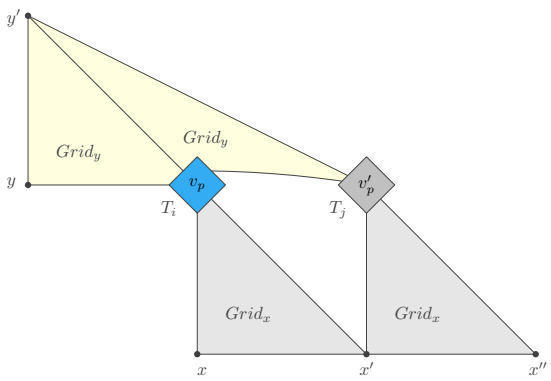

Figure 10: Visualization of a "horizontal $Q^p_{\texttt{bad}}$" query.

Let $\boldsymbol{v_p}$ be a vector $\langle v_p \rangle_{p \in \mathcal{F}}$ of variables. Define $Q_{\texttt{net}}(v_0, \boldsymbol{v_p})$ as a Net-clique over $\{v_0\} \cup \{v_p \mid p \in \mathcal{F}\}$ variables. See Fig. 11 for a visualization.

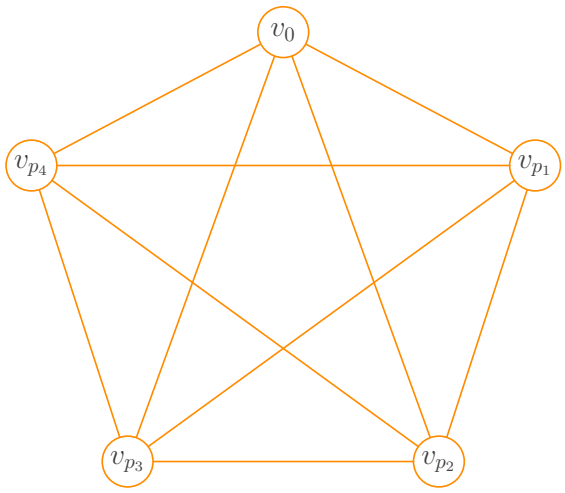

Figure 11: Complete depiction of $Q_{\texttt{net}}$ for $|\mathcal{F}| = 4$.

Finally we define $Q_{\texttt{free}}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p})$ as:

$$Q_{\texttt{start}}(v_0, x_0, x_1, y_0, y_1) \wedge Q_{\texttt{net}}(v_0, \boldsymbol{v_p}) \wedge \bigwedge_{p \in \mathcal{F}} Q^p_{\texttt{bad}}(v_p).$$

And Q as the Boolean CQ obtained by fully existentially quantifying $Q_{\texttt{free}}$. See Fig. 12 and Fig. 13.

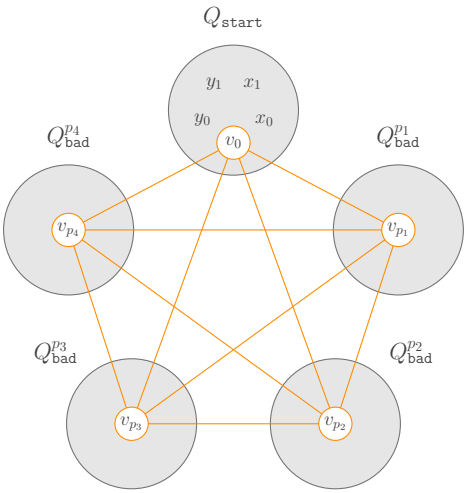

Figure 12: High level visualization of vertical $Q_{\texttt{free}}$ with free variables and Net atoms exposed.

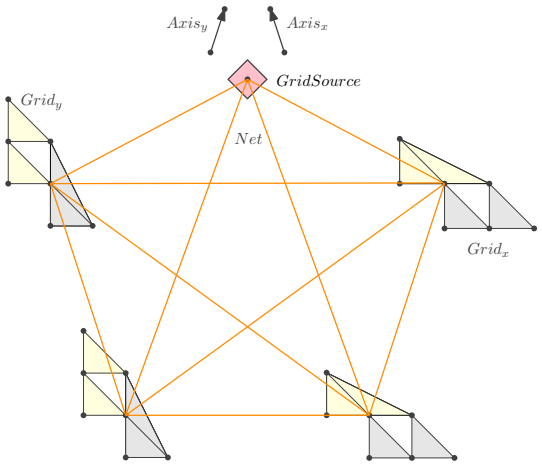

Figure 13: Structural depiction of $Q_{\texttt{free}}$ with every atom exposed.

Tuple Generating Dependencies. We will take our UIDs Σ to be:

$$Axis_x(x, x') \to \exists x'' Axis_x(x', x'') \quad \text{and} \quad Axis_y(y, y') \to \exists y'' Axis_y(y', y'').$$

The structure $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$ will then consist of $\textsc{CanonDB}(Q)$ with two infinite $Axis_x$ and $Axis_y$ chains. For clarity we name the elements of those chains $x^*_0, x^*_1, x^*_2, \ldots$ and $y^*_0, y^*_1, y^*_2, \ldots$ respectively. Moreover, to distinguish constants of $\textsc{CanonDB}(Q)$ from variables of Q we will use a star in the upper-right subscript, for example a constant v^*_0 of $\textsc{CanonDB}(Q)$ will correspond to variable v_0 of Q. Finally, we will use $\boldsymbol{v^*_p}$ to denote domain elements of the chase that correspond to $\boldsymbol{v_p}$. For further clarification see Fig. 14, which shows the structure, with the canonical database at the bottom right.

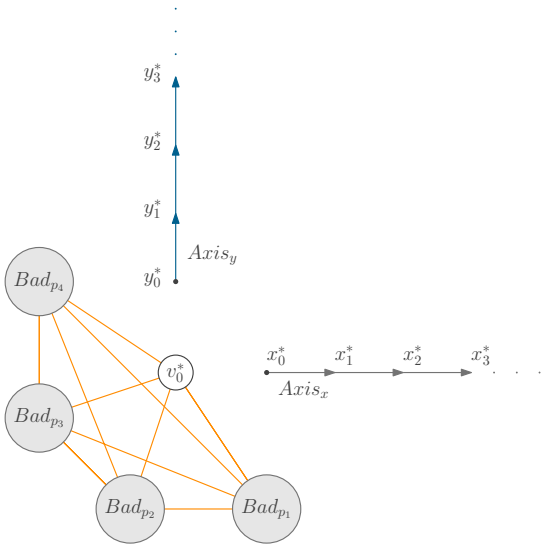

Figure 14: Depiction of the chase $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$.

View construction.
We will have a single UCQ view. Intuitively, there is a "start disjunct" that will match in the canonical database of the chase to produce many view facts. Then there will be some "unfaithful" disjuncts, representing alternative ways of producing these facts. A choice of these disjuncts will represent an attempt to tile the quarter plane. Applying Q at the final stage will correspond to checking correctness.
Define $Slot(x)$ as a conjunction of all atoms over our schema σ in one free variable x, with the exception of $Net(x, x)$. We will refer to the structure defined by this query as a *slot*. Then define $V_{\texttt{slot}}$ as $\bigwedge_{p \in \mathcal{F}} Slot(v_p)$. In the end, $Q_{\texttt{free}}$ will be a disjunct of our single UCQ view V. Note that $V_{\texttt{slot}}$ shares free variable v_p for $p \in \mathcal{F}$ with $Q_{\texttt{free}}$: v_p appears in a $Q^p_{\texttt{bad}}$ conjunct of $Q_{\texttt{free}}$. Query $V_{\texttt{slot}}$ is depicted in Fig. 15.

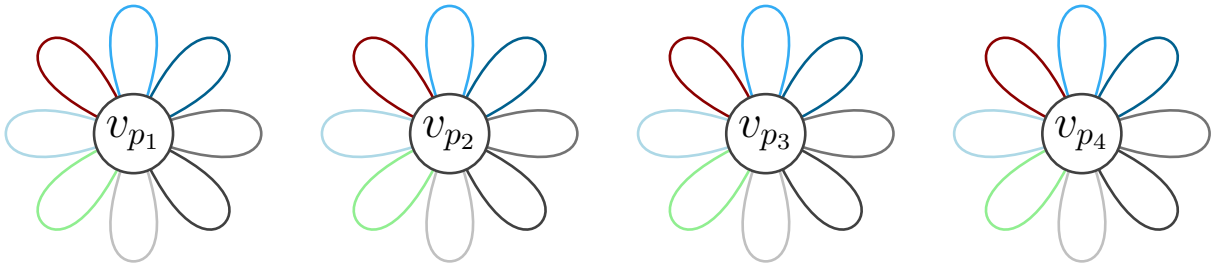

Figure 15: Depiction of $V_{\texttt{slot}}$. Note free variables shared with $Q_{\texttt{free}}$.

We define $V_{\texttt{net}}$ as $Q_{\texttt{net}}$.
Then we define $V_{\texttt{chose}_i}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p})$ as the CQ:

$$T_i(v_0) \wedge Grid_x(v_0, x_0, x_1) \wedge Grid_y(v_0, y_0, y_1) \wedge V_{\texttt{net}}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p}) \wedge V_{\texttt{slot}}(\boldsymbol{v_p}).$$

$V_{\texttt{chose}_i}$ is depicted in Fig. 16.
Again, keeping in mind that $Q_{\texttt{free}}$ will eventually be a disjunct of our single UCQ view V note that $V_{\texttt{chose}_i}$ shares free variables v_0, x_0, x_1, y_0, and y_1 with $Q_{\texttt{free}}$. Those variables appear in the $Q_{\texttt{start}}$ conjunct of $Q_{\texttt{free}}$.

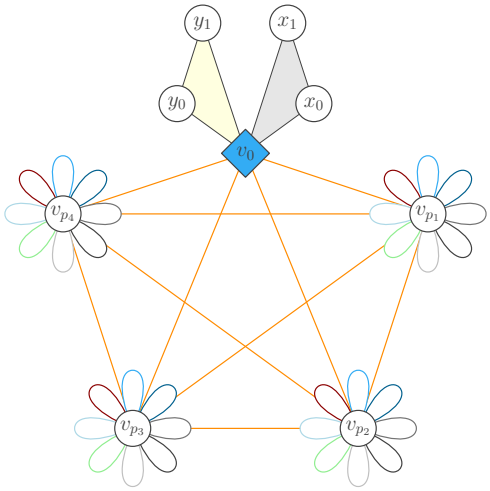

Figure 16: Depiction of $V_{\texttt{chose}_i}$.

MonDet$(Q, \mathbf{V}, \Sigma)$:
1: **for** Q_n approximation of Q **do**
2: $C_n :=$ Chase$_\Sigma$(CanonDB(Q_n))
3: $\mathcal{J}_n := \mathbf{V}(F_n)$
4: **for** $Q'_{m,n} \in$ BackV$_{\mathbf{V}}(D_n)$ **do**
5: $C'_{m,n} :=$ Chase$_\Sigma(Q'_{m,n})$
6: **if** $C'_{m,n} \not\models Q$ **then**
7: **return false**
8: **return true**

Figure 17: Process for checking monotonic determinacy.

We let $V_{\texttt{unfaithful}}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p}) = \bigvee_{i \in \mathcal{T}} V_{\texttt{chose}_i}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p})$. And finally we are able to define the unique view of our schema:

$$V(x_0, x_1, y_0, y_1, \boldsymbol{v_p}) = \exists v_0\ Q_{\texttt{free}}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p}) \vee V_{\texttt{unfaithful}}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p})$$

Correctness of the reduction. With Q, V, Σ constructed, let us show that the transformation is a correct reduction from tiling to monotonic determinacy. We consider the process Fig. 3, restated for convenience in Figure 17. For the remainder of this section we will focus on proving the following lemma:

Lemma 12. *There exists a valid tiling of $\langle \mathcal{T}, \mathcal{F} \rangle$ if and only if* MonDet$(Q, \mathbf{V}, \Sigma)$ *returns false.*

We start by taking the view image of the chase of the canonical database. We note that the "unfaithful disjuncts" will not contribute anything:

Claim 6. *$V_{\texttt{unfaithful}}$ does not map to* Chase$_\Sigma$(CanonDB(Q)).

Proof. $V_{\texttt{unfaithful}}$ contains $Grid_x$ and $Grid_y$ atoms, which do not appear in Q. □

In contrast, $Q_{\texttt{free}}$ will map to Chase$_\Sigma$(CanonDB(Q)) and produce a number of view facts.

Claim 7. *Let h be any homomorphism from $Q_{\texttt{free}}$ to* Chase$_\Sigma$(CanonDB(Q)). *Then $h(v_i) = v_i^*$ for any variable subscript i. Thus different view facts generated by such an h can only differ on the "axis-related variables" x_0, x_1, y_0, and y_1.*

Proof. Note that:

1. For every $Q^p_{\texttt{bad}}$, its only automorphism is the identity.
2. Every $Q^p_{\texttt{bad}}$ can be mapped only to itself.
3. Variables of $Q_{\texttt{net}}$ need to be mapped bijectively onto themselves due to the Net-clique structure of $Q_{\texttt{net}}$.

The claim follows easily from these observations. □

Thus we will generate a collection of view facts for different matches of $Q_{\texttt{free}}$. Now we characterize exactly what these view facts are.

Claim 8. *Let H be the set of homomorphisms from $Q_{\texttt{free}}$ to $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$. Then $\{h(x_0, x_1, y_0, y_1) \mid h \in H\} = \left\{\langle x^*_n, x^*_{n+1}, y^*_m, y^*_{m+1}\rangle \mid n, m \in \mathbb{N}\right\}$.*

Proof. Here we note that:

1. The set of $Axis_x$ atoms contained in $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$ is $\left\{Axis_x(x^*_n, x^*_{n+1}) \mid n \in \mathbb{N}\right\}$.
2. The set of $Axis_y$ atoms $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$ contains is $\left\{Axis_y(y^*_m, y^*_{m+1}) \mid m \in \mathbb{N}\right\}$.
3. Atom $Axis_x x_0, x_1$ can be freely mapped to any $Axis_x$ atom of $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$.
4. Atom $Axis_y y_0, y_1$ can be freely mapped to any $Axis_y$ atom of $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$.

The claim follows directly from these observations. □

Thus, we conclude that we have view facts corresponding to axis-related variables as follows:

Lemma 13. *When applying the view $V(x_0, x_1, y_0, y_1, \boldsymbol{v_p})$ to $\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))$, we get view facts for exactly these tuples:*

$$\left\{\langle x^*_n, x^*_{n+1}, y^*_m, y^*_{m+1}, \boldsymbol{v^*_p}\rangle \mid n, m \in \mathbb{N}\right\}.$$

Proof.

$$V(\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))) = Q_{\texttt{free}}(\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))) \quad \text{(Claim 6)}$$

$$Q_{\texttt{free}}(\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q))) = \left\{\langle x^*_n, x^*_{n+1}, y^*_m, y^*_{m+1}, \boldsymbol{v^*_p}\rangle \mid n, m \in \mathbb{N}\right\} \quad \text{(Claims 7 and 8)}$$

□

Now let us inspect the structures considered in Line 4 of $\textsc{MonDet}(Q, \mathbf{V}, \Sigma)$. For each view fact, we will choose a view definition, either as a $Q_{\texttt{free}}$ disjunct or an unfaithful disjunct. Whenever a structure is created by choosing a $Q_{\texttt{free}}$ disjunct of V, Q will be satisfied (the return in Line 6 will not fire), and thus we continue with the process.
Let UT, the set of *unfaithful tests*, be the set of all structures considered in the **for** loop of Fig. 3 that are created by taking a $V_{\texttt{unfaithful}}$ disjunct of V.
Consider applying the "inverse rule" corresponding to $V_{\texttt{unfaithful}}$, applied to an element $\langle x^*_n, x^*_{n+1}, y^*_m, y^*_{m+1}, \boldsymbol{v^*_p}\rangle$ within $V(\textsc{Chase}_\Sigma(\textsc{CanonDB}(Q)))$. As $V_{\texttt{unfaithful}}$ is a UCQ this means applying one of its $V_{\texttt{chose}_t}$ disjuncts. More precisely, we apply the inverse of $\exists v_0\, V_{\texttt{chose}_t}(v_0, x_0, x_1, y_0, y_1, \boldsymbol{v_p})$ to $\langle x^*_n, x^*_{n+1}, y^*_m, y^*_{m+1}, \boldsymbol{v^*_p}\rangle$.[3]
From now on, for convenience, we rename the fresh element created from applying a step for an unfaithful test above as $v^*_{i,j}$. Also, let:

$$\textsc{TileAt}^t_{n,m} ::= V_{\texttt{chose}_t}(v^*_{i,j}, x^*_m, x^*_{n+1}, y^*_m, y^*_{m+1}, \boldsymbol{v^*_p}).$$

Definition 16. *Let $f \in \mathcal{T}^{\mathbb{N}\times\mathbb{N}}$ be a function encoding a tiling of a quarter plane. Then let*

$$\textsc{Tiling}_f ::= \bigcup_{\langle n,m\rangle \in \mathbb{N}\times\mathbb{N}} \textsc{TileAt}^{f(n,m)}_{n,m}.$$

From Lemma 13 we conclude that the structures of interest to us in Line 4 – that is, those contained in UT – will have the shape of the structures above.

Corollary 2. *For every (not necessarily valid), tiling $f \in \mathcal{T}^{\mathbb{N}\times\mathbb{N}}$ of the quarter plane the structure $\textsc{Tiling}_f$ is contained (up to isomorphism) in* UT. *Conversely, every member of* UT *is isomorphic to such a structure.*

Proof. Consider Definition 16. Then the range of i and j comes from Lemma 13, and the range of f comes from disjuncts of $V_{\texttt{unfaithful}}$. □

[3] Note, that the variable v_0 is existentially quantified in V.

This means that each element of UT is a grid-like structure, part of which is depicted on Fig. 18.

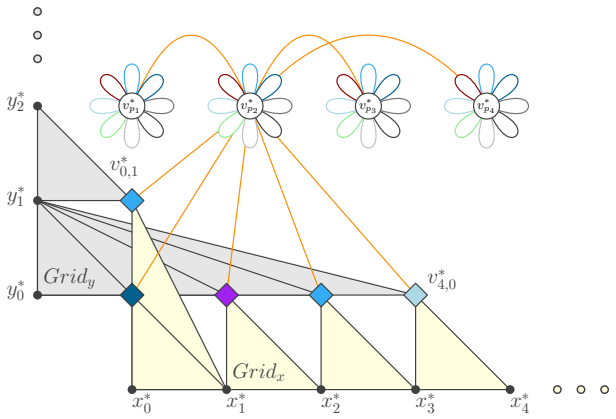

Figure 18: Partial depiction of a structure UT.

In the absence of any assumption of monotonic determinacy, the structures in UT, based on the $T_i(v_{m,n})$ atoms, define all tilings of a quarter plane, including the invalid ones.

Definition 17. *For $p \in \mathcal{F}$ and natural n, m define a mapping $\texttt{BadMatch}_{\texttt{p,n,m}}$ from the variables of $Q^p_{\texttt{bad}}$ to the domain of* UT *as:*

$$\texttt{BadMatch}_{\texttt{p,n,m}} = \begin{cases} \langle x, x,', x'', y, y', v_p, v_p' \rangle \mapsto \langle x_n^*, x_{n+1}^*, x_{n+2}^*, y_m^*, y_{m+1}^*, v_{n,m}^*, v_{n+1,m}^* \rangle & \textit{when } p \textit{ is a horizontal pattern,} \\ \langle x, x,', y, y', y'', v_p, v_p' \rangle \mapsto \langle x_n^*, x_{n+1}^*, y_m^*, y_{m+1}^*, y_{m+2}^*, v_{n,m}^*, v_{n,m+1}^* \rangle & \textit{when } p \textit{ is a vertical pattern.} \end{cases}$$

Claim 9. *There exists $p \in \mathcal{F}$ appearing at coordinates*[4] *(n, m) in the tiling defined by* UT $\in$ UT *if and only if $\texttt{BadMatch}_{\texttt{p,n,m}}$ is a homomorphism from $Q^p_{\texttt{bad}}$ to* UT.

Proof. Let $f \in \mathcal{T}^{\mathbb{N}\times\mathbb{N}}$ define the tiling of UT.
($\Rightarrow$) Assume that one of the horizontal forbidden patterns $p = \langle i, j, \texttt{horizontal} \rangle \in \mathcal{F}$ appears in the tiling defined by f at coordinates $\langle n, m \rangle$. That is, $\langle f(n, m), f(n + 1, m) \rangle = \langle i, j \rangle$. Then $\texttt{BadMatch}_{\texttt{p,n,m}}$ clearly defines a homomorphism from $Q^p_{\texttt{bad}}$ to UT. The case for vertical forbidden pattern is analogous.
($\Leftarrow$) Assume that $\texttt{BadMatch}_{\texttt{p,n,m}}$ is a homomorphism from $Q^p_{\texttt{bad}}$ to UT. Assume $p = \langle i, j, \texttt{horizontal} \rangle$, then $\langle f(n, m), f(n + 1, m) \rangle = \langle i, j \rangle$ thus p appears in the tiling defined by UT. □

Thus we know that incorrectness of a tiling associated with a member UT of UT corresponds to mapping one of the $Q^p_{\texttt{bad}}$ queries, which are a component of Q, into UT. We still need to show that an unsuccessful tiling corresponds to mapping the entirety of Q, which involves a mapping witnessing all $Q^p_{\texttt{bad}}$ queries at once. That is, we need to solve the challenge mentioned early, due to the fact that Q is a CQ.
Informally, the idea is as follows. In one direction, suppose the tiling is invalid. This is witnessed by the presence of some forbidden pattern p. Towards defining a homomorphism of Q, we map v_0 to the constant v_p^*, and the other variables $v_{p'}$ to the constants $v_{p'}^*$ corresponding to themselves. For a given $p' \neq p$, we can extend the mapping to become a homomorphism of $Q^{p'}_{\texttt{bad}}$ in a "vacuous way" – by mapping all quantified variables of $Q^{p'}_{\texttt{bad}}$ into $v_{p'}^*$. This will give us a homomorphism because $v_{p'}^*$ satisfies each unary predicate, other than Net self-loops. Turning to the variable v_p, we can extend to a homomorphism of $Q^p_{\texttt{bad}}$, because p is a violation.
In the other direction, suppose we have a homomorphism h of Q. All variables v_p must map either to constant $v_{p'}^*$ for some p', or to some constant $v_{n,m}^*$. Because there are no Net self-loops, these variables must map injectively. Thus there is one variable v_p which must map on to some constant $v_{n,m}^*$. The fact that h is a homomorphism and the properties of $v_{n,m}^*$ imply that the forbidden pattern p occurs in the tiling.
We now explain more formally.

Lemma 14. *The tiling defined by* UT $\in$ UT *is invalid if and only if Q maps to* UT.

Proof. We will start with a number of observations. Recall that from Corollary 2 the shape of UT is as in Definition 16.
Consider a $Q_{\texttt{start}}$ conjunct. It consists of three atoms: $Axis_x(x_0, x_1)$, $Axis_x(y_0, y_1)$, and $GridSource(v_0)$. For x_0 and x_1, the only way to map them homomorphically is to use one of $\{\langle x_0, x_1 \rangle \mapsto \langle v_p^*, v_p^* \rangle \mid p \in \mathcal{F}\}$ mappings as $Axis_x$ (and $Axis_y$)

[4] Recall that this means tiles at $\langle (n, m), (n + 1, m) \rangle$ form p if it is horizontal, or tiles at $\langle (n, m), (n, m + 1) \rangle$ form p if it is vertical.

hold only for elements of $\boldsymbol{v_p^*}$ in UT. The same can be said about $\langle y_0, y_1 \rangle$. From now on, we will disregard the mapping of x_0, x_1, y_0, and y_1 variables, as they can be always properly mapped to UT, and are not appearing in $Q_{\texttt{net}}$.
The atom $GridSource(v_0)$ can only be homomorphically mapped to a $Slot(v_p^*)$ substructure of UT, since $GridSource$ atoms appear only there.
Consider a $Q^p_{\texttt{bad}}$ conjunct of Q. Note that it can be homomorphically mapped to any $Slot(v^*_{p'})$, no matter which p' is considered, as all atoms except Net hold for $v_{p'}$. Observe that there are no other mappings of $Q^p_{\texttt{bad}}$ to UT when elements of $\left\{\texttt{BadMatch}_{\texttt{p,n,m}} \mid n, m \in \mathbb{N}\right\}$ are disregarded[5].
Now let us consider $Q_{\texttt{net}}$. The variables in consideration are those in $\boldsymbol{v_p}$ and v_0. As $\exists x\ Net(x, x)$ does not hold in UT, all of those variables need to be mapped to different elements of UT. Moreover due to the shape of UT the variables in $\boldsymbol{v_p}$ and v_0 have to be mapped to one of the following sets of variables $\left\{\boldsymbol{v_p^*} \cup \{v_{n,m}\} \mid n, m \in \mathbb{N}\right\}$.
Let us gather together what we noted so far about homomorphisms from Q to UT:

($\diamondsuit$) v_0 is always mapped to a variable of $\boldsymbol{v_p}$.

($\spadesuit$) All variables $Q^p_{\texttt{bad}}$ conjuncts can only be mapped to a single $v^*_{p'}$ of UT except for mappings defined by elements of $\left\{\texttt{BadMatch}_{\texttt{p,n,m}} \mid n, m \in \mathbb{N}\right\}$ that are homomorphisms.

($\heartsuit$) Variables in $\boldsymbol{v_p}$ and v_0 have to be injectively mapped to one of the following sets of variables $\left\{\boldsymbol{v_p^*} \cup \{v_{n,m}\} \mid n, m \in \mathbb{N}\right\}$.

($\clubsuit$) Any mapping from $\texttt{vars}(Q) \setminus \{x_0, x_1, y_0, y_1\}$ to the domain of UT that is a homomorphism can always be extended to variables x_0, x_1, y_0, and y_1.

From the above reasoning it should be clear, that a mapping of $\texttt{vars}(Q) \setminus \{x_0, x_1, y_0, y_1\}$ to the domain of UT is a homomorphism if and only if it satisfies conditions ($\diamondsuit$), ($\spadesuit$), and ($\heartsuit$).

Let $f \in \mathcal{T}^{\mathbb{N}\times\mathbb{N}}$ define the tiling of UT.
($\Rightarrow$) Assume that one of the horizontal forbidden patterns $p = \langle i, j, \texttt{horizontal}\rangle \in \mathcal{F}$ appears in the tiling defined by f at coordinates $\langle n, m\rangle$. That is $\langle f(n, m), f(n + 1, m)\rangle = \langle i, j\rangle$. We describe a mapping of $\texttt{vars}(Q) \setminus \{x_0, x_1, y_0, y_1\}$ to the domain of UT:

- v_0 is mapped to v_p. ($\diamondsuit$)
- Variables of $Q^p_{\texttt{bad}}$ are mapped to UT via $\texttt{BadMatch}_{\texttt{p,n,m}}$. satisfies ($\spadesuit$) from Claim 9
- For $p' \in \mathcal{F}$ s.t. $p' \neq p$ variables of $Q^{p'}_{\texttt{bad}}$ are mapped to $v^*_{p'}$. trivially satisfies ($\spadesuit$)

Note that ($\heartsuit$) is also satisfied. From ($\clubsuit$) we conclude that there exists a homomorphism from Q to UT.
($\Leftarrow$) Let h be a witness homomorphism from Q to UT. We know that h satisfies conditions ($\diamondsuit$), ($\spadesuit$), and ($\heartsuit$). Moreover there are only $|\mathcal{F}|$ $Slot$ substructures in UT, there are $|\mathcal{F}|$ $Q^p_{\texttt{bad}}$ conjuncts in UT, and h maps v_0 to one of the elements in $\boldsymbol{v_p^*}$. Thus we can conclude that one of the $Q^p_{\texttt{bad}}$ is mapped via $\texttt{BadMatch}_{\texttt{p,n,m}}$ to UT. Then from Claim 9 we observe that UT encodes an invalid tiling.

□

D.4 Rewritability for entailment problems with full TGDs, via inverse rules and forward-backward

In this appendix we will give an additional rewritability result for full TGDs: with CQ views and Datalog queries, monotonically determined queries have Datalog rewritings. This is of independent interest, although it is a variation of a standard result in the absence of background knowledge in the form of rules. We will use it in the proof of one of the undecidability results, Theorem 19, which claims not just undecidability of monotonic determinacy, but undecidability of Datalog rewritability.

Recall the notion of Q-entailing from the body of a paper: a view instance entails Q with respect to $\mathbf{V}$, Σ if for every instance satisfying Σ whose view image contains the given view instance, Q holds.

A $\mathbf{V}$, Σ *certain answer rewriting of* Q is a query R on the view schema such that R holds on $\mathcal{J}$ exactly when $\mathcal{J}$ is Q-entailing with respect to Σ, $\mathbf{V}$.

Notice that a certain answer rewriting is required to simulate entailment on all instances of the view schema, not just the ones that are view images of base instances satisfying the rules.

In particular, if Q is monotonically determined by $\mathbf{V}$ with respect to Σ, and R is a $\mathbf{V}$, Σ certain answer rewriting of Q, then R is also a rewriting of Q over the views

Theorem 24. *If Q is in Datalog, $\mathbf{V}$ are CQ views, and Σ consists of full TGDs, then we can compute a $\mathbf{V}$, Σ certain answer rewriting in Datalog.*

[5]Note that some $\texttt{BadMatch}_{\texttt{p,n,m}}$ might be a homomorphism from $Q^p_{\texttt{bad}}$ to UT.

Proof. The approach is a variation of the standard "inverse rules" algorithm (Duschka, Genesereth, and Levy 2000). Q-entailment corresponds to entailment with Σ and the "backward view rules" $v(\boldsymbol{x}) \rightarrow \phi_v(\boldsymbol{x})$. In the case of CQ views, the latter are linear source-to-target TGDs. We can convert the existentials in the head to skolem functions, and consider the view predicates as extensional predicates, the base predicates as intensional. Then the backward view rules become Datalog rules with Skolem functions. Composing these with the TGDs and the rules of Q, we get a Datalog program with Skolem functions. But since the rules of Σ and of Q never introduce Skolems, the depth of functions in Skolem terms never goes above 1. Then the Skolem functions can be simulated with additional predicates. See (Duschka, Genesereth, and Levy 2000) or (Benedikt et al. 2023) for details on how to perform this simulation. □

Corollary 3. *If Q is in Datalog, $\mathbf{V}$ are CQ views, and Σ consists of full TGDs, and Q is monotonically determined by $\mathbf{V}$ with respect to Σ, then there is a rewriting in Datalog.*

D.5 Undecidability results for the case of full TGDs, proof of Theorem 19

Using the forward-backward technique, we showed decidability of monotonic determinacy in cases where the BTVIP holds and the rules, queries, and views are reasonably well-behaved – e.g. FGDL views and FGTGD rules. All of our undecidability results will rely on *failure* of the BTVIP: we need the first steps of Algorithm 3 – unfolding the query, chasing, applying views – to build a grid-like structure.

Recall that the chase terminates if for every finite instance we can apply a finite number of chase steps to reach another finite instance where the rules hold. In the body of the paper, we focused on rules where the chase is not guaranteed to terminate. The chase can fail to terminate even for the simplest rules used in our main undecidability results, which involve UIDs: see Theorem 18.

An obvious question is what happens if we only have terminating chase, but do not have the bounded treewidth property. Obviously, if we are in one of the situations where we have undecidability without rules, we are still undecidable.

But can adding rules with terminating chase move us from decidability to undecidability?

The answer is yes. We illustrate this with the case where the query is in MDL and the views are CQs. This case is decidable in the absence of rules, using the BTVIP: see Figure 1. Recall from the body that *full TGDs* are TGDs with no existentials in the head. Clearly, applying chase steps to a finite instance with full TGDs always leads to a finite instance where the rules hold – that is, the chase always terminates.

We now prove Theorem 19, which we recall:

The problem of monotonic determinacy is undecidable when Q ranges over MDL, $\mathbf{V}$ over atomic views, and Σ over full TGDs. Likewise the problem of Datalog rewritability, CQ rewritability, and the variants of all these problems over finite instances.

Notice first that by Corollary 3, monotonic determinacy is the same as Datalog rewritability for this class of $Q, \mathbf{V}, \Sigma$. Notice also that since the chases of all instances involved will be finite, finite controllability of monotonic determinacy (as well as Datalog or CQ rewritability) will be obvious.

Thus our strategy will consist of first showing a family of examples where monotonic determinacy is undecidable. And this example will have some additional properties that will suffice to show that *whenever they are monotonically determined, they are CQ rewritable*. From this last fact, it will follow from undecidability of monotonic determinacy that CQ rewritability and Datalog rewritability are undecidable.

The first property they will have is that *all view predicates are unary atomic queries*. In particular, when we chase the view images of approximations of the query Q, as in the early steps of Algorithm 3, we will get a database with only unary atoms. If we turn each such database into a Boolean query, we get an infinite disjunction of CQs with only unary atoms, and clearly any such disjunction degenerates to a UCQ. But in fact, *there will only be one possible view image*, so the corresponding query is a CQ. Thus if the query is monotonically determined, the CQ formed from the view image will be a rewriting.

We now give the details.

Deterministic Turing Machines Throughout this section, we will utilize a deterministic Turing Machine (TM) variant operating on a finite tape. Formally, we define a TM as a tuple $\langle \mathcal{A}, \texttt{Blank}, \texttt{Left}, \texttt{Right}, \mathcal{S}, s_{\text{start}}, s_{\text{end}}, \Delta \rangle$, consisting of the tape alphabet $\mathcal{A}$ with a distinguished symbol `Blank` and symbols `Left` and `Right`, signifying the left and right ends of the tape, respectively. The set of states $\mathcal{S}$ includes the distinguished starting and halting states s_{start} and s_{end}, along with the transition function Δ. We assume that a $\mathcal{M}$ is not allowed to traverse beyond the tape or change the tape cells marked with `Left` and `Right`, and that in a state s_{end}, it halts the execution.

Halting problem. Given a tape, we say that it is *initial* when its first cell is filled with `Left`, the last is filled with `Right` and the rest are filled with `Blank`. The problem of determining whether there exists i such that a given TM $\mathcal{M}$ halts when starting from the state s_{start} with its head over the leftmost tape cell of the initial tape of length i blank-filled tape tape of length i with its edges marked by `Left` and `Right` is undecidable. We say that TM $\mathcal{M}$ *halts* if there exists such natural i.

Relational encoding. In a later construction, we will encode the above-defined machine in a relational structure. To this end, we define a set of binary predicates representing cell states of the tape over time. Let $\mathcal{C}$ be the set $\mathcal{A} \cup \mathcal{S} \times \mathcal{A}$, where each of

its elements is treated as a binary predicate symbol. Here, the elements of $\mathcal{A}$ represent cells without the head of the TM, and $\mathcal{S} \times \mathcal{A}$ is used to represent a cell content with a head over it. We intend to use the first position of each symbol in $\mathcal{C}$ to represent a position on the tape (space), and the second to represent a step number of the TM's execution (time). Thus, $\mathtt{A}(s,t)$, for $\mathtt{A} \in \mathcal{C}$, should be read as ''the s^{th} cell at time t contains $\mathtt{A}$." We will consistently use variables s, and t to denote "space and time coordinates".

Exploiting determinism. Given a TM $\mathcal{M}$ and an initial tape T of length i, let $CellAt_i(s,t) : \mathbb{N} \times \mathbb{N} \to \mathcal{C}$ be a partial binary function returning the contents of the s^{th} tape cell in the time t of the run of $\mathcal{M}$ over T. Note that there is a functional dependence between:

$$\begin{aligned}
CellAt_i(1,t),\ CellAt_i(2,t) \quad &\text{and} \quad CellAt_i(1,t+1), && \text{for any } t \\
CellAt_i(s-1,t),\ CellAt_i(s,t),\ CellAt_i(s+1,t) \quad &\text{and} \quad CellAt_i(s,t+1), && \text{for any } t \text{ and } 1 < s < i \\
CellAt_i(i-1,t),\ CellAt_i(i,t) \quad &\text{and} \quad CellAt_i(s,t+1). && \text{for any } t
\end{aligned}$$

as TM is deterministic. Let functions $\delta_{\text{left}} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, $\delta_{\text{mid}} : \mathcal{C} \times \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, and $\delta_{\text{right}} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ represent those dependencies. Note that δ_{left} and δ_{right} are not constant, as the head might appear over the edges of the tape.

The reduction In this section we show the reduction from the non-halting problem to monotonic determinacy problem from Theorem 19. From now on fix a DTM $\mathcal{M} = \langle \mathcal{A}, \mathtt{Blank}, \mathtt{Left}, \mathtt{Right}, \mathcal{S}, s_{\text{start}}, s_{\text{end}}, \Delta \rangle$. We use the signature σ consisting of a set of unary symbols $\{\mathtt{First}, \mathtt{Last}\}$ and a set of binary symbols $\{\mathtt{Succ}, \mathtt{Succ}^+\} \cup \mathcal{C}$.

Query. Let Q be the following Boolean MDL query:

$$\begin{aligned}
\mathtt{Goal} &:= \mathtt{Reached}(x) \wedge \mathtt{Last}(x) \\
\mathtt{Reached}(x) &:= \mathtt{First}(x) \\
\mathtt{Reached}(y) &:= \mathtt{Reached}(x) \wedge \mathtt{Succ}(x,y)
\end{aligned}$$

Views. Let V be the following set of unary atomic views:

$$\begin{aligned}
V_{\mathtt{First}}(x) &= \mathtt{First}(x) \\
V_{\mathtt{Last}}(x) &= \mathtt{Last}(x)
\end{aligned}$$

Constraints. Let Σ be the set of full TGDs formed by the union of four disjoint below-defined sets $\Sigma_{\mathtt{Succ}^+}, \Sigma_{\text{setup}}, \Sigma_{\Delta} \Sigma_{\text{bad}}$. As noted before, we will use variables s, and t to denote "space and time coordinates". We allow for a few comments in the definition to explain their intended meanings. Finally, the definition order reflects the execution order during the chase.

$\Sigma_{\mathtt{Succ}^+}$ computes transitive closure of $\mathtt{Succ}$:

$$\begin{aligned}
\mathtt{Succ}(x,y) &\to \mathtt{Succ}^+(x,y) \\
\mathtt{Succ}^+(x,y), \mathtt{Succ}(y,z) &\to \mathtt{Succ}^+(x,z)
\end{aligned}$$

Σ_{setup} sets up the relational encoding of the initial tape (binary atoms in first two TGDs are necessary for Proposition 23):

$$\mathtt{First}(s) \wedge \mathtt{First}(t) \wedge \mathtt{Succ}(t,t') \to \langle s_{\text{start}}, \mathtt{Left} \rangle(s,t)$$
(the first cell contains $\mathtt{Left}$ and the head)

$$\mathtt{Last}(s) \wedge \mathtt{First}(t) \wedge \mathtt{Succ}(t,t') \to \mathtt{Right}(s,t)$$ (the last cell contains $\mathtt{Right}$)

$$\mathtt{First}(s) \wedge \mathtt{Succ}^+(s,s') \wedge \mathtt{Succ}^+(s',s'') \wedge \mathtt{Last}(s'') \wedge \mathtt{First}(t) \to \mathtt{Blank}(s',t)$$
(every cell between the first and the last contains $\mathtt{Blank}$)

Σ_{Δ} simulates $\delta_{\text{left}}, \delta_{\text{mid}}$, and δ_{right} and contains:

$$\mathtt{First}(s) \wedge \mathtt{Succ}(s,s') \wedge \mathtt{A}(s,t) \wedge \mathtt{B}(s',t) \wedge \mathtt{Succ}(t,t') \to \delta_{\text{left}}(\mathtt{A},\mathtt{B})(s,t') \quad \text{(for every } (\mathtt{A},\mathtt{B}) \in dom(\delta_{\text{left}}))$$

$$\mathtt{Succ}(s,s') \wedge \mathtt{Succ}(s',s') \wedge \mathtt{A}(s,t) \wedge \mathtt{B}(s',t) \wedge \mathtt{C}(s'',t) \wedge \mathtt{Succ}(t,t') \to \delta_{\text{mid}}(\mathtt{A},\mathtt{B},\mathtt{C})(s',t')$$
(for every $(\mathtt{A},\mathtt{B},\mathtt{C}) \in dom(\delta_{\text{mid}})$)

$$\mathtt{Succ}(s,s') \wedge \mathtt{Last}(s') \wedge \mathtt{A}(s,t) \wedge \mathtt{B}(s',t) \wedge \mathtt{Succ}(t,t') \to \delta_{\text{right}}(\mathtt{A},\mathtt{B})(s',t')$$
(for every $(\mathtt{A},\mathtt{B}) \in dom(\delta_{\text{right}})$)

Σ_{bad} contains, for every $\mathtt{S} \in \mathcal{C}$ that encodes a tape cell containing the head of $\mathcal{M}$ in a non-halting state:

$$\mathtt{S}(c,t) \wedge \mathtt{Last}(t) \to \mathtt{First}(t)$$

Reduction - proof of correctness.

Note that Q is a very simple MDL query. We have:

Proposition 21. *The set of CQ approximations of Q consists of the following instances Q_i, for every $i \in \mathbb{N}$:*

$$\texttt{First}(a_1), \texttt{Succ}(a_1, a_2), \ldots, \texttt{Succ}(a_{n-1}, a_n), \texttt{Last}(a_n).$$

As the case for Q_0 is always trivial ($Q_0 = \{\texttt{First}(x_1), \texttt{Last}(x_1)\}$) - the set of views determines the query under the rules, from now on we shall consider only cases for Q_i with $i \geq 1$.

Consider Fig. 3, and let Q'_i be the instance $\textsc{BackV}_V(V(\textsc{Chase}_\Sigma(Q_i)))$. Note Q'_i is a single instance as $\textsc{BackV}_V$ is deterministic for CQ views. Now, the intermediate goal is to connect satisfaction of Q in Q'_i with a non-halting state appearing in a precise time in the chase of Q_i.

Proposition 22. *Instance Q'_i consists only of unary atoms.*

Proof. Note that V consists of unary and atomic views. □

Proposition 23. *Instances Q'_i and $\textsc{Chase}_\Sigma(Q'_i)$ are equal.*

Proof. Note that every body of TGDs in Σ contains at least one binary atom, then the proposition follows from Proposition 22. □

The following is an immediate conclusion from the above. Note that the MDL query Q can be satisfied over an instance $\mathcal{I}$ containing only unary atoms if and only if $\mathcal{I} \models \exists x\ \texttt{First}(x) \wedge \texttt{Last}(x)$.

Corollary 4. $\textsc{Chase}_\Sigma(Q_i) \models \exists x\ \texttt{First}(x) \wedge \texttt{Last}(x)$ *iff* $\textsc{Chase}_\Sigma(Q'_i) \models Q$.

Proposition 24. $\textsc{Chase}_\Sigma(Q_i) \models \exists x\ \texttt{First}(x) \wedge \texttt{Last}(x)$ *iff* $\textsc{Chase}_\Sigma(Q_i) \models \exists s, t\ \texttt{S}(s,t) \wedge \texttt{Last}(t)$ *for some $\texttt{S} \in \mathcal{C}$ that encodes a tape cell containing the head of $\mathcal{M}$ in a non-halting state.*

Proof. First, note that $Q_i \not\models \exists x\ \texttt{First}(x) \wedge \texttt{Last}(x)$. From this, it is enough to note that only TGDs from Σ_{bad} can derive $\texttt{First}$ atoms, and that no $\texttt{Last}$ atom can be derived during the chase. □

We are now ready to connect satisfaction of Q in Q'_i with the Turing Machine:

Corollary 5. $\textsc{Chase}_\Sigma(Q'_i) \models Q$ *iff* $\textsc{Chase}_\Sigma(Q_i) \models \exists s, t\ \texttt{S}(s,t) \wedge \texttt{Last}(t)$ *for some $\texttt{S} \in \mathcal{C}$ that encodes a tape cell containing the head of $\mathcal{M}$ in a non-halting state.*

Proof. Follows directly from Corollary 4 and Proposition 24 □

Now we want to argue the chase over Q_i simulates the first i steps of the run of TM $\mathcal{M}$ over initial tape of length i.

Proposition 25. *For every natural number i and every pair of natural numbers $s, t \leq i$, and every $\texttt{A} \in cencodings$ we have:*

$$CellAt_i(s,t) = \texttt{A} \quad \textit{iff} \quad \texttt{A}(x_s, x_t) \in \textsc{Chase}_\Sigma(Q_i).$$

Proof. Consider Σ_{setup}, it should be clear that the lemma holds for time $t = 1$, that is for the initial initial tape of length i.

Finally, note how each application of TGDs from Σ_Δ simulate δ_{left}, δ_{mid}, and δ_{right}. The rest of the lemma follows by a simple inductive argument. □

We can conclude the following from combining Corollary 5 and Proposition 25.

Corollary 6. *The following are equivalent:*

- $\textsc{Chase}_\Sigma(Q'_i) \models Q$, *for every i.*
- *TM $\mathcal{M}$ does not halt.*

From this, we conclude that the monotonic decidability problem for MDL queries, unary CQ views, and full TGDs is undecidable.

For the undecidability of CQ rewritability and Datalog rewritability, we observe that we have fulfilled the requirements promised before we began the construction. The construction guarantees that whenever monotonic determinacy holds, the view image is always isomorphic to the instance $\{V_{\texttt{First}}(x), V_{\texttt{Last}}(x), V_{\texttt{First}}(y)\}$. Thus monotonic determinacy is equivalent to the existence of a CQ rewriting: the CQ obtained by turning this view image into a CQ by existentially quantifying the variables.

This concludes the proof of Theorem 19.