On Monotonic Determinacy and Rewritability For Recursive Queries and Views

Michael Benedikt
University of Oxford

Stanislav Kikot
University of Oxford

Piotr Ostropolski-Nalewaja
University of Wrocław

Miguel Romero
Universidad Adolfo Ibáñez

ABSTRACT

A query Q is monotonically determined over a set of views $\mathbf{V}$ if Q can be expressed as a monotonic function of the view image. In the case of relational algebra views and queries, monotonic determinacy coincides with rewritability as a union of conjunctive queries, and it is decidable in important special cases, such as for CQ views and queries [8, 22]. We investigate the situation for views and queries in the recursive query language Datalog. We give both positive and negative results about the ability to decide monotonic determinacy, and also about the co-incidence of monotonic determinacy with Datalog rewritability.

ACM Reference Format:
Michael Benedikt, Stanislav Kikot, Piotr Ostropolski-Nalewaja, and Miguel Romero. 2020. On Monotonic Determinacy and Rewritability For Recursive Queries and Views. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3375395.3387661

1 INTRODUCTION

View definitions allow complex queries to be represented by simple relation symbols. They have many uses, including as a means to protect access to data, as a means to raise the level of abstraction available to data users, and as a means to speed up the evaluation of queries [2]. Views represent a restricted interface to a dataset, and thus an associated question is what class of queries can be answered via accessing this interface. More formally, given a query Q expressed as a logical formula over the base relations, can the answer to Q be obtained via accessing the views. There are several different formulations of this computational problem, depending on what one means by "answering a query accessing the views". One can ask whether Q is expressible as an arbitrary function of the views, or as an arbitrary monotone function of the views. Alternatively, one can choose a particular query language L and ask whether Q can be transformed to a query Q' over the views, where Q' is in L. The first choice is that Q is *determined over the views*, the second that Q is *monotonically determined over the views*, and the last that Q is *L-rewritable over the views.* Each of these notions can be relativized to finite instances.

These questions were studied initially in the case where both queries and views are given by conjunctive queries (CQs). It is known that:

- determinacy of CQ query over a collection of CQ views is equivalent to rewritability of Q over the views in relational algebra [22]
- determinacy of a CQ over CQ views does not agree neither with determinacy over finite instances [16] nor with monotone determinacy [3]
- determinacy of a CQ query over CQ views is undecidable [16], and the same holds for determinacy over finite instances [17]
- determinacy is decidable for queries and views given as *path-CQs* [3]
- monotonic determinacy of a CQ query over CQ views implies rewritability of Q as a CQ [8], agrees with monotonic determinacy over finite instances and is NP-complete to decide [20]

These results have been generalized to the case of queries and views built up with more general constructs of active-domain first-order logic (or equivalently, in relational algebra). Then monotonic determinacy becomes, like determinacy, undecidable, and monotonic determinacy, like determinacy, disagrees with its variant over finite instances. But there is still a relationship between determinacy/monotonic determinacy and rewritability in a logic: determinacy is the same as rewritability in first-order logic; monotonic determinacy is the same as rewritability as a UCQ [8, 22].

Less is known where queries and views are *recursive*, for example, when views and queries are in the common recursive query language Datalog. Since in the non-recursive case of CQ queries and views, monotone determinacy is decidable and co-incides with UCQ rewritability, a natural question is what conditions on Datalog queries and views allow monotone determinacy to be decidable and to co-incide with Datalog rewritability. For specialized recursive queries and views over a graph schema, the *regular path queries*, both the determinacy and monotonic determinacy problem have been studied. For *one- and two-way regular path queries and views* monotonic determinacy (aka "*losslessness with respect to the sound view assumption*") is decidable in ExpSpace ([9] for 1-way,[10] for 2-way), and implies Datalog rewritability [14], while plain determinacy is undecidable [15]. It follows from [13] that monotonic determinacy is undecidable for Datalog queries and CQ views and implies rewritability in Datalog over views.

PODS'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-7108-7/20/06...$15.00
https://doi.org/10.1145/3375395.3387661

The status of these questions for more general recursive queries — e.g., queries and views in Datalog over higher-arity relations — is to the best of our knowledge unknown.

Example 1. *Consider a schema with a ternary relation T, and binary relation B and unary relations U_1, U_2. Consider the Boolean Datalog query Q given as:*

$$\textsc{Goal}_Q \leftarrow\ U_1(x), W_1(x)$$
$$W_1(x) \leftarrow T(x, y, z), B(z, w), B(y, w), W_1(w)$$
$$W_1(x) \leftarrow U_2(x)$$

Consider the following CQ views:

$$V_0(x, w) := T(x, y, z), B(z, w), B(y, w)$$
$$V_1(x) := U_1(x) \qquad V_2(x) := U_2(x)$$
$$V_3(y, z) := U_1(x), T(x, y, z)$$

and the binary Datalog view V_4:

$$\textsc{Goal}_{V_4}(y, z) \leftarrow T(x, y, z), B(z, w),$$
$$B(y, w), T(w, q, r), \textsc{Goal}_{V_4}(q, r)$$
$$\textsc{Goal}_{V_4}(y, z) \leftarrow B(y, w), B(z, w), U_2(w)$$

We can see that Q is monotonically determined over the views V_0-V_2. In fact there is a Datalog rewriting, obtained from Q by first replacing the second rule by $W_1(x) \leftarrow V_0(x, w), W_1(w)$ and then replacing each U_i by V_i in the other rules. Further Q is monotonically determined using views V_3 - V_4, since it can be rewritten as the CQ $\exists y\ z\ V_3(y, z) \wedge V_4(y, z)$.

Note that query Q is not contained in any of the classes considered in past work (e.g. regular path queries).

Our results. We give results on the complexity of deciding monotonic determinacy and on the ability to rewrite monotonically-determined queries into suitable languages, for views and queries expressed in Datalog or in sublanguages such as Monadic Datalog (MDL), or frontier-guarded Datalog (FGDL).

We provide new positive results about rewritability, showing monotonic determinacy implies L-rewritability for some natural query languages L. We show that monotonic determinacy implies rewritability in Datalog for Datalog queries and FGDL views (Theorem 1), as well as for MDL queries and a collection of FGDL and CQ views (Theorem 2). We observe that for CQ Q and Datalog **V**, monotonic determinacy implies rewritability as a CQ, and the same holds if CQ is replaced with UCQ. Note that an analysis of the "inverse rules" algorithm [13] implies that FGDL queries monotonically determined over CQ views have FGDL rewritings. On the negative side, we show that MDL queries monotonically determined over CQ views are not necessarily rewritable in MDL (Theorem 7). This contrasts with the observation from [13] mentioned above. In contrast to Theorem 2, we give an example of an MDL query monotonically determined over UCQ views without a Datalog rewriting (Theorem 8). Our results on rewritability are summarized in Figure 1 where "nn" stands for "not necessarily".

We now turn to results about deciding monotonic determinacy. We show that monotonic determinacy is

- decidable in 2ExpTime for CQ queries and Datalog views (Theorem 5),
- decidable in 2ExpTime for queries and views in frontier-guarded Datalog (Theorem 3),
- decidable in 3ExpTime for MDL queries and a collection of MDL and CQ views (Theorem 4),
- 2ExpTime-hard for CQ queries and MDL views and for MDL queries and CQ views (Proposition 9)
- undecidable for MDL queries and UCQ views (Theorem 6)

Known and new results on decidability of monotonic determinacy are presented in Figure 2 where we use [upper bound]/[lower bound] notation for sources.

Alongside with L-rewritability we can ask whether there are computable functions lying within a certain complexity class which separate the images of instances where Q is true from images of those where Q is false. We call such a function a *separator* for Q over **V**. Note that Datalog rewritings give rise to PTime separators, while UCQ-rewritings produce AC^0 separators. Our additional observations on separators, outside of those that follow from rewritability results, are: (1) for Datalog queries and UCQ views there is always a separator in NP as well as one in co-NP; (2) for any primitive recursive function f there are Datalog queries monotonically determined over Datalog views without a separator in $TIME(f(x))$ (Theorem 9).

Techniques. A contribution of the paper is to show how techniques arising from earlier work can be adapted for the analysis of monotone determinacy. For our positive results, a key tool is an automata-theoretic technique, involving bounds on the treewidth of view images and the *forward-backward method* developed for analysis of guarded logics [7, 18]. For our negative results, we show how to adapt some of the coding ideas used in showing undecidability of determinacy [15–17] to the setting of monotonic determinacy, and we also show how tools from constraint satisfaction [4] can be used to provide monotonically-determined queries that have no Datalog rewriting.

Organization. Section 2 contains preliminaries about Datalog and monotonic determinacy, while Section 3 presents key tools that we make use of in our positive results. Section 4 presents our rewritability results, while Section 5 gives results on deciding monotonic determinacy. Section 6 contains lower bounds on detecting monotonic determinacy, while Section 7 provides non-rewritability results. The paper ends with conclusions and some open questions in Section 8. The details of some proofs are deferred either to the appendix or to the full version.

2 PRELIMINARIES

We will work with relational schemas, consisting of a finite set of relations, with each relation R associated with a number the *arity of* R. For R of arity n, an R-fact is an expression $R(c_1 \dots c_n)$, where $c_1 \dots c_n$ are elements. A fact over schema **S** is an R fact for some relation R of **S**. A *database instance* (or simply *instance* when it is clear that we are discussing data) for a schema is a set of facts over the schema. The *active domain* of an instance $\mathcal{I}$, denoted $\textsc{adom}(\mathcal{I})$, is the set of elements that occur as c_i in some fact $R(c_1 \dots c_n)$ of $\mathcal{I}$. A *query* of arity n over schema **S** is a function from instances of **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 $\mathrm{Output}(Q, \mathcal{I})$. We will also write $\mathcal{I} \models Q(\boldsymbol{c})$ or $\mathcal{I}, \boldsymbol{c} \models Q$ to indicate that $\boldsymbol{c}$ is in

Query \ Views	CQ	MDL, FGDL	FGDL + CQ	UCQ	Datalog
CQ	CQ [Prop. 8, (a)]				
UCQ	UCQ [Prop. 8, (b)]				
MDL	FGDL, nn MDL [13] and [Th. 7]	MDL [Th. 1]	Datalog, nn MDL [Th. 2] and [Th. 7]	not necessarily Datalog [Th. 8]	
FGDL	FGDL [13]	Datalog [Th. 1]	rewritability in Datalog is open		
Datalog	Datalog [13]				

Table 1: Rewritability of Queries Monotonically Determined by the Views

Query \ Views	CQ	MDL, FGDL	FGDL + CQ	UCQ	Datalog
CQ UCQ	NP-c [20]	2ExpTime-c [Th. 5]/[Prop. 9]		Π_2^p-c [21]	2ExpTime-c [Th. 5]/[Prop. 9]
MDL	in 3ExpTime [Th. 4] 2ExpTime-hard [Cor. 9]	2ExpTime-c [Th. 3]/[Prop. 9]	in 3ExpTime [Th. 4] 2ExpTime-hard [Prop. 9]	Undecidable [Th. 6]	
FGDL	decidability is open		decidability is open		
Datalog	undecidable for a fixed atomic view [Prop. 9], see also [13], Th. 3.1				

Table 2: Decidability and Complexity of Monotonic Determinacy

the output of Q on input $\mathcal{I}$. 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 $R(h(c_1) \dots h(c_n)) \in \mathcal{I}'$. If there is a homomorphism from $\mathcal{I}$ to $\mathcal{I}'$ then we write $\mathcal{I} \to \mathcal{I}'$.

The *Gaifman graph* of an instance $\mathcal{I}$ is the graph whose nodes are the elements of $\textsc{adom}(\mathcal{I})$ and whose edges connect any c_i and c_j in a $\boldsymbol{c}$ such that $R(\boldsymbol{c})$ holds. The *radius* of a graph G is defined as $\min_{u \in \textsc{vert}(G)} \max_{v \in \textsc{vert}(G)} dist_G(u, v)$ where $dist_G(u, v)$ is the distance between u and v in G.

Conjunctive queries and Datalog. A *conjunctive query* (CQ) is a logical formula of the form $\mathbf{q}(\boldsymbol{x}) = \exists \boldsymbol{y}\, \phi(\boldsymbol{x}, \boldsymbol{y})$, where $\phi(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms. Given any CQ Q, its *canonical database*, denoted $\textsc{Canondb}(Q)$, is the instance formed by turning each atom $R(x_1 \dots x_n)$ into a fact $R(c_{x_1} \dots c_{x_n})$, where for each variable or constant x in Q we have a constant c_x. Each CQ Q with free variables ordered as $x_1 \dots x_n$ defines a query of arity n in the obvious way: a tuple $t_1 \dots t_n$ is in the output of Q on $\mathcal{I}$ if there is a homomorphism of $\textsc{Canondb}(Q)$ into $\mathcal{I}$ mapping each x_i to t_i. The radius of a CQ is the radius of the Gaifman graph of its canonical database.

Datalog is a language for defining queries over a relational schema $\mathbf{S}$. Datalog rules are of the form:

$$P(\boldsymbol{x}) \leftarrow \phi(\boldsymbol{x})$$

where $P(\boldsymbol{x})$ is an atom over a relation P that is not in $\mathbf{S}$, ϕ is a conjunctive query and every variable in $P(\boldsymbol{x})$ occurs in ϕ. 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 that occur in the head of a rule are the *intensional database predicates* (IDBs). The relations in $\mathbf{S}$ are called the *extensional relations* of the rule. A *Datalog program* is a finite collection of rules. For a database instance $\mathcal{I}$ and a set of Datalog rules Π by $\mathsf{FPEval}(\Pi, \mathcal{I})$ we denote the minimal IDB-extension of $\mathcal{I}$ satisfying Π. A *Datalog query* $Q = (\Pi, \textsc{Goal})$ is a Datalog program Π together with a distinguished intensional *goal relation* $\textsc{Goal}$ 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 $Q(\mathcal{I})$) consists of all tuples $\boldsymbol{c}$ such that $\textsc{Goal}(\boldsymbol{c}) \in \mathsf{FPEval}(\Pi, \mathcal{I})$.

For example, consider a signature where there is a binary relation R and unary relation U. The formula expressing that x has a path consisting of R edges to an element in U would be written in Datalog as the following query $\textsc{Conn}(x) = (\Pi, \textsc{Goal}(x))$ where Π consists of the following rules:

$$\begin{aligned} P(x) &\leftarrow U(x) \\ P(x) &\leftarrow R(x, y), P(x) \\ \textsc{Goal}(x) &\leftarrow P(x) \end{aligned}$$

Above, $P(x)$ and $\textsc{Goal}(x)$ are intensional relations while $R(x, y)$ and $U(x)$ are extensional. We follow conventions concerning Datalog rules and omit the existential quantifiers on the variables in the body that do not appear in the head; we also use “,” for conjunction.

A Datalog query Q_1 is *contained in* a Datalog query Q_2 if $\mathsf{Output}(Q_1, \mathcal{I}) \subseteq \mathsf{Output}(Q_2, \mathcal{I})$ for every instance $\mathcal{I}$. Datalog containment is known to be undecidable in general [23].

Fragments of Datalog. *Monadic Datalog* (MDL) is the fragment of Datalog where all intensional predicates are unary. *Frontier-guarded Datalog* (FGDL) requires that in each rule all the variables in the head co-occur in a single extensional atom of the body. Frontier-guarded Datalog does not contain MDL; for example, in an MDL program we can have a rule $I_1(x) \leftarrow I_2(x)$, where I_1 and I_2 are both intensional. However every MDL program can be rewritten to be in FGDL, and thus we declare, as a convention, that any MDL program is Frontier-guarded. Frontier-Guarded Datalog containment is known to be decidable (e.g. [6]).

Conjunctive queries and approximating Datalog. A Datalog query $Q = (\Pi, \textsc{Goal})$ can be approximated by CQs. We define collections of CQs $\text{CQAppr}(\Pi, U(\boldsymbol{x}), i)$ with free variables $\boldsymbol{x}$ for all atoms $U(\boldsymbol{x})$ that occur in the head of a rule in Π by induction on i. For the base case, $\text{CQAppr}(\Pi, U(\boldsymbol{x}), 1)$ consists of all CQs obtained by taking the body of a rule with the head $U(\boldsymbol{x})$ in Π which contains no intensional predicate.

For the inductive step, $\text{CQAppr}(\Pi, U(\boldsymbol{x}), i+1)$ consists of all CQs obtained by taking any body of a rule whose head is $U(\boldsymbol{x})$ and replacing all intensional atoms $V(\boldsymbol{y})$ with $\mathbf{q}(\sigma(z))$, where $\mathbf{q}(z)$ is in $\text{CQAppr}(\Pi, V(z), k)$ for $k \leq i$ and σ unifies $V(z)$ with $V(\boldsymbol{y})$ by sending $\boldsymbol{z}$ to $\boldsymbol{y}$.

A *CQ approximation* of a Datalog query $(\Pi, \textsc{Goal}(\boldsymbol{x}))$ is any element of $\text{CQAppr}(\Pi, \textsc{Goal}(\boldsymbol{x}), i)$ for some i.

Proposition 1. *For any Datalog query Q, if $\mathcal{I}, \boldsymbol{c} \models Q$ then there is a CQ approximation Q_0 of Q such that $\mathcal{I} \models Q_0(\boldsymbol{c})$.*

We often identify an approximation Q_0 of a Datalog query Q with its canonical database; for example, for another Datalog query Q', we can write $\text{Output}(Q', Q_0)$ to indicate the output of Q' on $\textsc{CanonDB}(Q_0)$. We can also talk about the approximation of an atom A in a Datalog program, which is defined by considering the program with A as the goal predicate. We will exclude approximations that are *redundant*, in that the corresponding multi-set of atoms, including the intensional atoms used in generating the approximation, include a repeated intensional atom. For example, if we have rules $U(x) \leftarrow V(x) \wedge \phi$ and $V(x) \leftarrow U(x) \wedge \phi'$, we should not apply the inductive construction first with $U(x)$ and $V(x)$ in succession.

Views, determinacy, 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 where each view predicate $V \in \Sigma_{\mathbf{V}}$ is interpreted by $\text{Output}(Q_V, \mathcal{I})$. A query Q over schema $\mathbf{S}$ is *determined over* $\mathbf{V}$ if

for any two instances $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathbf{V}(\mathcal{I}_1) = \mathbf{V}(\mathcal{I}_2)$
we have $\text{Output}(Q, \mathcal{I}_1) = \text{Output}(Q, \mathcal{I}_2)$.

A query Q over schema $\mathbf{S}$ is *monotonically determined over* $\mathbf{V}$ if

for any two instances $\mathcal{I}_1, \mathcal{I}_2$ such that $\mathbf{V}(\mathcal{I}_1) \subseteq \mathbf{V}(\mathcal{I}_2)$
we have $\text{Output}(Q, \mathcal{I}_1) \subseteq \text{Output}(Q, \mathcal{I}_2)$.

Given views $\mathbf{V}$ and a query Q, a query R over the view schema $\Sigma_{\mathbf{V}}$ is a *separator* of Q with respect to $\mathbf{V}$ if: for each $\mathcal{I}$ over $\mathbf{S}$, the output of R on $\mathbf{V}(\mathcal{I})$ is the same as the output of Q on $\mathcal{I}$. A separator 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 if this exists we say Q is *L-rewritable* over $\mathbf{V}$.

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 and $L_{\mathbf{V}}$ for the queries and views (e.g. Datalog, fragments of Datalog) are:

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

In this paper, *for simplicity we will always consider the determinacy and rewritability problems restricting to the case when the query Q is Boolean.* But all of our results extend to the non-Boolean case. In addition, we allow our instances to be finite or infinite, but *all of the results extend when the instances are assumed to be finite.*

3 FORWARD AND BACKWARD BETWEEN DATALOG AND AUTOMATA

We overview an automata-theoretic technique that will prove useful in rewriting results. It involves *treewidth bounds*, along with the idea of combining *forward mappings* from Datalog to automata, projection of an automata onto a subvocabulary, and *backward mappings* from an automaton to Datalog. The approach derives from work on guarded logics [7, 18].

Treewidth and tree codes. For a number k a *tree decomposition of width k* for an instance $\mathcal{I}$ is a pair $\mathbb{D} = (\tau, \lambda)$ consisting of a rooted directed tree $\tau = (V, E)$ 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 V such that the following conditions hold:

– for any atom $R(\boldsymbol{c})$ in $\mathcal{I}$, there is a vertex $v \in V$ with $\boldsymbol{c} \subseteq \lambda(v)$;
– for any element c in $\mathcal{I}$, the set $\{v \in V \mid c \in \lambda(v)\}$ is connected in τ.

Above we abuse notation slightly by using $\lambda(v)$ also to refer to the underlying set of elements as well as the tuple. Also in the literature the width associated to such a decomposition is $k - 1$, but this distinction will not be important for any of our results. Will also talk about a tree decomposition of width k for a pair $(\mathcal{I}, \boldsymbol{a})$ consisting of an instance and a tuple. In this case we add to the requirements above that $\boldsymbol{a}$ is an initial segment of $\lambda(r)$ for r the root of the tree.

The *treewidth* of an instance $\mathcal{I}$, $\textsc{tw}(\mathcal{I})$, is the minimum width of a tree decomposition of $\mathcal{I}$. For a tree decomposition $\mathbb{D}$ of data instance $\mathcal{I}$ let $l(\mathbb{D})$ (the "tree span") be the maximum over elements e of $\mathcal{I}$ of the number of bags containing e.

We will now discuss how to represent tree decompositions by labeled trees called codes. In this context, we will always assume that in tree decompositions, *all vertices $v \in V$ have outdegree at most* 2. It is easy to show that if an instance has any tree decomposition of width k, it has one with this property.

We represent such tree decompositions as instances in a signature $\textsc{Code}(\mathbf{S}, k)$ which contains the following relations:

- for every relation $R \in \mathbf{S}$ of arity m and every sequence $\boldsymbol{n} = n_1, \ldots, n_m$ of numbers of size at most k there is a unary relation $T^R_{\boldsymbol{n}}$ in $\textsc{Code}(\mathbf{S}, k)$ to mark the nodes v in τ such that the atom $R(b_{n_1}, \ldots, b_{n_m})$ is in $\mathcal{I}$, where $\lambda(v) = (b_1, \ldots, b_k)$.
- for every partial 1-1 map s from $\{1, \ldots, k\}$ to $\{1, \ldots, k\}$, there is a binary relation T_s to indicate the "same as" relation between positions in neighboring bags. For example, if $(u, v) \in T_s$ and $s(3) = 1$, then the position 3 in u and the position 1 in v stand for the same element. All relations T_s are directed from a parent to a child.

We use $\textsc{UnPred}(\mathrm{S}, k)$ and $\textsc{BinPred}(\mathrm{S}, k)$ to denote the sets of all unary and binary predicates in $\textsc{Code}(\mathrm{S}, k)$ respectively. A tree over this signature will be referred to as a *tree code of width k for* S.

It should be clear how each tree decomposition of $(\mathcal{I}, \boldsymbol{a})$ of width k gives rise to a tree code of width k for S; if there are bags with less than k elements, we fill them up with dummy elements to the length k. We now show how to *decode* an instance from such a code $\mathcal{T}$. For nodes u, v in a code $\mathcal{T}$, we write $(u, i) \equiv_0 (v, j)$ if $(u, v) \in T_s$ holds in $\mathcal{T}$ and $s(i) = j$. For a node u and position i we let $[u, i]$ be the equivalence class of (u, i) in the equivalence relation generated by $\equiv_0$. In words, the position i in the node u corresponds to the position j in the node v if there is an undirected path leading from u to v with the edge labels that in a step-by-step manner establish a match between i in u and j in v. The *decoding* of $\mathcal{T}$, denoted $\mathcal{I} = \mathfrak{D}(\mathcal{T})$, is the S database instance $\mathcal{I}$ consisting of atoms $R([v_1, i_1], \ldots, [v_r, i_r])$ where each R from S is applied to exactly those tuples $([v_1, i_1], \ldots, [v_r, i_r])$ for which there is some node $w \in \mathrm{dom}(\mathcal{T})$ such that $w \in T^R_{j_1 \ldots j_r}$ and $[w, j_m] = [v_m, i_m]$ for all $m \in \{1, \ldots, r\}$. In this case we also say the $\mathcal{T}$ *is a code of* $\mathcal{I}$.

Specialized tree decompositions for Monadic Datalog. As observed in [11], for Monadic Datalog the CQ approximations have tree decompositions with small treespan.

Lemma 1. *Let Q be a Monadic Datalog query. Then there is a number $k = O(|Q|)$ such that all CQ-approximations of Q have a tree decomposition $\mathbb{D}$ of width k with $l(\mathbb{D}) \leq \mid Q \mid$.*

Bounding the treewidth of view images. We present results showing that, for certain classes of sets of views $\mathbf{V}$ and Datalog queries Q, we can find a uniform bound on the treewidth of the $\mathbf{V}$-image of the approximations of Q.

It is easy to see that expanding an instance with the evaluation of all intensional predicates of a frontier-guarded program does not blow-up treewidth:

Lemma 2. *If $\Pi \in FGDL$ and $\mathcal{I}$ is an instance of treewidth k, then* $\mathsf{FPEval}(\Pi, \mathcal{I})$ *is of treewidth k.*

A locality argument shows that applying connected CQ views preserves bounded treewidth, assuming that we deal with tree decompositions with low tree span:

Lemma 3. *Let $\mathbb{D}$ be a tree decomposition of a data instance $\mathcal{I}$ of width k with $l(\mathbb{D}) \leq s$. Let $\mathbf{V}$ be a set of connected CQ views, and $\mathbf{V}(\mathcal{I})$ the view image of $\mathcal{I}$ under $\mathbf{V}$. Let r be the greatest radius of a CQ in $\mathbf{V}$. Then the treewidth of $\mathbf{V}(\mathcal{I})$ is at most $k' = \frac{k\cdot((s\cdot k)^{r+1}-1)}{(s\cdot k)-1}$.*

Proof. For a bag b of $\mathbb{D}$ and an integer n define recursively its *n-extension* by setting $ext(b, 0) = b$ and $ext(b, n) = \{u \mid \exists v \in ext(b, n-1)$ such that u and v belong to a same bag of $\mathbb{D}\}$. Since $l(\mathbb{D}) \leq s$, it is easy to see by induction that $|ext(b, n)| \leq k + k \cdot (s \cdot k) \cdots + k \cdot (s \cdot k)^{n+1} = \frac{k((s\cdot k)^{n+1}-1)}{((s\cdot k)-1}$. Let $\mathbb{D}'$ be a tree of bags whose set of nodes is $ext(b, r)$, with an edge between $ext(b, r)$ and $ext(b', r)$ exactly when there is an edge from b to b' in T. We claim that $\mathbb{D}'$ is a tree decomposition of $\mathbf{V}(\mathcal{I})$.

First, it is easy to see that for any element v the set of all bags in $\mathbb{D}'$ containing v is connected.

Secondly we show that for each atom $S(\boldsymbol{c})$ from $\mathbf{V}(\mathcal{I})$ there is a node in $\mathbb{D}'$ containing $\boldsymbol{c}$. Suppose that $S(\boldsymbol{c})$ was generated by the view definition $S(\boldsymbol{x}) \leftarrow \phi(\boldsymbol{x}, \boldsymbol{y})$ for a connected CQ $\phi(\boldsymbol{x}, \boldsymbol{y})$ with free variables $\boldsymbol{x}$ and quantified variables $\boldsymbol{y}$ under some assignment η defined on both $\boldsymbol{x}$ and $\boldsymbol{y}$. As ϕ is of radius at most r, it should have a variable $z \in \boldsymbol{x} \cup \boldsymbol{y}$ such that all other variables are at distance at most r from z in the Gaifman graph of $\phi(\boldsymbol{x}, \boldsymbol{y})$. Therefore the range of η lies within distance r from $\eta(z)$. Let b be any bag of $\mathbb{D}$ containing $\eta(z)$. If follows that $\boldsymbol{c}$ is contained in the r-extension of b.

Finally, it is easy to see that the sizes of bags of $\mathbb{D}'$ are as required. □

Tree automata. We describe our variant of tree automata that accept binary trees $\mathcal{T}$ with edges labelled by binary relations from the set $\textsc{BinPred}(\mathrm{S}, k)$ and nodes labelled with unary predicates from a set $\textsc{UnPred}(\mathrm{S}, k)$. We consolidate node- and edge-labels by considering a tree alphabet $\mathcal{A}$. $\mathcal{A}$ contains labels $\sigma_L^{s_1,s_2}$ indexed by *sets* of unary predicates $L \subseteq \textsc{UnPred}(\mathrm{S}, k)$ and pairs of binary predicates $s_1, s_2 \in \textsc{BinPred}(\mathrm{S}, k)$. It also contains leaf labels σ_L indexed by $L \subseteq \textsc{UnPred}(\mathrm{S}, k)$. We sometimes treat trees as terms over this alphabet: a tree with root labeled $\sigma_L^{s_1,s_2}$ with children t_1 and t_2 would be written as $\sigma_L^{s_1,s_2}(t_1, t_2)$.

A *nondeterministic finite tree automaton* (NTA) *over* $\mathcal{A}$ is a tuple $\mathfrak{A} = (Q, Q_f, \Delta_0, \Delta_2)$, where

- Q is a finite set of *states*
- $Q_f \subseteq Q$ is a set of *final* states,
- Δ_0 is a set of *initial transitions* of the form $\sigma_L \to q$, and
- Δ_2 is a set of *transitions* of the form $q_1, q_2, \sigma_L^{s_1,s_2} \to q$.

A run of $\mathfrak{A}$ on a tree $\mathcal{T}$ is a label function $f : \textsc{Nodes}(\mathcal{T}) \to Q$ satisfying the following: if $t_v = \sigma_L^{s_1,s_2}(t_{v_1}, t_{v_2})$ for $\sigma_L^{s_1,s_2} \in \mathcal{A}$, then $(f(v_1), f(v_2), \sigma_L^{s_1,s_2} \to f(t)) \in \Delta_2$ **and** if $f(v) = q$ for a leaf v of $\mathcal{T}$ with $t_v = \sigma_L$, then $\sigma_L \to q \in \Delta_0$. We say that $\mathcal{T}$ is *accepted* by $\mathfrak{A}$ if there is a run of $\mathfrak{A}$ on $\mathcal{T}$ that labels the root of $\mathcal{T}$ with a final state.

Forward from Datalog to NTA. We now show how to create a tree automaton accepting the view images of approximations of a given Datalog query. We say that a class $\mathbb{C}$ of instances is *k-regular* if the treewidth of instances in $\mathbb{C}$ is at most k, and there is an automaton $\mathfrak{A}$ such that

- for codes $\mathcal{T}$ of width k, $\mathfrak{A}$ accepts $\mathcal{T}$ implies $\mathfrak{D}(\mathcal{T}) \in \mathbb{C}$.
- for each instance $\mathcal{F} \in \mathbb{C}$ there is a code $\mathcal{T}$ such that $\mathfrak{D}(\mathcal{T}) = \mathcal{F}$ and $\mathfrak{A}$ accepts $\mathcal{T}$.

In this case we say that $\mathfrak{A}$ *captures* $\mathbb{C}$. If the stronger condition "for all codes $\mathcal{T}$, $\mathfrak{A}$ accepts $\mathcal{T}$ iff $\mathfrak{D}(\mathcal{T}) \in \mathbb{C}$" holds, we say that $\mathfrak{A}$ *recognizes* $\mathbb{C}$.

The following simple "forward mapping" proposition shows that we can capture the approximations of Datalog queries with an automaton:

Proposition 2. *For any Datalog query $Q = (\Pi, \textsc{Goal})$, there is an* ExpTime *function that outputs an NTA $\mathfrak{A}_Q$ that captures the set of canonical databases of CQ approximations of Q.*

If we restrict to instances of a fixed treewidth, we can do better, obtaining an NTA that recognizes all trees that satisfy the Datalog program considered as a set of Horn clauses:

Proposition 3. *For any Datalog program Π, the class $\{\mathcal{F} \mid \mathcal{F} \models \Pi, \textsc{tw}(\mathcal{F}) \leq k\}$ (here $\mathcal{F}$ are finite instances which contain both EDBs* *and IDBs of* Π*) is* k*-regular and is recognized by an NTA at most doubly-exponential sized in* k *and singly-exponential in* $|\Pi|$.

We also note that if we have captured a class of codes of instances with an automaton, we can project away some of the signature and still capture:

Proposition 4. *If* $\mathbb{C}$ *is a* k*-regular class in* Σ *captured by NTA* $\mathfrak{A}$ *and* $\Sigma' \subseteq \Sigma$*, then the class*

$$\mathbb{C} \upharpoonright \Sigma' = \{\mathcal{F} \upharpoonright \Sigma' \mid \mathcal{F} \in \mathbb{C}\}$$

is also k*-regular, captured by an automaton of size at most* $|\mathfrak{A}|$*. The same holds with "captured" replaced by "recognized".*

Our next "forward mapping" result shows that we can recognize the set of codes of small treewidth which fail to satisfy clauses of a frontier-guarded program:

Proposition 5. *For a FGDL query* $Q = (\Pi, \textsc{Goal})$ *the set* $\{(\mathcal{I}, \boldsymbol{a}) \mid \mathcal{I} \not\models Q(\boldsymbol{a}), \textsc{tw}(\mathcal{I}) \leq k\}$ *is* k*-regular and recognized by an NTA of size at most doubly-exponential in* k.

Proof. Follows from Propositions 3 and 4 since for a frontier-guarded $Q = (\Pi, \textsc{Goal})$ we have, using Lemma 2,

$$\begin{aligned}\{(\mathcal{I}, \boldsymbol{a}) \mid \mathcal{I} \not\models Q(\boldsymbol{a}),\ \textsc{tw}(\mathcal{I}) \leq k\} &= \\ &= \{(\mathcal{F} \upharpoonright \Sigma, \boldsymbol{a}) \mid \mathcal{F} \models \Pi,\ \mathcal{F} \not\models \textsc{Goal}(\boldsymbol{a}),\ \textsc{tw}(\mathcal{F}) \leq k\}\end{aligned}$$

where Σ is the signature of the EBDs in Π. □

In applying these results, we will sometimes use implicitly that *if* $\mathbb{C}_1$ *is captured by* $\mathfrak{A}$ *and* $\mathbb{C}_2$ *is recognized by* $\mathfrak{A}'$*, then* $\mathbb{C}_1 \cap \mathbb{C}_2$ *is captured by the product of* $\mathfrak{A}$ *and* $\mathfrak{A}'$. Note that, in contrast, classes of instances that are captured are *not* closed under intersection.

Homomorphic determinacy. A query Q is said to be *homomorphically determined* by views $\mathbf{V}$ if:

> Whenever we have two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ and a homomorphism h from $\mathbf{V}(\mathcal{I}_1)$ to $\mathbf{V}(\mathcal{I}_2)$, then for each tuple $(c_1, \ldots, c_k) \in Q(\mathcal{I}_1)$ we also have $(h(c_1), \ldots, h(c_k)) \in Q(\mathcal{I}_2)$.

Note that if Q is rewritable over $\mathbf{V}$ in Datalog, or any other homomorphism-invariant query language, then Q must be homomorphically determined by $\mathbf{V}$.

Homomorphic determinacy of Q over $\mathbf{V}$ always implies monotonic determinacy of Q over $\mathbf{V}$; monotonic determinacy is simply the case where h is the identity. Surprisingly, for Datalog queries and views the converse also holds:

Lemma 4. *For any Datalog query* Q *and Datalog views* $\mathbf{V}$*, if* Q *is monotonically determined over* $\mathbf{V}$ *then it is homomorphically determined over* $\mathbf{V}$.

Backwards from NTAs to Datalog. Consider arbitrary NTA $\mathfrak{A}$ that works on tree codes of width k. From $\mathfrak{A}$ we construct a Datalog program. For every transition of the form $q_1, q_2, \sigma_L^{s_1,s_2} \rightarrow q$ with $L = \{T_{\boldsymbol{n}_1}^{R^1}, \ldots, T_{\boldsymbol{n}_m}^{R^m}\}$ we create a rule

$$\begin{aligned}&P_q(x_1, \ldots, x_k) \leftarrow \\ &\qquad \bigwedge_{i=1}^{k} Adom(x_i) \wedge P_{q_1}(x_1^1, \ldots, x_k^1) \wedge P_{q_2}(x_1^2, \ldots, x_k^2) \\ &\qquad \wedge \bigwedge_{i \in \mathrm{dom}(s_1)} x_i = x_{s_1(i)}^1 \wedge \bigwedge_{i \in \mathrm{dom}(s_2)} x_i = x_{s_2(i)}^2 \wedge \bigwedge_{l=1}^{m} R^l(\boldsymbol{x}_{\boldsymbol{n}_l})\end{aligned} \tag{1}$$

where j ranges over 1 and 2, x_i^j are fresh variables for indices $j \in \{1, 2\}$ and $i \in \{1, \ldots, k\}$, and for $\boldsymbol{n}_l = (n_l^1, \ldots, n_l^d)$ we have $\boldsymbol{x}_{\boldsymbol{n}_l} = (x_{n_l^1}, \ldots, x_{n_l^d})$. For initial transitions of the form $\sigma_L \rightarrow q$ with $L = \{T_{\boldsymbol{n}_1}^{R^1}, \ldots, T_{\boldsymbol{n}_m}^{R^m}\}$ we have rules

$$P_q(x_1, \ldots, x_k) \leftarrow \bigwedge_{i=1}^{k} Adom(x_i) \wedge \bigwedge_{l=1}^{m} R^l(\boldsymbol{x}_{\boldsymbol{n}_l}). \tag{2}$$

For each accepting state q we add the rules $\textsc{Goal}_{\mathfrak{A}} \leftarrow P_q(x_1, \ldots, x_k)$ defining the goal predicate $\textsc{Goal}_{\mathfrak{A}}$; recall that we are assuming here that the original query Q is Boolean, so we are looking for a Boolean Datalog rewriting. We also add a standard set of rules which, when evaluated on any data instance $\mathcal{I}$ under fixed-point semantics, guarantee that the interpretation of the IDB $Adom(x)$ is the active domain of $\mathcal{I}$. Denote the resulting *backward map Datalog query* by $Q_{\mathfrak{A}}$.

We now get to the main result of this section, which states that if we assume homomorphic determinacy and begin with an automaton representing view images of approximations of Q, then applying the backward mapping produces a Datalog rewriting of Q over $\mathbf{V}$.

Proposition 6. *Let* Q *be homomorphically determined over* $\mathbf{V}$ *and* $\mathfrak{A}$ *be any automaton working on* k*-codes such that* $\{\mathbf{V}(Q_i) \mid i \in \omega\} \subseteq \mathfrak{D}(L(\mathfrak{A})) \subseteq \{\mathcal{J} \mid \mathbf{V}(Q_i) \text{ maps into } \mathcal{J} \text{ for some } i \in \omega\}$*. That is to say, we require that*

1. *for each CQ approximation* Q_i *of* Q *there is a code* $\mathcal{T}$ *such that* $\mathfrak{D}(\mathcal{T}) = \mathbf{V}(Q_i)$ *and* $\mathcal{T}$ *is accepted by* $\mathfrak{A}$*;*
2. *for each* $\mathcal{T}$ *accepted by* $\mathfrak{A}$ *there is a CQ approximation* Q_i *of* Q *and a homomorphism from* $\mathbf{V}(Q_i)$ *into* $\mathfrak{D}(\mathcal{T})$.

Then for each data instance $\mathcal{I}$ *we have* $\mathcal{I} \models Q$ *iff* $\mathbf{V}(\mathcal{I}) \models Q_{\mathfrak{A}}(\boldsymbol{a})$ *for some* $\boldsymbol{a} \in \textsc{adom}(\mathcal{I})^k$.

To prove the proposition, we need the notion of a "proof certificate" for backward mappings of an automaton.

When a Datalog query $Q = (\Pi, \textsc{Goal})$ holds for a tuple $\boldsymbol{d}$ in an instance $\mathcal{I}$, there is a derivation that witnesses this, which has a tree-like structure. A *proof term* for $\mathcal{I} \models Q(\boldsymbol{d})$ is a labelled finite tree in which every node v is labelled with a ground fact $\textsc{FactOf}(v)$ over the predicates mentioned in Π, and every non-leaf node v is additionally labelled with a rule $\textsc{RuleOf}(v)$ of Π such that:

- If v is the root, $\textsc{FactOf}(v) = \textsc{Goal}(\boldsymbol{d})$
- If v is a leaf then $\textsc{FactOf}(v)$ is a fact over the extensional predicates of Q, and this fact holds in $\mathcal{I}$
- If v is not a leaf, let $\mathcal{I}_v$ be the instance consisting of $\textsc{FactOf}(v)$ and all facts $\textsc{FactOf}(c)$ for c a child of v. Then there is a map h_v from the variables in the body of $\textsc{RuleOf}(v)$ into the active domain of $\mathcal{I}_v$ that maps the facts in the body

of $\textsc{RuleOf}(v)$ onto the facts of $\mathcal{I}_v$, and maps the head of $\textsc{RuleOf}(v)$ to $\textsc{FactOf}(v)$.

It is well-known [1] and easy to see that proof terms represent a semantics for Datalog: $\mathcal{I} \models Q(\boldsymbol{d})$ exactly when there is a proof term that witnesses this.

We now give a notion of a witness for acceptance of an automaton running over codes. A *jointly-annotated term* for automaton $\mathfrak{A}$, instance $\mathcal{I}$, and k-tuple $\boldsymbol{a}$ is a pair $(\mathcal{T}, \boldsymbol{b})$ where

- $\mathcal{T}$ is a tree code accepted by $\mathfrak{A}$;
- the map $\boldsymbol{b}$ assigns each vertex of $\mathcal{T}$ to a k-tuple of elements from $\mathcal{I}$, with the root of $\mathcal{T}$ mapped to $\boldsymbol{a}$;

which satisfy the following condition: **if** $t_v = \sigma_L^{s_1,s_2}(t_{v_1}, t_{v_2})$ with $L = \{T^{R^1}_{\boldsymbol{n}_1}, \dots, T^{R^m}_{\boldsymbol{n}_m}\}$, $\boldsymbol{b}(v) = (b_1, \dots, b_k)$, $\boldsymbol{b}(v_j) = (b^j_1, \dots, b^j_k)$ for $j = 1, 2$ **then**

$$\mathcal{I} \models \bigwedge_{j=1}^{2} \quad \bigwedge_{i \in \mathrm{dom}(s_j)} b_i = b^j_{s_j(i)} \wedge \bigwedge_{l=1}^{m} R^l(\boldsymbol{b}_{\boldsymbol{n}_l}). \tag{3}$$

We also require that

$$\begin{array}{l} \textbf{if } t = \sigma_L \text{ is a leaf symbol in } T \text{ with } L = \{R^1_{\boldsymbol{n}_1}, \dots, R^m_{\boldsymbol{n}_m}\} \\ \quad \textbf{then } \text{ the atoms } R^l(\boldsymbol{b}_{\boldsymbol{n}_l}) \text{ are in } \mathcal{I} \text{ for } l = 1, \dots, m \end{array} \tag{4}$$

In other words, $\boldsymbol{b}$ can be considered as a homomorphism from $\mathfrak{D}(T)$ into $\mathcal{I}$.

We now verify the key property of a jointly-annotated term:

Proposition 7. *For each data instance $\mathcal{I}$ it follows that $\mathcal{I}, \Pi_{\mathfrak{A}} \models$ $\textsc{Goal}_{\mathfrak{A}}(\boldsymbol{a})$ if and only if there is a jointly-annotated term for $\mathfrak{A}$, $\mathcal{I}$ and $\boldsymbol{a}$.*

Proof. We prove the two directions of the if and only if separately.

($\Rightarrow$) Take a proof term t that witnesses $\mathcal{I}, \Pi_{\mathfrak{A}} \models \textsc{Goal}_{\mathfrak{A}}(\boldsymbol{a})$. We transform t into a jointly-annotated term $(\mathcal{T}, \boldsymbol{b})$ on the set of all vertices of t with $\textsc{FactOf}(v)$ being an IDB. Note that this gives us a binary tree since all rule bodies in $\Pi_{\mathfrak{A}}$ have either 0 or 2 IDBs by assumption. For each vertex v we take some ordering $(u^v_1, \dots u^v_k)$ of elements in $\mathcal{I}_v$ without duplicates; we use fresh dummy elements to fill up the tuple if $\mathcal{I}_v$ has less then k elements. Now we define unary labels of $\mathcal{T}$ by setting $T^R_{n_1,\dots,n_m}(v) \in \mathcal{T}$ iff $R(u^v_{n_1}, \dots, u^v_{n_m}) \in \mathcal{I}_v$. We define edge labels s between a parent v and its child w by setting $s(n) = m$ if u^v_n is the same element as u^w_m; it should be clear that s is a partial bijection. This constitutes the definition of $\mathcal{T}$. It remains to define $\boldsymbol{b}$ by setting $\boldsymbol{b}(v)$ to be $(u^v_1, \dots u^v_k)$. We can create an accepting run f by setting $f(v)$ to be the state of the automaton q such that $\textsc{FactOf}(v)$ is labelled by and IDB P_q.

($\Leftarrow$) It is easy to show by induction that if v is a vertex of a jointly annotated-term $(\mathcal{T}, \boldsymbol{b})$ for $\mathfrak{A}$, $\mathcal{I}$, and $\boldsymbol{a}$ and f is an accepting run for $\mathfrak{A}$ on $\mathcal{T}$ with $f(v) = q$, then $\mathcal{I}, \Pi_{\mathfrak{A}} \models P_q(\boldsymbol{b}(v))$. It follows that $\mathcal{I}, \Pi_{\mathfrak{A}} \models \textsc{Goal}_{\mathfrak{A}}(\boldsymbol{a})$.

Indeed, if v is a leaf, then then $\mathcal{I}, \Pi_{\mathfrak{A}} \models P_q(\boldsymbol{b}(v))$ by the rule (2) because its body holds due to condition (4) and the fact that for all u in $\boldsymbol{b}(v)$ we have $\mathcal{I}, \Pi_{\mathfrak{A}} \models \mathit{Adom}(u)$.

If v has children v_1 and v_2, then there must be q_1 and q_2 such that $f(v_1) = q_1$, $f(v_2) = q_2$, production $(q_1, q_2, \sigma_L^{s_1,s_2} \to q)$ is a transition of $\mathfrak{A}$, and the vertex label of v is L while edge labels between v, v_1 and v_2 are s_1 and s_2.

We claim that $\mathcal{I}, \Pi_{\mathfrak{A}} \models P_q(\boldsymbol{b}(v))$ can be inferred by the rule (1) for this production under assignment $\{(x_1, \dots, x_k) := \boldsymbol{b}(v), (x^1_1, \dots, x^1_k) := \boldsymbol{b}(v_1), (x^2_1, \dots, x^2_k) := \boldsymbol{b}(v_2)\}$. Indeed, we have $\mathcal{I}, \Pi_{\mathfrak{A}} \models \mathit{Adom}(u)$ for all elements in the body of the rule, we have $\mathcal{I}, \Pi_{\mathfrak{A}} \models P_{q_1}(\boldsymbol{b}(v_1))$ and $\mathcal{I}, \Pi_{\mathfrak{A}} \models P_{q_2}(\boldsymbol{b}(v_2))$ by the induction hypothesis, and the rest of the rule by (3).

□

We are now ready for the proof of Proposition 6:

Proof. Suppose Π is a Datalog program containing intensional predicate A, $\mathcal{I}$ is an instance for the extensional (input) signature of Π, and $\boldsymbol{a}$ is a tuple of elements from $\mathcal{I}$. Below we write

$$\mathcal{I}, \Pi \models A(\boldsymbol{a})$$

to indicate that the least fixpoint of Π on $\mathcal{I}$ contains $A(\boldsymbol{a})$.

($\Rightarrow$) Suppose that $\mathcal{I} \models Q$. Then there is an approximation Q_i of Q and a homomorphism h from Q_i into $\mathcal{I}$, which is also a homomorphism from $\mathbf{V}(Q_i)$ to $\mathbf{V}(\mathcal{I})$. As Q_i is an approximation of Q, by the first inclusion for $L(\mathfrak{A})$, $\mathfrak{A}$ must accept some code $\mathcal{T}$ of $\mathbf{V}(Q_i)$. Choose an arbitrary element e_0 from $\textsc{adom}(\mathbf{V}(\mathcal{I}))$. For a vertex v of $\mathcal{T}$ we define $\boldsymbol{b}(v)$ to be the tuple $(e_1, \dots, e_k)$ of elements of $\mathbf{V}(\mathcal{I})$ where each e_i is defined as follows:

$$e_i = \begin{cases} h([v,i]), & \text{if } [v,i] \in \textsc{adom}(\mathcal{I}) \\ e_0, & \text{otherwise.} \end{cases}$$

We claim that $(\mathcal{T}, \boldsymbol{b})$ is a jointly-annotated term for $\mathfrak{A}$, $\mathbf{V}(\mathcal{I})$ and the $\boldsymbol{b}$-image of the root of $\mathcal{T}$. Indeed, if equation (3) contains an equality $[v,i] = [u,j]$, it follows that $[v,i]$ and $[u,j]$ are indeed equivalent. The R-atoms of equations (3) and (4) hold because h is a homomorphism, and also because they are never applied to dummies. It follows (by Proposition 7) that $\mathbf{V}(\mathcal{I}) \models Q_{\mathfrak{A}}(\boldsymbol{a})$ for some $\boldsymbol{a}$.

($\Leftarrow$) Suppose that $\mathbf{V}(\mathcal{I}), \Pi_{\mathfrak{A}} \models \textsc{Goal}_{\mathfrak{A}}(\boldsymbol{a})$. Let $(\mathcal{T}, \boldsymbol{b})$ be a jointly-annotated term for the inference of $\textsc{Goal}_{\mathfrak{A}}(\boldsymbol{a})$ for $\mathbf{V}(\mathcal{I}), \Pi_{\mathfrak{A}}$ and $\boldsymbol{a}$ (which exists by Proposition 7), and f be an accepting run of $\mathfrak{A}$ on $\mathcal{T}$. Thus, by the second inclusion for $L(\mathfrak{A})$, there must be a homomorphism h from $\mathbf{V}(Q_i)$ for some i into $\mathfrak{D}(\mathcal{T})$. Note that by Proposition 7, we know that $\boldsymbol{b}$ can be considered as a homomorphism from $\mathfrak{D}(\mathcal{T})$ into $\mathbf{V}(\mathcal{I})$. By composing h with $\boldsymbol{b}$, we obtain a homomorphism g from $\mathbf{V}(Q_i)$ into $\mathbf{V}(\mathcal{I})$. Now we have a data instance $\mathcal{I}' = Q_i$ such that $\mathcal{I}' \models Q$ and a homomorphism g from $\mathbf{V}(\mathcal{I}')$ into $\mathbf{V}(\mathcal{I})$. As Q is homomorphically determined by $\mathbf{V}$, we have $\mathcal{I} \models Q$. □

4 REWRITABILITY

We are now ready to present our main results about rewritings of queries that are monotonically determined over views. The following result exhibits how the forward and backward mappings help us obtain Datalog rewritings.

Theorem 1. *Suppose Q is a Datalog query and $\mathbf{V}$ is a collection of FGDL views. If Q is monotonically determined by $\mathbf{V}$, then Q is rewritable over $\mathbf{V}$ in Datalog. The size of the rewriting is at most double-exponential in $|Q|$ and exponential in $|\mathbf{V}|$. If Q is MDL such a rewriting exists in MDL as well.*

Proof. Consider the class $\mathbb{C}$ of canonical databases of CQ approximations of Q. By Proposition 2, $\mathbb{C}$ is k-regular for some $k = O(|Q|)$ and is captured by an NTA $\mathfrak{A}'$ of at most exponential size in $|Q|$. By Lemma 2, the treewidth of the class of view images of $\mathbb{C}$ is also bounded by k. We claim that there is an automaton $\mathfrak{A}$ that captures $\mathbb{V} = \{\mathcal{J} \mid \text{TW}(\mathcal{J}) \leq k, \mathbf{V}(Q_i) \subseteq \mathcal{J} \text{ for some } i \in \omega\}$ of size at most double-exponential in $|Q|$ and single-exponential in $|\mathbf{V}|$ ("of required size" below) and argue that it satisfies the conditions of Proposition 6.

Without loss of generality we assume that the sets of IDBs of programs for different views are disjoint, and that their goal predicates are identical with the view predicates. Denote by $\Pi_\mathbf{V}$ the union of all rules in Datalog queries in $\mathbf{V}$. Note that by definition, for any instance $\mathcal{I}$, the restriction of $\mathsf{FPEval}(\Pi_\mathbf{V}, \mathcal{I})$ on the view signature is exactly $\mathbf{V}(\mathcal{I})$.

By Proposition 3, there is an NTA $\mathfrak{A}^{\Pi_\mathbf{V}}$ of required size which recognizes all codes of $\{\mathcal{F} \mid \mathcal{F} \models \Pi_\mathbf{V}, \text{TW}(\mathcal{F}) \leq k\}$. Hence the class $\mathbb{F} = \{\mathcal{F} \mid \mathcal{F} \restriction \Sigma_\text{B} \in \mathbb{C}, \mathcal{F} \models \Pi_\mathbf{V}, \text{TW}(\mathcal{F}) \leq k\}$ is captured by the intersection of $\mathfrak{A}'$ and $\mathfrak{A}^{\Pi_\mathbf{V}}$, which is also of required size. Observe that $\mathbb{V}$ is the projection of $\mathbb{F}$ on the signature of view predicates and so $\mathbb{V}$ is captured by some NTA $\mathfrak{A}$ of required size by Proposition 4.

By Lemma 4, Q is homomorphically determined over $\mathbf{V}$. Since $\{\mathbf{V}(Q_i) \mid Q_i \text{ is a CQ approximation of } Q\}$ is the projection of

$$\{\mathsf{FPEval}(\Pi_\mathbf{V}, \mathcal{I}) \mid \mathcal{I} \in \mathbb{C}\}$$

on the signature of view predicates, we have:

$$\begin{aligned}\{\mathsf{FPEval}(\Pi_\mathbf{V}, \mathcal{I}) \mid \mathcal{I} \in \mathbb{C}\} \subseteq \\ \{\mathcal{F} \mid \mathcal{F} \restriction \Sigma \in \mathbb{C}, \mathcal{F} \models \Pi_\mathbf{V}, \text{TW}(\mathcal{F}) \leq k\} \subseteq \\ \{\mathcal{F} \mid \mathcal{F} \restriction \Sigma \in \mathbb{C}, \mathcal{F} \models \Pi_\mathbf{V}\},\end{aligned}$$

The inclusions above are preserved when projecting to the signature of view predicates. From this we can verify that the condition of Proposition 6 holds for $\mathfrak{A}$. Now applying Proposition 6, we conclude that Q is Datalog rewritable, and that the rewriting is of required size.

If Q is MDL the construction can be refined to produce an MDL rewriting; see the full version for details. □

We can use the same technique in the setting where the views are combinations of Monadic Datalog and CQs, while the query is Monadic Datalog, using Lemma 1 and the bound of Lemma 3, which relies on the bound on l produced in Lemma 1. Although Lemma 3 requires connectivity, we can show that disconnected views can be replaced by connected ones.

Theorem 2. *Suppose Q is a Monadic Datalog query and $\mathbf{V}$ is a collection of Monadic Datalog and CQ views. If Q is monotonically determined by $\mathbf{V}$, then Q is rewritable over $\mathbf{V}$ in Datalog. The size of the rewriting is at most double-exponential in $K = O(|Q|^{|\mathbf{V}|})$.*

The previous rewriting results involved restricting the views. We now note that if we restrict the query to be a UCQ, monotonic determinacy implies not only Datalog rewritability, but even UCQ rewritability, for arbitrary Datalog views:

Proposition 8. *For views $\mathbf{V}$ in arbitrary Datalog we have:*
(1) if a CQ Q is monotonically determined by $\mathbf{V}$, then there is a CQ-rewriting of Q in terms of $\mathbf{V}$;
(2) if a UCQ Q is monotonically determined by $\mathbf{V}$, then there is a UCQ-rewriting of Q in terms of $\mathbf{V}$.
In both cases the rewritings are polynomial size in $|Q|$ and $|\mathbf{V}|$.

Proof. This can be seen as a "degenerate" variant of the forward-backward technique, which is well-known in the DB and KR literature [3, 21, 22]. Let Q be the disjunction of $Q_i : i \in S$. Let $Q' = \bigvee_{i \in S} \mathbf{V}(Q_i)$ denote the query that holds on an instance $\mathcal{I}'$ of the view schema exactly when for some $i \in S$, there is a homomorphism of $\mathbf{V}(Q_i)$ into $\mathcal{I}'$. Equivalently, this is the query obtained by applying the views to each canonical database of a disjunct of Q, and then interpreting the resulting facts as a query.

We claim that if Q is monotonically determined by $\mathbf{V}$, then Q' is a rewriting of Q. In particular, if Q is a CQ, then Q' is just a CQ. We need to show that for each instance $\mathcal{I}$, $\mathcal{I} \models Q$ iff $\mathbf{V}(\mathcal{I}) \models Q'$.

($\Rightarrow$) If some Q_k maps into $\mathcal{I}$, then $\mathbf{V}(Q_k)$ maps into $\mathbf{V}(\mathcal{I})$.

($\Leftarrow$) Suppose some $\mathbf{V}(Q_k)$ maps into $\mathbf{V}(\mathcal{I})$. Monotonic determinacy implies homomorphic determinacy by Lemma 4 In the definition of homomorphic determinacy, take $\mathcal{I}_1 = Q_k$ and $\mathcal{I}_2 = \mathcal{I}$. It is easy to check that $\mathbf{V}(\mathcal{I}_1)$ maps into $\mathbf{V}(\mathcal{I}_2)$ and $\mathcal{I}_1 \models Q$. It follows that $\mathcal{I}_2 \models Q$, in other words, that $\mathcal{I} \models Q$.

□

5 DECIDABILITY

We move from rewritability results to decision procedures for monotonic determinacy.

Monotonic determinacy testing procedure. Our decidability results will depend upon an characterization of monotonic determinacy, which we review here. Given a Datalog query Q and Datalog views $\mathbf{V}$, a *canonical test for Monotonic Determinacy* is a tuple (Q_i, D') that consists of:

- A CQ Q_i that is a CQ-approximation of Q
- An instance D' of the input schema formed by taking each fact $F = V(\boldsymbol{c})$ in $\mathbf{V}(Q_i)$, choosing a CQ approximation Q' of Q_V, and replacing F with fresh elements and facts from Q' that witness $V(\boldsymbol{c})$. That is, firing the rule $\forall \boldsymbol{x}\ V(\boldsymbol{x}) \rightarrow Q'(\boldsymbol{x})$. In this case we say that *D' is obtained from $\mathbf{V}(Q_i)$ by applying inverses of view definitions.*

Such a test *succeeds* if D' satisfies Q. It is easy to see that monotonic determinacy is characterized using tests:

Lemma 5. *Q is monotonically determined over $\mathbf{V}$ if and only if every test succeeds.*

Proof. We assume Q is Boolean for simplicity. In one direction, assume Q is monotonically determined over $\mathbf{V}$, and consider a test (Q_i, D'). By virtue of (Q_i, D') being a test, we have $\mathbf{V}(D) \subseteq \mathbf{V}(D')$. Monotonic determinacy and $Q_i \models Q$ thus imply that $D' \models Q$.

In the other direction, assume every test succeeds, and consider instance $\mathcal{I}_1$ and $\mathcal{I}_2$ with $\mathcal{I}_1$ satisfying Q and $\mathbf{V}(\mathcal{I}_1) \subseteq \mathbf{V}(\mathcal{I}_2)$. As $\mathcal{I}_1 \models Q$, there is a homomorphism α from some Q_i into $\mathcal{I}_1$. Since the views are preserved under homomorphism, α is also a homomorphism from $\mathbf{V}(Q_i)$ into $\mathbf{V}(\mathcal{I}_1)$.

We will now create a D' such that (Q_i, D') forms a test, along with an extension of α that is a homomorphism taking D' into $\mathcal{I}_2$. D' will be the union of a set of facts S_F (defined below) for every fact F from $\mathbf{V}(Q_i)$. For a fact $F = V(\boldsymbol{c})$ from $\mathbf{V}(Q_i)$ let $F' = \alpha(F)$. Note that F' is in $\mathbf{V}(\mathcal{I}_1)$. By assumption, F' is also in $\mathbf{V}(\mathcal{I}_2)$. Thus there is a proof term $\tau_{F'}$ witnessing that $\mathcal{I}_2 \models F'$. Moving top-down on $\tau_{F'}$, we form a proof term for F. The root of the term $\tau_{F'}$ is labelled with the fact $\text{Goal}_V(\alpha(c_1)\ldots\alpha(c_n))$ for the goal predicate Goal_V of the Datalog program Q_V. Since α is not injective, c_i may not be unique, but we choose one such tuple $c_1 \ldots c_n$ and fix it for the transformation of $\tau_{F'}$. This choice will impact the proof term that we create, but will not impact the homomorphism extending α. We first transform $\tau_{F'}$ by replacing any element $\alpha(c_i)$ occurring in $\tau_{F'}$ by c_i. We then continue our transformation by proceeding top-down on the partially-transformed term. At the root of the term we do nothing more. In the inductive step, we consider an intensional fact $U(\boldsymbol{d})$ in τ_F witnessed by a set of facts J that are a substitution instance of some rule body B. In J, we uniformly replace any witness w to an existentially quantified variable x of B by a fresh element d_w, and extend the homomorphism to take d_w to w. We set S_F to be the union of all EDB facts occurring in the proof term we have constructed for F.

It is easy to see that the union of the facts S_F forms an appropriate D' giving a test. By assumption this test succeeds, so $D' \models Q$. But since D' is homomorphically embedded into $\mathcal{I}_2$, this means that $\mathcal{I}_2 \models Q$ as required. □

We show that monotonic determinacy is decidable for some classes of views by bounding the treewidth of all instances D' that are the second component of some test.

Theorem 3. *Suppose Q and $\mathbf{V}$ are Frontier-guarded Datalog queries. Then there is an algorithm that decides if Q is monotonically determined by $\mathbf{V}$ in 2ExpTime.*

Proof. In this proof the words "of required size" mean "doubly-exponential in Q and single-exponential in $\mathbf{V}$", $\mathbb{C}$ stands for the class of all CQ-approximations of Q, $\Sigma_\mathbf{V}$ is the view signature and Σ_B is the initial signature.

We must check whether Q holds on all tests. As observed in the proof of Theorem 1, there is an integer $k = O(|Q|, |\mathbf{V}|)$ bounding the treewidth of all CQ-approximations of Q and views in $\mathbf{V}$. Let $\mathbb{V} = \{\mathcal{F} \upharpoonright \Sigma_\mathbf{V} \mid \mathcal{F} \upharpoonright \Sigma_\mathrm{B} \in \mathbb{C}, \mathrm{TW}(\mathcal{F}) \leq k, \mathcal{F} \models \Pi_\mathbf{V}\}$. As argued in the proof of Theorem 1, $\mathbb{V}$ is k-regular and captured by an NTA $\mathfrak{A}_\mathbf{V}$ of required size.

Since Q is a monotone query, instead of checking whether all tests succeed, we will check an equivalent condition that Q holds on the class $ETEST(Q, \mathbf{V})$ which consists of all instances D' which can be obtained from an instance in $\mathbb{V}$ by applying inverses of view definitions while keeping the atoms of the view signature. Note that the treewidth of all instances in $ETEST(Q, \mathbf{V})$ is also bounded by k. By Proposition 2, for each view V with definition Q_V there exists an automaton $\mathfrak{A}'_V$ running on codes $\mathcal{T}$ which for each atom $V(\boldsymbol{c})$ at a node n checks whether n has a descendant n' such that n' contains $\boldsymbol{c}$ and the subtree of $\mathcal{T}$ rooted at n' is a code of some CQ approximation of Q_V. It should be clear that the automaton $\mathfrak{A}_{ETEST}$ obtained as the product of $\mathfrak{A}_\mathbf{V}$ and $\mathfrak{A}'_V$ for all $V \in \mathbf{V}$ captures $ETEST(Q, \mathbf{V})$.

By Proposition 5, there is an NTA $\mathfrak{A}''$ of required size which recognizes those codes which do not satisfy Q. So to check if Q is monotonically determined by V we construct the intersection of $\mathfrak{A}_{ETEST}$ and $\mathfrak{A}''$ (which is of required size) and check if it is empty. The latter check is linear in the size of the automaton. □

Using Lemma 1 and the treewidth bounds of Lemma 3 we can use the same proof technique to extend this to a mix of CQ and Frontier-guarded Datalog views, provided that Q is in Monadic Datalog.

Theorem 4. *Suppose Q is in Monadic Datalog, and $\mathbf{V}$ is a collection of CQ and Frontier-guarded Datalog views. Then there is an algorithm that decides if Q is monotonically determined by $\mathbf{V}$ in 3ExpTime.*

The previous cases of decidability required restricting the views. We now observe that if we only restrict Q to be a CQ, then we can reduce monotonic determinacy to checking equivalence between a recursive and a non-recursive query, the one created by the "simple forward backward method" of Proposition 8.

Theorem 5. *If Q is a CQ and $\mathbf{V}$ is a collection of Datalog views, then the problem of monotonic determinacy of Q over $\mathbf{V}$ is decidable in 2ExpTime.*

6 LOWER BOUNDS ON TESTING MONOTONIC DETERMINACY

We now begin our negative results, starting with lower bounds for testing monotonic determinacy. We first note some lower bounds on monotonic determinacy that can be obtained through straightforward reductions from containment or equivalence:

Proposition 9. *Monotonic determinacy is*

- *NP-hard for CQ queries and views [8, 20]*
- *Π_2^p-hard for UCQ queries and UCQ views*
- *2ExpTime-hard for CQ queries and MDL views*
- *2ExpTime-hard for MDL queries and a fixed atomic view*
- *undecidable for Datalog queries and a fixed atomic view (cf [13])*

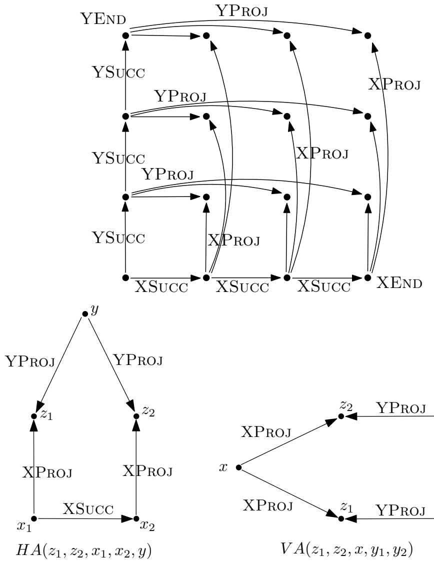
It is more challenging to get undecidability results in settings where the equivalence problem for the views and queries is decidable, as is the case for UCQs and Monadic Datalog [12]. The remainder of this section will be devoted to developing techniques for this case.

A *tiling problem* is a tuple $TP = (Tiles, HC, VC, IT, FT)$ where $Tiles = \{T_1, \ldots, T_k\}$, HC and VC are binary relations ("horizontal and vertical compatibility"), and IT and FT are subsets of tiles that must be placed at the bottom left and top right corner respectively.

A *solution* to a tiling problem consists of numbers n and m, and map $\tau : \{1, \ldots, n\} \times \{1, \ldots, m\} \longrightarrow Tiles$ such that
(T1) $(\tau(i, j), \tau(i+1, j)) \in HC$ for $1 \leq j \leq m$ and $1 \leq i < n$;
(T2) $(\tau(i, j), \tau(i, j+1)) \in VC$ for $1 \leq j < m$ and $1 \leq i \leq n$.
(T3) $\tau(1, 1) \in IT$ and (T4) $\tau(n, m) \in FT$.
By a standard reduction from the halting problem for Turing machines, it is easy to show that the problem "given a tiling problem TP, tell if it has a solution" is undecidable. By reducing this tiling problem to the problem of monotonic determinacy for MDL queries and UCQ views we obtain

Theorem 6. *The problem of monotonic determinacy for MDL queries and UCQ views is undecidable.*

The idea of the reduction is, given TP, to construct Q_{TP} and $\mathbf{V}_{TP}$ which generate tests for monotonic determinacy that look like (n, m)-grids with assignments of tiles. The query Q_{TP} will



$HA(z_1, z_2, x_1, x_2, y)$ $VA(z_1, z_2, x, y_1, y_2)$

Figure 1: A grid-like test for monotonic determinacy (a) and CQs for checking horizontal and vertical adjacency between grid points (b)

have disjuncts that return "true" when they detect violations of conditions (T1)–(T4). Thus Q_{TP} and $\mathbf{V}_{TP}$ will have a failing test for monotonic determinacy iff the tiling problem TP has a solution.

Figure 1, (a) shows such a test. We code the grid using four binary relations YSucc, XSucc, XProj, YProj and unary markers XEnd and YEnd. Vertical and horizontal axes are represented as chains of YSucc- and XSucc-atoms respectively. The "grid points" are linked via XProj- and YProj-edges to their projections on the axes. The unary predicates XEnd and YEnd mark the ends of the axes.

Note how CQs

$$\mathrm{HA}(z_1, z_2, x_1, x_2, y) = \textsc{YProj}(y, z_1) \wedge \textsc{YProj}(y, z_2) \wedge \\ \wedge \textsc{XProj}(x_1, z_1) \wedge \textsc{XProj}(x_2, z_2) \wedge \textsc{XSucc}(x_1, x_2)$$

and

$$\mathrm{VA}(z_1, z_2, x, y_1, y_2) = \textsc{YProj}(y_1, z_1) \wedge \textsc{YProj}(y_2, z_2) \wedge \\ \wedge \textsc{XProj}(x, z_1) \wedge \textsc{XProj}(x, z_2) \wedge \textsc{XSucc}(y_1, y_2)$$

(see Figure 1, (b)) can be used to check vertical and horizontal adjacency between grid points. E. g., $\mathrm{HA}(z_1, z_2, x_1, x_2, y)$ says 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. Thus the query

$$H(z_1, z_2) = \exists y \exists x_1 \exists x_2 \, \mathrm{HA}(z_1, z_2, x_1, x_2, y)$$

holds of grid points z_1^0 and z_2^0 iff z_2^0 is the right neighbour of z_1^0.

Given a tiling problem TP, we define the query Q_{TP} as a disjunction $Q_{\text{START}} \vee Q_{\text{HELPER}} \vee Q_{\text{VERIFY}}$ where Monadic Datalog query Q_{START} and UCQs Q_{HELPER} and Q_{VERIFY} are defined by the following programs:

(1) $Q_{\text{START}} \leftarrow A(x), B(x)$
(2) $A(x) \leftarrow \textsc{XSucc}(x, x'), A(x'), C(x')$
(3) $A(x) \leftarrow \textsc{XEnd}(x)$
(4) $B(y) \leftarrow \textsc{YSucc}(y, y'), B(y'), D(y')$
(5) $B(y) \leftarrow \textsc{YEnd}(y)$
(6) $Q_{\text{HELPER}} \leftarrow C(u), \textsc{YProj}(y, z), \textsc{XProj}(x, z)$
(7) $Q_{\text{HELPER}} \leftarrow D(u), \textsc{YProj}(y, z), \textsc{XProj}(x, z)$

(8) $Q_{\text{VERIFY}} \leftarrow \mathrm{HA}(z_1, z_2, y, x_1, x_2), T_i(z_1), T_j(z_2)$
for all pairs $(T_i, T_j) \notin HC$
(9) $Q_{\text{VERIFY}} \leftarrow \mathrm{VA}(z_1, z_2, y_1, y_2, x), T_i(z_1), T_j(z_2)$
for all pairs $(T_i, T_j) \notin VC$
(10) $Q_{\text{VERIFY}} \leftarrow \textsc{YSucc}(o, y), \textsc{YSucc}(y, z), \textsc{XSucc}(o, x),$
$\textsc{XProj}(x, z), T_i(z)$ for all $T_i \notin IT$
(11) $Q_{\text{VERIFY}} \leftarrow \textsc{YEnd}(y), \textsc{YProj}(y, z),$
$T_i(z), \textsc{XProj}(x, z), \textsc{XEnd}(x)$ for all $T_i \notin FT$

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

- the *grid-generating view*

$$S(x, y) \leftarrow C(x), D(y)$$
$$S(x, y) \leftarrow \textsc{XProj}(x, z), T_i(z), \textsc{YProj}(y, z) \text{ for all } T_i \text{ in } Tiles;$$

- the *atomic views* $V_{\textsc{YSucc}}$, $V_{\textsc{XSucc}}$, $V_{\textsc{YEnd}}$, $V_{\textsc{XEnd}}$ and V_{T_i} for EDBs YSucc, XSucc, YEnd, XEnd and each T_i in $Tiles$;
- the following *special* views

$$\begin{aligned}
V_C^{\text{HELPER}}(u, x, y, z) &\leftarrow C(u), \textsc{XProj}(x, z), \textsc{YProj}(y, z) \\
V_D^{\text{HELPER}}(u, x, y, z) &\leftarrow D(u), \textsc{XProj}(x, z), \textsc{YProj}(y, z) \\
V_{\text{HA}}(z_1, z_2, y, x_1, x_2) &\leftarrow \mathrm{HA}(z_1, z_2, y, x_1, x_2) \\
V_{\text{VA}}(z_1, z_2, y_1, y_2, x) &\leftarrow \mathrm{VA}(z_1, z_2, y_1, y_2, x) \\
V_I(o, x, y, z) &\leftarrow \textsc{XSucc}(o, x), \textsc{XProj}(x, z), \\
&\quad\ \textsc{YSucc}(o, y), \textsc{YProj}(y, z) \\
V_F(x, y, z) &\leftarrow \textsc{XProj}(x, z), \textsc{XEnd}(x), \\
&\quad\ \textsc{YEnd}(y), \textsc{YProj}(y, z)
\end{aligned}$$

A typical CQ-approximation of Q_{START} is shown in Figure 2 (a), and it generates the axes of the grid which are marked with unary predicates C and D. The view-image of such CQ is shown in Figure 2 (b). This view image for each grid-point contains an S-atom, and so a grid-like test as in Figure 1 (a) can be constructed out of this view image by replacing each of these S-atoms with any of the disjuncts other than the first disjunct in the definition of the grid-generating view.

When we run Q on the tests, Q_{VERIFY} comes into play. Note the correspondence between rules 8) – 11) for Q_{VERIFY} and the negations of conditions (1) – (4) in the definition of a solution of a tiling problem. Thus, when executed on a grid-test from Figure 1 (a), Q_{VERIFY} returns False iff a grid test is a solution to TP. The query Q_{HELPER} ensures that we are not harmed in the case where the grid-generating views are applied with the first rule.

We can verify that a solution of our tiling problem corresponds to monotonic determinacy, which will prove useful in both our undecidability and non-rewritability results:

Proposition 10. *Q_{TP} is not monotonically determined by $\mathbf{V}_{TP}$ iff TP has a solution.*

7 NON-REWRITABILITY

We now turn to negative results concerning rewritability.

Pebble games. In order to prove non-definability in Datalog and Monadic Datalog, we use the well-known tool of *existential pebble games*. A *partial homomorphism* from $\mathcal{I}$ to $\mathcal{I}'$ is a mapping h from a subset $D \subseteq \textsc{adom}(\mathcal{I})$ to $\textsc{adom}(\mathcal{I}')$ such that $R(c_1 \ldots c_n) \in \mathcal{I}$ implies $R(h(c_1) \ldots h(c_n)) \in \mathcal{I}'$, provided each $c_i \in D$. Let $k \geq 2$.

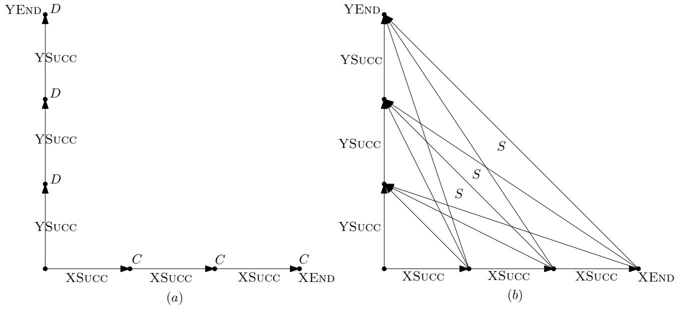


Figure 2: A typical approximation of a START atom (a) and its view image (b). (b) is obtained from (a) by replacing C and D with their cross-product $S = C \times D$

In the *existential k-pebble game* we have two players, the *Spoiler* and the *Duplicator*, each having a set of pebbles $\{\mathsf{p}_1, \ldots, \mathsf{p}_k\}$ and $\{\mathsf{q}_1, \ldots, \mathsf{q}_k\}$, respectively. The game is played on two instances $\mathcal{I}$ and $\mathcal{I}'$ over the same schema. In each round, the Spoiler either places a pebble p_i on some element of $\mathcal{I}$ or removes p_i from $\mathcal{I}$, to which the Duplicator responds by placing its corresponding pebble q_i on some element of $\mathcal{I}'$ or by removing q_i from $\mathcal{I}'$, respectively. The Duplicator wins the game if he has a *winning strategy*, i.e., if he can indefinitely continue playing the game in such a way that after each round, if $a_1, \ldots, a_k$ are the elements in $\mathcal{I}$ marked by the Spoiler's pebbles $\{\mathsf{p}_1, \ldots, \mathsf{p}_k\}$, and $a'_1, \ldots, a'_k$ are the elements in $\mathcal{I}'$ marked by the Duplicator's pebbles $\{\mathsf{q}_1, \ldots, \mathsf{q}_k\}$, then the relation $\{(a_1, a'_1), \ldots, (a_k, a'_k)\}$ is a partial homomorphism from $\mathcal{I}$ to $\mathcal{I}'$.

Recall that if there is a homomorphism from $\mathcal{I}$ to $\mathcal{I}'$, we write $\mathcal{I} \to \mathcal{I}'$. Similarly, if Duplicator wins the game on $\mathcal{I}$ and $\mathcal{I}'$, then we write $\mathcal{I} \to_k \mathcal{I}'$. Observe that $\mathcal{I} \to \mathcal{I}'$ implies $\mathcal{I} \to_k \mathcal{I}'$, for every $k \geq 2$.

The following property relates the game to homomorphisms from structures of bounded treewidth:

Fact 1. *[5] Let $k \geq 2$. Let $\mathcal{I}$ and $\mathcal{I}'$ be two instances over the same schema. Then the following are equivalent:*

(1) $\mathcal{I} \to_k \mathcal{I}'$,

(2) for every instance $\mathcal{I}''$ of treewidth $\leq k-1$, if $\mathcal{I}'' \to \mathcal{I}$, then $\mathcal{I}'' \to \mathcal{I}'$.

Existential pebble games with k pebbles preserve truth of Boolean Datalog queries with rule bodies of size at most k. Thus games can be used to show non-definability in Datalog:

Fact 2. *[19] Let Q be a Boolean query. Suppose there exists two instances $\mathcal{I}_k$ and $\mathcal{I}'_k$ such that $Q(\mathcal{I}_k) =$ True, $Q(\mathcal{I}'_k) =$ False and $\mathcal{I}_k \to_k \mathcal{I}'_k$, for infinitely many k's. Then Q is not definable in Datalog.*

Let $\mathcal{I}$ be an instance and $k \geq 2$ be an integer. An instance $\mathcal{I}'$ is a *k-unravelling* of $\mathcal{I}$ if there is a homomorphism Φ from $\mathcal{I}'$ to $\mathcal{I}$ and a tree decomposition (τ, λ) of $\mathcal{I}'$ of width at most k, such that:

(1) For each $u \in \text{VERT}(\tau)$, the mapping $\Phi|_{\lambda(u)}$ is a partial isomorphism from $\mathcal{I}'$ to $\mathcal{I}$.

(2) For $u \in \text{VERT}(\tau)$ with children $u_1, \ldots, u_\ell$, the set $\{\Phi(\lambda(u_1)), \ldots, \Phi(\lambda(u_\ell))\}$ contains the collection of all non-empty subsets of $\mathcal{I}$ of size $\leq k$.

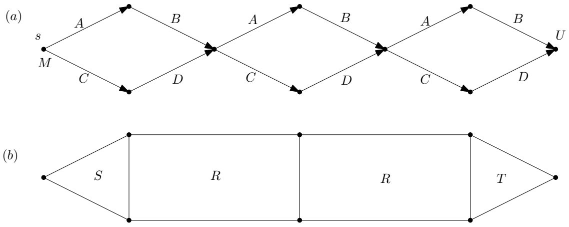


Figure 3: An unravelling of Q (a) and its view image (b)

If, in addition we have $|\lambda(u) \cap \lambda(v)| \leq 1$ for all non-equal u and v in $\text{VERT}(\tau)$, then we say that $\mathcal{I}'$ is $(1,k)$-unravelling of $\mathcal{I}$. Duplicator has a winning strategy between an instance and its $(1,k)$-unravelling in a variation of the k-pebble games in which at most one pebble can remain in place in each move. Such games preserve Boolean Monadic Datalog queries with bodies of size k, and hence each Boolean Monadic Datalog query is preserved under $(1,k)$-unravellings for sufficiently large k. So we have the following variant of Fact 2:

Fact 3. *Let Q be a Boolean query. Suppose there exists two instances $\mathcal{I}_k$ and $\mathcal{I}'$ such that $Q(\mathcal{I}_k) =$ True, $Q(\mathcal{I}'_k) =$ False and $\mathcal{I}'_k$ is a (1,k)-unravelling of $\mathcal{I}_k$, for infinitely many k's. Then Q is not definable in Monadic Datalog.*

Note that the treewidth of any k-unravelling is at most $k-1$. Observe also that all k-unravellings of an instance $\mathcal{I}$ are homomorphically equivalent. The following facts about unravellings, proved in the appendix, will be useful:

Fact 4. *Let $k \geq 2$. Let $\mathcal{I}$ be an instance and U be any k-unravelling of $\mathcal{I}$. Then the following hold:*

(1) $U \to \mathcal{I}$ and $\mathcal{I} \to_k U$.

(2) For every instance $\mathcal{I}'$, we have $\mathcal{I} \to_k \mathcal{I}'$ iff $U \to \mathcal{I}'$.

Non-rewritability in Monadic Datalog. We recall that Monadic Datalog queries monotonically determined over CQ views always have FGDL rewritings (e.g. [13], or Thm 2). We show that they may not be rewritable in MDL:

Theorem 7. *There exists a Monadic Datalog query Q and a set of CQ views $\mathbf{V}$ such that Q is rewritable with respect to $\mathbf{V}$ in Datalog – and thus monotonically determined – but not rewritable in Monadic Datalog.*

Proof. Consider the following Monadic Datalog query Q

$$\begin{array}{rcl} W(x) & \leftarrow & A(x,y), B(y,v), C(x,z), D(z,v), U(v) \\ W(x) & \leftarrow & A(x,y), B(y,v), C(x,z), D(z,v), W(v) \\ Goal & \leftarrow & W(x), M(x) \end{array}$$

and a set of views $\mathbf{V}$

$$\begin{array}{rcl} S(x,y,z) & \leftarrow & M(x), A(x,y), C(x,z) \\ R(y,z,y',z') & \leftarrow & B(y,v), D(z,v), A(v,y'), C(v,z') \\ T(y,z,v) & \leftarrow & U(v), B(y,v), D(z,v). \end{array}$$

Q checks whether the instance $\mathcal{I}$ contains the points $s \in M^{\mathcal{I}}$ and $t \in U^{\mathcal{I}}$ which are connected by a sequence of "diamonds" (see Figure 3, (a)).

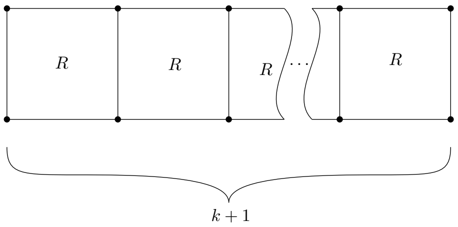


Figure 4: A long row of R rectangles

Relying on Figure 3 it is easy to see that the query Q'

$$\begin{array}{rcl} P(y,z) & \leftarrow & T(y,z,v) \\ P(y,z) & \leftarrow & R(y,z,y',z'), P(y',z') \\ \mathit{Goal} & \leftarrow & S(x,y,z), P(y,z) \end{array}$$

is a Datalog rewriting of Q with respect to the views.

We claim that there is no Monadic Datalog rewriting of Q in terms of these views. To prove this, given an integer k, we construct two instances $\mathcal{I}_k$ and $\mathcal{I}'_k$ such that $\mathcal{I}_k \models Q$, $\mathcal{I}'_k \not\models Q$, but Duplicator wins in the $(1,k)$-game for the view images $\mathbf{V}(\mathcal{I}_k)$ and $\mathbf{V}(\mathcal{I}'_k)$.

Let $\mathcal{I}_k$ be a sequence of $k+1$ diamonds from Figure 3, (a) and $\mathcal{J}_k$ be the view image of $\mathcal{I}_k$ from Figure 3, (b). Let $\mathcal{J}'_k$ be the (infinite) $(1,k)$-unravelling of $\mathcal{J}_k$. Let $\mathcal{I}'_k$ be the result of applying "inverse rules":

$$\begin{array}{rcl} S(x,y,z) & \rightarrow & M(x) \wedge A(x,y) \wedge C(x,z) \\ R(y,z,y',z') & \rightarrow & \exists v\, B(y,v) \wedge D(z,v) \wedge A(v,y') \wedge C(v,z') \\ T(y,z,v) & \rightarrow & U(v) \wedge B(y,v) \wedge D(z,v). \end{array}$$

to $\mathcal{J}'_k$ and removing the view predicates. There are two types of elements in $\mathcal{I}'_k$, those that were present in $\mathcal{J}'_k$ and those introduced by the existential quantifier over v in the second rule which are called *anonymous*.

We first claim that the view image of $\mathcal{I}'_k$ is $\mathcal{J}'_k$. To see this, consider a homomorphism h from the body of $q(y,z,y',z') = B(y,v), D(z,v), A(v,y'), C(v,z')$ into $\mathcal{I}'_k$. Note that h must map v into an anonymous point, and so y, z, y' and z' must be mapped to the points y_0, z_0, y'_0, z'_0 for some $R(y_0, z_0, y'_0, z'_0) \in \mathcal{J}'_k$. If follows that any R-atom in $V(\mathcal{I}'_k)$ is in $\mathcal{J}'_k$. This also holds for S-atoms thanks to the unary predicate M; and also for R-atoms, thanks to the unary predicate U.

Our next claim is that $\mathcal{I}'_k \models Q$ iff $\mathcal{J}'_k \models Q'$. Indeed, $\mathcal{I}_n$ maps into $\mathcal{I}'_k$ iff $\mathcal{J}_n$ maps into $\mathcal{J}'_k$.

Finally, we claim that $\mathcal{J}'_k \not\models Q'$. We show that for any n there is no homomorphism from $\mathcal{J}_n$ to $\mathcal{J}'_k$. Indeed, as $\mathcal{J}'_k$ maps homomorphically onto $\mathcal{J}_k$, for any points s, t in $\mathcal{J}'_k$ with $s \in \pi_1(S^{\mathcal{J}_k})$, $t \in \pi_3(T^{\mathcal{J}'_k})$, the distance between s and t (measured in the Gaifman graph for $\mathcal{J}'_k$) cannot be less then $k+1$. This implies the claim for $1 \leq n \leq k$. For $n \geq k+1$ the claim follows from the observation that there is no homomorphism from the query describing the pattern in Figure 4 into $\mathcal{J}'_k$. Indeed, it can be easily shown by induction that if such a homomorphism existed, then there would be a single bag in $\mathcal{J}'_k$ containing its whole image. This is a contradiction, as all bags are of size k, and the query in question has $2k+2$ variables.

It follows that $\mathcal{I}_k$ and $\mathcal{I}'_k$ are as required. □

Non-rewritability in Datalog. Now we show that monotonic determinacy does not imply Datalog rewritability, even for Monadic Datalog queries and UCQ views:

Theorem 8. *There exists a Monadic Datalog query Q and a set of UCQ views $\mathbf{V}$ such that Q is monotonically determined by $\mathbf{V}$ but there is no Datalog rewriting of Q over $\mathbf{V}$.*

The query Q and views $\mathbf{V}$ we use will be Q_{TP^*} and $\mathbf{V}_{TP^*}$ for a particular tiling problem TP^* as defined in Section 6. Define a schema $\delta := \{\mathtt{H}, \mathtt{V}, \mathtt{I}, \mathtt{F}\}$ where $\mathtt{H}, \mathtt{V}$ are binary and $\mathtt{I}, \mathtt{F}$ are unary relations. Consider a tiling problem $TP = (\mathit{Tiles}, HC, VC, IT, FT)$. Given a database instance $\mathcal{I}$ for schema δ, we say that $\mathcal{I}$ *can be tiled by TP* if there is an assignment of each element of $\mathcal{I}$ to a tile where $\mathtt{H}, \mathtt{V}, \mathtt{I}, \mathtt{F}$ satisfy the horizontal, vertical, initial and final constraints. We denote by $\mathcal{I}_{TP}$ the tiling problem TP viewed as a relational structure for δ, with Tiles the domain. Then an instance for δ can be tiled by TP exactly when it has a homomorphism into $\mathcal{I}_{TP}$. For $n, m \geq 1$, we denote by $\mathcal{I}^{grid}_{n,m}$ the database instance with domain $\{(i,j) : 1 \leq i \leq n \text{ and } 1 \leq j \leq m\}$ and facts $\mathtt{I}((1,1))$, $\mathtt{F}((n,m))$, $\mathtt{H}((i,j),(i+1,j))$, for every $1 \leq i < n$ and $1 \leq j \leq m$, and $\mathtt{V}((i,j),(i,j+1))$, for every $1 \leq i \leq n$ and $1 \leq j < m$. Then TP has a solution in the usual sense if $\mathcal{I}^{grid}_{n,m}$ can be tiled with TP.

We can adapt techniques of [4] to show that there is a tiling problem for which no $n \times m$ rectangular grid can be tiled, but where for each k large enough grids can be "k-approximately tiled", in the sense of having $k-$unravellings that can be tiled. The proof can be found in the appendix.

Lemma 6. *There is a tiling instance TP^* such that $\mathcal{I}^{grid}_{n,m}$ can not be tiled with TP^* for each $n, m \geq 1$ but for each $n, m \geq 3$ and each k with $2 \leq k < \min\{n,m\}$ any k-unravelling of $\mathcal{I}^{grid}_{n,m}$ can be tiled with TP^*.*

Proof of Theorem 8. Let TP^* be the tiling instance from Lemma 6 and let Q_{TP^*} and $\mathbf{V}_{TP^*}$ be the MDL query and UCQ views from Theorem 6. Recall that Q_{TP^*} and $\mathbf{V}_{TP^*}$ are defined over the schema

$$\sigma := \{\textsc{XSucc}, \textsc{YSucc}, C, D, \textsc{XEnd}, \textsc{YEnd}, \\ \textsc{XProj}, \textsc{YProj}, T_1, \ldots, T_p\}$$

where $\{T_1, \ldots, T_p\}$ is the tile set of TP^*. Since $\mathcal{I}^{grid}_{n,m}$ cannot be tiled with TP^*, for each $n, m \geq 1$, the tiling instance TP^* has no solution, and hence Q_{TP^*} is monotonically determined by $\mathbf{V}_{TP^*}$.

Fix $\ell \geq 10$. We shall define instances $\mathcal{I}_\ell$ and $\mathcal{I}'_\ell$ over σ such that $\mathbf{V}_{TP^*}(\mathcal{I}_\ell) \to_{\lfloor\sqrt{\ell-1}\rfloor} \mathbf{V}_{TP^*}(\mathcal{I}'_\ell)$, $Q_{TP^*}(\mathcal{I}_\ell) = \textsc{True}$ and $Q_{TP^*}(\mathcal{I}'_\ell) = \textsc{False}$. By applying Fact 2, this implies that Q_{TP^*} has no Datalog rewriting over $\mathbf{V}_{TP^*}$, as required. The instance $\mathcal{I}_\ell$ has domain $z_0 \cup X \cup Y$, where $X := \{x_1, \ldots, x_\ell\}$ and $Y := \{y_1, \ldots, y_\ell\}$, and facts $D(x_i), C(y_i)$, for all $1 \leq i \leq \ell$, along with

$$\textsc{XSucc}(x_i, x_{i+1}), \textsc{YSucc}(y_i, y_{i+1}) \text{ for all } 1 \leq i < \ell \text{ and}$$

with $\textsc{XEnd}(x_\ell)$, $\textsc{YEnd}(y_\ell)$, $\textsc{YSucc}(z_0, y_1)$ and $\textsc{XSucc}(z_0, x_1)$. Figure 2 (a) depicts $\mathcal{I}_3$. Informally, $\mathcal{I}_\ell$ is the expansion of Q_{TP^*} (more precisely of $Q_{\textsc{start}}$) representing the $(\ell \times \ell)$-grid. In particular, $Q_{TP^*}(\mathcal{I}_\ell) = \textsc{True}$.

Intuitively, we would now like to define $\mathcal{I}'_\ell$ so that its view image contains a $\lfloor\sqrt{\ell-1}\rfloor$ unravelling of the view image of $\mathcal{I}_\ell$. By

Fact 4 we would have $\mathbf{V}_{TP^*}(\mathcal{I}_\ell) \to_{\lfloor\sqrt{\ell-1}\rfloor} \mathbf{V}_{TP^*}(\mathcal{I}'_\ell)$ as required. But using Lemma 6 and the definition of Q_{TP^*} we hope to show $Q_{TP^*}(\mathcal{I}'_\ell) = \textsc{False}$. We will follow this intuition, but to define the appropriate $\mathcal{I}'_\ell$ we will need to construct several auxiliary instances. Let $E_\ell := \mathbf{V}_{TP^*}(\mathcal{I}_\ell)$. Figure 2 (b) depicts E_3. Recall that view images are defined over schema τ:

$$\{V_{\text{XSucc}}, V_{\text{YSucc}}, V_{\text{XEnd}}, V_{\text{YEnd}}, V_{T_1}, \dots, V_{T_p}, \\ V_C^{\text{HELPER}}, V_D^{\text{HELPER}}, V_{HA}, V_{VA}, V_I, V_F, S\}$$

Intuitively, E_ℓ copies the XSucc, YSucc, XEnd and YEnd-facts from $\mathcal{I}_\ell$, while the S-facts correspond to the product $Y \times X$. Let U_ℓ be a $\lfloor\sqrt{\ell-1}\rfloor$-unravelling of E_ℓ, which is witnessed by a homomorphism $\Phi : U_\ell \to E_\ell$ and a tree decomposition (τ, λ) of U_ℓ.

In order to exploit Lemma 6, we need to interpret U_ℓ as an unravelling of the grid $\mathcal{I}^{grid}_{\ell,\ell}$. The idea is to define a new instance W_ℓ over schema $\delta = \{\mathsf{H}, \mathsf{V}, \mathsf{I}, \mathsf{F}\}$ (recall that δ is the schema of $\mathcal{I}^{grid}_{\ell,\ell}$) whose domain consists of all the S-facts of U_ℓ and the horizontal and vertical successor relations are interpreted in the natural way. Thus we can think of W_ℓ as an unravelling of the S-facts of E_ℓ, which in turn correspond to grid points of $\mathcal{I}^{grid}_{\ell,\ell}$ (the fact $S(y_j, x_i)$ corresponds to the point (i, j)). Formally, W_ℓ is defined as follows:

(1) The domain of W_ℓ contains all pairs (w, z) such that $S(w, z)$ is a fact in U_ℓ.
(2) $\mathsf{I}((w, z))$ is a fact iff $\Phi(w) = y_1$ and $\Phi(z) = x_1$. Similarly, $\mathsf{F}((w, z))$ is a fact iff $\Phi(w) = y_\ell$ and $\Phi(z) = x_\ell$.
(3) $\mathsf{H}((w, z), (w', z'))$ is a fact iff $w = w'$ and $V_{\text{XSucc}}(z, z')$ is a fact in U_ℓ. Similarly, $\mathsf{V}((w, z), (w', z'))$ is a fact iff $z = z'$ and $V_{\text{YSucc}}(w, w')$ is a fact in U_ℓ.

Claim 1. *W_ℓ can be tiled by TP^*.*

Proof. We use the characterization for tilings of W_ℓ as homomorphisms into $\mathcal{I}_{TP^*}$. Then by Lemma 6 and Fact 1, it suffices to show (a) $W_\ell \to \mathcal{I}^{grid}_{\ell,\ell}$ and (b) $\text{tw}(W_\ell) \leq \ell - 2$.

For (a), we can take the homomorphism ψ such that for every (w, z) in W_ℓ, we have $\psi((w, z)) = (i, j)$ iff $\Phi(w) = y_j$ and $\Phi(z) = x_i$, for $1 \leq i, j \leq \ell$. Let us argue that ψ is a homomorphism. If $\mathsf{I}((w, z))$ is a fact in W_ℓ, by definition we have $\Phi(w) = y_1$ and $\Phi(z) = x_1$, and then $\mathsf{I}(\psi((w, z))) = \mathsf{I}((1, 1))$, which is a fact in $\mathcal{I}^{grid}_{\ell,\ell}$. If $\mathsf{H}((w, z), (w', z'))$ is a fact in W_ℓ, then $w = w'$ and $S(w, z), S(w', z'), V_{\text{XSucc}}(z, z')$ are facts in U_ℓ. It follows that $\Phi(w) = \Phi(w') = y_j$, $\Phi(z) = x_i$ and $\Phi(z') = x_{i+1}$, for some $1 \leq j \leq \ell$ and $1 \leq i < \ell$. Hence $\mathsf{H}(\psi((w, z)), \psi((w', z'))) = \mathsf{H}((i, j), (i + 1, j))$, which is a fact in $\mathcal{I}^{grid}_{\ell,\ell}$. The argument is analogous for F and V-facts.

For condition (b), recall that (τ, λ) is a decomposition of U_ℓ with $|\lambda(u)| \leq \lfloor\sqrt{\ell-1}\rfloor$, for all $u \in \text{vert}(\tau)$. We define a decomposition (τ', λ') for W_ℓ with $\tau' := \tau$ and, for each $u \in \text{vert}(\tau')$, we have $\lambda'(u) :=$

$$\{(w, z) : \{w, z\} \subseteq \lambda(u) \text{ and } S(w, z) \text{ is a fact in } U_\ell\}$$

The connectedness condition is inherited from τ. Suppose that we have a fact $\mathsf{H}((w, z), (w', z'))$ in W_ℓ. Then $w = w'$, and

$$S(w, z), S(w', z'), V_{\text{XSucc}}(z, z')$$

are facts in U_ℓ. There must exist $u \in \text{vert}(\tau) = \text{vert}(\tau')$ such that $\{w = w', z, z'\} \subseteq \lambda(u)$ (as every clique is always contained in a bag). It follows that $\{(w, z), (w', z')\} \subseteq \lambda'(u)$. The argument for V-facts is analogous. Finally, note that $|\lambda'(u)| \leq |\lambda(u)|^2 \leq \ell - 1$, for all $u \in \text{vert}(\tau')$. We conclude that the treewidth of W_ℓ is $\leq \ell - 2$ as required. ■

Using the tiling solution χ for W_ℓ given by Claim 1 and "chasing with the inverse rules of the view definitions" we can move to the desired instance $\mathcal{I}'_\ell$ for the base schema σ. The instance $\mathcal{I}'_\ell$ is obtained from U_ℓ by replacing each fact $V_{\text{XSucc}}(w, z)$, $V_{\text{YSucc}}(w, z)$, $V_{\text{XEnd}}(w)$ and $V_{\text{YEnd}}(w)$, by facts $\text{XSucc}(w, z)$, $\text{YSucc}(w, z)$, $\text{XEnd}(w)$ and $\text{YEnd}(w)$, respectively; and by replacing each fact $S(w, z)$ by three facts $\text{XProj}(z, s_{w,z})$, $\text{YProj}(w, s_{w,z})$, and $T_i(s_{w,z})$, where $s_{w,z}$ is a fresh element and $\chi((w, z)) = T_i$. By construction, all facts of U_ℓ are contained in those of $\mathbf{V}_{TP^*}(\mathcal{I}'_\ell)$ and hence $U_\ell \to \mathbf{V}_{TP^*}(\mathcal{I}'_\ell)$. By Fact 4 (2), we have $\mathbf{V}_{TP^*}(\mathcal{I}_\ell) \to_{\lfloor\sqrt{\ell-1}\rfloor} \mathbf{V}_{TP^*}(\mathcal{I}'_\ell)$.

It remains to show that $Q_{TP^*}(\mathcal{I}'_\ell) = \textsc{False}$. Since there are no C or D-facts in $\mathcal{I}'_\ell$, Q_{START} and Q_{VERIFY} cannot hold in $\mathcal{I}'_\ell$. Towards a contradiction, suppose some rule (8)–(11) holds in $\mathcal{I}'_\ell$. If rule (8) holds then there are elements w, z, z' in $\mathcal{I}'_\ell$ and facts

$$\text{YProj}(w, s_{w,z}), \text{YProj}(w, s_{w,z'}), \text{XProj}(z, s_{w,z}), \\ \text{XProj}(z', s_{w,z'}), \text{XSucc}(z, z')$$

along with $T_i(s_{w,z})$, $T_j(s_{w,z'})$, for tiles $(T_i, T_j) \notin HC$ for TP^*. By construction of $\mathcal{I}'_\ell$, we know that $S(w, z)$, $S(w, z')$ and $V_{\text{XSucc}}(z, z')$ are in U_ℓ. In particular, $\mathsf{H}((w, z), (w, z'))$ is a fact in W_ℓ. Also, by definition of $\mathcal{I}'_\ell$, we know $\chi((w, z)) = T_i$ and $\chi((w, z')) = T_j$. Since χ is a valid tiling of W_ℓ for TP^*, $(T_i, T_j) \in HC$ in TP^*; a contradiction. The case of rule (9) is symmetric. If rule (10) holds, there are u, w, z in $\mathcal{I}'_\ell$ and facts

$$\text{YSucc}(u, w), \text{XSucc}(u, z), X(z, s_{w,z}), Y(w, s_{w,z})$$

along with $T_i(s_{w,z})$, for some tile T_i not an initial tile of TP^*. It follows that $V_{\text{YSucc}}(u, w)$, $V_{\text{XSucc}}(u, z)$ and $S(w, z)$ are facts in U_ℓ. Note that $\mathsf{I}((w, z))$ is a fact in W_ℓ since Φ is a homomorphism from the unravelling U_ℓ to E_ℓ and then we must have $\Phi(u) = z_0$, $\Phi(w) = y_1$ and $\Phi(z) = x_1$. Now by definition of $\mathcal{I}'_\ell$, we know that $\chi((w, z)) = T_i$. Since χ is a valid tiling of W_ℓ for TP^*, T_i is an initial tile of TP^*, which is a contradiction. The argument for rule (11) is analogous. We conclude that $Q_{TP^*}(\mathcal{I}'_\ell) = \textsc{False}$. □

PTIME rewritability of Q_{TP^*} over $\mathbf{V}_{TP^*}$. For the query Q_{TP^*} and views $\mathbf{V}_{TP^*}$ produced in the proof of Theorem 8 there is a rewriting in a slightly larger language, stratified Datalog. The details of stratified Datalog will not concern us here, except that it includes positive Boolean combinations of Datalog queries and relational algebra queries. We show that for every tiling problem TP for which rectangular grids can not be tiled, the query Q_{TP} from Theorem 6 has a rewriting that is a positive Boolean combination of Datalog queries and relational algebra queries. In particular, this shows that Q_{TP} always has a separator in PTime.

Denote by Q^*_{START} the query obtained from Q_{START} by replacing C and D by the first and second projections of S, respectively. Let Q^*_{VERIFY} be obtained from Q^{VERIFY} by using the views. That is, by replacing:

- CQ HA by view V_{HA} and similarly for VA,

- relations T_i by the corresponding atomic views
- rewriting rules corresponding to the second to last bullet item as $V_I(o, x, y, z)$, $V_{T_i}(z)$, and similarly rewriting rules corresponding to the final bullet item using V_F.

Let ProductTest be a query that tests whether S is the product of its projections. ProductTest can be expressed in relational algebra, hence in stratified Datalog.

Consider the query R formed by existentially quantifying

$$V_C^{\text{HELPER}} \vee V_D^{\text{HELPER}} \vee Q^*_{\text{VERIFY}} \vee (Q^*_{\text{START}} \wedge \text{ProductTest})$$

Clearly R is a positive Boolean combination of Datalog queries and the relational algebra query ProductTest. We claim that R is a rewriting of q.

In one direction, suppose Q returns true on $\mathcal{I}$ and let $\mathcal{J}$ be the view image. We do a case analysis depending on which of the top-level disjuncts holds. If Q_{HELPER} holds on $\mathcal{I}$ then V_C^{HELPER} or V_D^{HELPER} is non-empty, and thus R holds in $\mathcal{J}$. If Q_{VERIFY} holds on $\mathcal{I}$ then Q^*_{VERIFY} holds on $\mathcal{J}$ and hence we conclude again that R holds on $\mathcal{J}$. Finally, suppose Q_{START} holds on $\mathcal{I}$. If ProductTest fails, we know one of C or D is empty. But then Q_{START} cannot hold, a contradiction to our assumption. Thus ProductTest must hold. From this, it is easy to see that Q^*_{START} holds. This completes the proof of this direction.

Conversely suppose that R holds on the view image $\mathcal{J}$. Again we do a case analysis on the top-level disjuncts. If V_C^{HELPER} or V_D^{HELPER} is nonempty on $\mathcal{J}$, then Q_{HELPER} holds on $\mathcal{I}$ and hence Q holds on $\mathcal{I}$. If Q^*_{VERIFY} holds on $\mathcal{J}$, then Q_{VERIFY} holds on $\mathcal{I}$, and again we conclude that Q holds on $\mathcal{I}$. Finally, suppose $Q^*_{\text{START}} \wedge$ProductTest holds on $\mathcal{J}$, and suppose that none of the disjuncts of Q hold. Note that since Q_{HELPER} fails, V_C^{HELPER} and V_D^{HELPER} must be empty. Thus we have two possibilities for S. There is the "projection case", where either one of C or D is empty, and all the S atoms are generated by the second rule. The alternative is the "product case", where both C and D are both nonempty and all the atoms of S are generated by the first rule.

We claim that we must be in the "product case" for S above. If we are in the projection case, then every pair must be associated with a tile. Further, since Q_{VERIFY} and Q_{HELPER} fail, we have a tiling of a rectangular grid, contradicting the hypothesis that there is no tiling. Since we have argued that we are in the product case, it follows that Q_{START} holds on $\mathcal{I}$ and thus Q holds in $\mathcal{I}$ as required.

Complexity of separators. Thus far we have seen that there may be no Datalog rewriting even in the case of UCQ views. What about separators, which are like rewritings, but not required to be in a logic? It is easy to see that for UCQ queries and views, there is always a rewriting in co-NP and a rewriting in NP. This is true because every view image is the view image of a small instance; basically the same observation was made for regular path queries in [14]. Thus if we want really strong lower bounds, we need to deal with recursive queries, and we need to look beyond regular path queries.

We show that when we turn to general Datalog queries and views, there may be no separator in PTime. In fact, we can find monotonically determined examples with no separator that can be performed within any given computable time bound.

Theorem 9. *There is no function F such that for all Q, $\mathbf{V}$ such that $\mathbf{V}$ and Q are in Datalog and Q is monotonically determined over $\mathbf{V}$, there is a separator of Q over $\mathbf{V}$ that runs in time $F(\mathbf{V}(I))$.*

The proof is inspired by a construction in [14] which obtained Datalog views and queries where the certain answers are difficult to compute. Roughly speaking, we modify this by considering a query verifying that the base data represent an input and a valid computation of a high-complexity deterministic Turing Machine, while the views verify that the computation is halting and return the input. Determinism of the machine will imply monotonic determinacy of the query over the views. An efficient separator will contradict the high-complexity of the machine. Details are in the appendix.

8 CONCLUSION

We have taken some basic steps in understanding monotonic determinacy for recursive queries. We leave quite a number of gaps in both the understanding of rewritability and decidability/complexity of testing monotonic determinacy, as one can see from Figures 1 and 2. To highlight just one, while we have shown that monotonic determinacy of a Datalog query over Datalog views does not imply a rewriting in any reasonable complexity class, we do not know what can be said when the query is restricted; e.g. to be in Frontier-guarded Datalog. While we have shown that when a Monadic Datalog query is monotonically determined over UCQ views, it may not be Datalog rewritable, we do not know whether a rewriting can always be obtained by expanding the language, e.g. to stratified Datalog. We have shown in the previous section that this is true of the particular rewritings constructed in Theorem 8.

ACKNOWLEDGMENTS

Benedikt and Kikot were supported by the Engineering and Physical Sciences Research Council, EP/M005852/1,EP/N014359/1, and P/L012138/1. Ostropolski-Nalewaja was supported by the Polish National Science Centre (NCN) grant 2016/23/B/ST6/01438. Romero was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

REFERENCES

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

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

[3] F. N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011.

[4] A. Atserias, A. A. Bulatov, and V. Dalmau. On the Power of k-Consistency. In *ICALP*, 2007.

[5] A. Atserias, P. Kolaitis, and M. Y. Vardi. Constraint propagation as a proof system. In *CP*, 2004.

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

[7] M. Benedikt, P. Bourhis, and M. Vanden Boom. Definability and interpolation within decidable fixpoint logics. *LMCS*, 15(3), 2019.

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

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

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

[11] S. Chaudhuri and M. Y. Vardi. On the complexity of equivalence between recursive and nonrecursive Datalog programs. In *PODS*, 1994.

[12] S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. Decidable optimization problems for database logic programs. In *STOC*, 1988.

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

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

[15] G. Gluch, J. Marcinkowski, and P. Ostropolski-Nalewaja. Can one escape red chains?: Regular path queries determinacy is undecidable. In *LICS*, 2018.

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

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

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

[19] P. Kolaitis and M. Y. Vardi. On the expressive power of Datalog: Tools and a case study. *JCSS*, 51:110–134, 1995.

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

[21] C. Lutz, J. Marti, and L. Sabellek. Query expressibility and verification in ontology-based data access. In *KR*, 2018.

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

[23] O. Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Prog.*, 15(3):231–241, 1993.

APPENDIX: PROOFS FOR SECTION 7 ON NON-REWRITABILITY RESULTS

Proof of Fact 4

Recall the statement:

Let $k \geq 2$. Let $\mathcal{I}$ be an instance and U be any k-unravelling of $\mathcal{I}$. Then the following hold:

(1) $U \to \mathcal{I}$ and $\mathcal{I} \to_k U$.
(2) For every instance $\mathcal{I}'$, we have $\mathcal{I} \to_k \mathcal{I}'$ iff $U \to \mathcal{I}'$.

For the first part, $U \to \mathcal{I}$ by definition. To see $\mathcal{I} \to_k U$, we form a strategy for the duplicator inductively, preserving the invariant that the pebbles of the duplicator are contained in a single bag of the tree decomposition. The induction step is accomplished using the second property of an unravelling.

We turn to the second part, fixing $\mathcal{I}'$. If $U \to \mathcal{I}'$ via some homomorphism h, we can apply h to the strategy witnessing $\mathcal{I} \to_k U$ to see $\mathcal{I} \to_k \mathcal{I}'$. Conversely, suppose $\mathcal{I} \to_k \mathcal{I}'$. Given $u \in U$ we know there is some some bag of the tree decomposition containing u with at most k elements, and Θ is a partial isomorphism on this bag. Consider a play for Spoiler in the pebble game from $\mathcal{I}$ to $\mathcal{I}'$ going down the branch of the tree decomposition to u. In this play, once Spoiler moves a pebble off of an element, he will never move back on to the element. Let $h(u)$ be the element in $\mathcal{I}'$ corresponding to u in the response of the duplicator playing according to his winning strategy witnessing $\mathcal{I} \to_k \mathcal{I}'$. One can verify that $h(u)$ is a homomorphism.

Proof of Lemma 6

Recall the statement:

There is a tiling instance TP^* such that $\mathcal{I}^{grid}_{n,m}$ can not be tiled with TP^* for each $n, m \geq 1$ but for each $n, m \geq 3$ and each k with $2 \leq k < \min\{n, m\}$ any k-unravelling of $\mathcal{I}^{grid}_{n,m}$ can be tiled with TP^*.

We can rephrase a tiling problem as a homomorphism problem. For a tiling problem $TP = (Tiles, HC, VC, IT, FT)$, we denote by $\mathcal{I}_{TP}$ the database instance over $\delta = \{\mathsf{H}, \mathsf{V}, \mathsf{I}, \mathsf{F}\}$ with domain $Tiles$ and facts $\mathsf{H}(T, T')$ (resp. $\mathsf{V}(T, T')$) for every $(T, T') \in HC$ (resp. $(T, T') \in VC$), and $\mathsf{I}(T)$ (resp. $\mathsf{F}(T)$) for every $T \in IT$ (resp. $T \in FT$). Then an instance can be tiled according to TP exactly when it has a homomorphism to $\mathcal{I}_{TP}$. We can thus rephrase the lemma as:

There is a tiling problem TP^* such that $\mathcal{I}^{grid}_{n,m} \not\to \mathcal{I}_{TP^*}$ for each $n, m \geq 1$, but $\mathcal{I}^{grid}_{n,m} \to_k \mathcal{I}_{TP^*}$ for each $n, m \geq 3$ and each k with $2 \leq k < \min\{n, m\}$.

Before going into the proof, we state a well-known characterization of winning strategies for the Duplicator in the existential pebble game:

Fact 5. *Let $k \geq 2$ and let $\mathcal{I}, \mathcal{I}'$ be two instances over the same schema. The Duplicator has a winning strategy in the existential k-pebble game on $\mathcal{I}$ and $\mathcal{I}'$ if and only if there is a non-empty collection $\mathcal{H}$ of partial homomorphisms from $\mathcal{I}$ to $\mathcal{I}'$ with domain size $\leq k$ such that: (1) if $f \in \mathcal{H}$ and $g \subseteq f$, then $g \in \mathcal{H}$, and (2) for each $f \in \mathcal{H}$ with domain size $< k$ and each $a \in \textsc{adom}(\mathcal{I})$, there is $g \in \mathcal{H}$ with $f \subseteq g$ whose domain contains a.*

Proof. Our proof is an adaptation of a construction from [4]. It was shown in [4] that if an instance $\mathcal{I}$ has a core of treewidth strictly bigger than k with $k \geq 2$, then there exists an instance $\mathcal{I}^*$ such that $\mathcal{I} \not\to \mathcal{I}^*$ and $\mathcal{I} \to_k \mathcal{I}^*$. We could apply this result to each $\mathcal{I}^{grid}_{n,m}$, where $n, m \geq 3$, and obtain $\mathcal{I}^*_{n,m}$ such that $\mathcal{I}^{grid}_{n,m} \not\to \mathcal{I}^*_{n,m}$ and $\mathcal{I}^{grid}_{n,m} \to_k \mathcal{I}^*_{n,m}$, for $2 \leq k < \min\{n, m\}$. By adapting the arguments in [4], we show that the family $\{\mathcal{I}^*_{n,m}\}_{n,m \geq 3}$ can actually be collapsed into a single instance $\mathcal{I}_{TP^*}$ with the desired properties.

For $n, m \geq 1$, let $G_{n,m}$ be the $(n \times m)$-grid graph. That is, $\textsc{vert}(G_{n,m}) := \{(i, j) : 1 \leq i \leq n, 1 \leq j \leq m\}$ and $\textsc{edges}(G_{n,m}) := \{\{(i, j), (i', j')\} : |i - i'| + |j - j'| = 1\}$. Observe that $G_{n,m}$ is precisely the Gaifman graph of the database instance $\mathcal{I}^{grid}_{n,m}$. Intuitively, a solution for our tiling problem on $G_{n,m}$ will describe a $0/1$ assignment to the edges of the grid $G_{n,m}$. In order to define TP^*, we consider the grid $G_{3,3}$. Intuitively, we want to think of grid points within $G_{3,3}$ as "grid point types" that can be assigned to a grid point in some larger grid $G_{n,m}$. For example, the tile $(2, 1)$ that lies in the center of the lower border represents the type of all elements that lie on the lower border of $G_{n,m}$, excluding the corner points. Our tiles will enhance each abstract grid point with a $0/1$ assignment to its incident edges.

For each vertex $u \in \textsc{vert}(G_{3,3})$, we denote by d_u the degree of u (note that $d_u \leq 4$) and fix an enumeration $e_1^u, \dots, e_{d_u}^u$ of all the edges in $G_{3,3}$ that are incident to u. The set of tiles $Tiles^*$ of TP^* contains all the tuples $(u, b_1, \dots, b_{d_u})$ such that

(1) $u \in \textsc{vert}(G_{3,3})$ and $b_1, \dots, b_{d_u} \in \{0, 1\}$,
(2) $b_1 + \cdots + b_{d_u} \equiv 0 \pmod 2$ if $u \neq (1, 1)$,
(3) $b_1 + \cdots + b_{d_u} \equiv 1 \pmod 2$ if $u = (1, 1)$.

That is, we consider assignments where the number of edges set to 1 is odd for the left-lower point but the number of edges set to 1 is even elsewhere.

Let us denote $\pi_1 : Tiles^* \to \textsc{vert}(G_{3,3})$ the first-coordinate projection. We define the set of initial and final tiles to be $IT^* := \{t \in Tiles^* : \pi_1(t) = (1, 1)\}$ and $FT^* := \{t \in Tiles^* : \pi_1(t) = (3, 3)\}$, respectively.

Our compatibility relation will ensure that the $0/1$ assignment to incident edges is consistent among adjacent nodes: if a grid point n has the outgoing edge to its right set to $b \in \{0, 1\}$ and n' is the neighbor of n to the right, then n' has the incoming edge to its left set to b.

We first give the constraints for pairs of grid points that are assigned to distinct abstract grid points in $G_{3,3}$. For each edge $e = \{u, v\} = \{(i, j), (i + 1, j)\} \in \textsc{edges}(G_{3,3})$ with $1 \leq i < 3$ and $1 \leq j \leq 3$, we add to the horizontal compatibility relation HC^* the pair $((u, b_1, \dots, b_{d_u}), (v, b'_1, \dots, b'_{d_v}))$ iff $e = e^u_\ell = e^v_m$, for some ℓ, m and $b_\ell = b'_m$. Similarly, for each edge $e = \{u, v\} = \{(i, j), (i, j+1)\} \in \textsc{edges}(G_{3,3})$ with $1 \leq i \leq 3$ and $1 \leq j < 3$, we add to the vertical compatibility relation VC^* the pair $((u, b_1, \dots, b_{d_u}), (v, b'_1, \dots, b'_{d_v}))$ iff $e = e^u_\ell = e^v_m$, for some ℓ, m and $b_\ell = b'_m$.

We now give the consistency restrictions for pairs of grid points that are assigned the same abstract grid point. We add the following pairs to HC^* and VC^*:

- For $u = (2, j)$ with $j \in \{1, 3\}$, the pair $((u, b_1, \dots, b_{d_u}), (u, b'_1, \dots, b'_{d_u})) \in HC^*$ iff $e = \{(2, j), (3, j)\}$, $e' = \{(1, j), (2, j)\}$, $e = e^u_\ell$, $e' = e^u_m$, for some ℓ, m, and $b_\ell = b'_m$,
- For $u = (i, 2)$ with $i \in \{1, 3\}$, the pair $((u, b_1, \dots, b_{d_u}), (u, b'_1, \dots, b'_{d_u})) \in VC^*$ iff $e = \{(i, 2), (i, 3)\}$, $e' = \{(i, 1), (i, 2)\}$, $e = e^u_\ell$, $e' = e^u_m$, for some ℓ, m, and $b_\ell = b'_m$,
- For $u = (2, 2)$, the pair $((u, b_1, \dots, b_{d_u}), (u, b'_1, \dots, b'_{d_u})) \in HC^*$ iff $e = \{(2, 2), (3, 2)\}$, $e' = \{(1, 2), (2, 2)\}$, $e = e^u_\ell$, $e' = e^u_m$, for some ℓ, m, and $b_\ell = b'_m$; and the pair
$$((u, b_1, \dots, b_{d_u}), (u, b'_1, \dots, b'_{d_u})) \in VC^*$$
if and only if $e = \{(2, 2), (2, 3)\}$, $e' = \{(2, 1), (2, 2)\}$, $e = e^u_\ell$, $e' = e^u_m$, for some ℓ, m, and $b_\ell = b'_m$.

Let $n, m \geq 3$. We define a function Ψ from $\textsc{vert}(G_{n,m})$ to $\textsc{vert}(G_{3,3})$ as follows. We set $\Psi((1, 1)) = (1, 1)$, $\Psi((n, 1)) = (3, 1)$, $\Psi((1, m)) = (1, 3)$ and $\Psi((n, m)) = (3, 3)$. For $1 < i < n$ and $1 < j < m$, we define $\Psi((i, j)) = (2, 2)$, $\Psi((1, j)) = (1, 2)$, $\Psi((n, j)) = (3, 2)$, $\Psi((i, 1)) = (2, 1)$ and $\Psi((i, m)) = (2, 3)$. We can now enumerate incident edges of a in $G_{n,m}$ according to the already defined enumeration for $\Psi(a)$ in $G_{3,3}$. More precisely, for each $a \in \textsc{vert}(G_{n,m})$, we define a bijection Δ_a from its incident edges in $G_{n,m}$ to the incident edges of $\Psi(a)$ in $G_{3,3}$ in the natural way: if e corresponds to the incident edge of a to the "up" direction in the grid $G_{n,m}$ then $\Delta_a(e)$ is also the incident edge of $\Psi(a)$ in the grid $G_{3,3}$ to the "up" direction; similarly for the "right", "down" and "left" directions. Then for each $a \in \textsc{vert}(G_{n,m})$, we enumerate its incident edges as $e^a_1, \dots, e^a_{d_a} = \Delta_a^{-1}(e_1^{\Psi(a)}), \dots, \Delta_a^{-1}(e^{\Psi(a)}_{d_{\Psi(a)}})$, where $e_1^{\Psi(a)}, \dots, e^{\Psi(a)}_{d_{\Psi(a)}}$ is the enumeration for $\Psi(a)$ already fixed in the construction of TP^*.

We now formalize the intuition that the parity and consistency conditions ensure that a rectangular grid cannot be tiled:

Claim 2. *$\mathcal{I}^{grid}_{n,m} \not\to \mathcal{I}_{TP^*}$, for every $n, m \geq 1$.*

Proof. Note that $\mathcal{I}^{grid}_{n,m} \not\to \mathcal{I}_{TP^*}$ if $\min\{n, m\} \leq 2$. Towards a contradiction, suppose $\mathcal{I}^{grid}_{n,m} \to \mathcal{I}_{TP^*}$ for some $n, m \geq 3$, via a homomorphism h. By construction, we must have $\pi_1(h(a)) = \Psi(a)$, for every a in $\mathcal{I}^{grid}_{n,m}$ and hence h corresponds to a $0/1$ assignment of the edges of the Gaifman graph $G_{n,m}$ of $\mathcal{I}^{grid}_{n,m}$. In particular, there exists a $0/1$ vector $(x_e)_{e \in E(G_{n,m})}$ such that for each verted $a \in \textsc{vert}(G_{n,m})$, we have $h(a) = (\Psi(a), x_{e^a_1}, \dots, x_{e^a_{d_a}})$.

Now we have
$$\sum_{a \in \textsc{vert}(G_{n,m})} (x_{e^a_1} + \cdots + x_{e^a_{d_a}}) =$$
$$(x_{e_1^{(1,1)}} + \cdots + x_{e^{(1,1)}_{d_{(1,1)}}}) + \sum_{a \in \textsc{vert}(G_{n,m}) \setminus \{(1,1)\}} (x_{e^a_1} + \cdots + x_{e^a_{d_a}})$$
$$= 1 \pmod 2$$

But this is impossible as each edge $e \in \textsc{edges}(G_{n,m})$ is counted exactly twice in $\sum_{a \in \textsc{vert}(G_{n,m})} (x_{e^a_1} + \cdots + x_{e^a_{d_a}})$; a contradiction. ■

While there is no total mapping from $\mathcal{I}^{grid}_{n,m}$ to $\mathcal{I}_{TP^*}$ that is a homomorphism, by considering partial mappings with domains that are not too large, we can easily satisfy the correct parity conditions, and hence we can define partial homomorphisms from $\mathcal{I}^{grid}_{n,m}$ to $\mathcal{I}_{TP^*}$. The next claim tells us that these partial homomorphisms can be chosen to be consistent.

Claim 3. *$\mathcal{I}^{grid}_{n,m} \to_k \mathcal{I}_{TP^*}$, for every $n, m \geq 3$ and $2 \leq k < \min\{n, m\}$.*

Proof. Let $P = (a_0, a_1, \dots, a_\ell)$ be a walk in $G_{n,m}$. For every edge $e \in \textsc{edges}(G_{n,m})$, we define:

(1) $x^P_e = 1$ if P visits e an odd number of times.
(2) $x^P_e = 0$ if P visits e an even number of times.

We also define $h^P(a) := (\Psi(a), x^P_{e^a_1}, \dots, x^P_{e^a_{d_a}})$, for each $a \in \textsc{vert}(G_{n,m})$ (i.e., in the domain of $\mathcal{I}^{grid}_{n,m}$).

Let $\mathcal{W}$ be the collection of all walks $P = (a_0, a_1, \dots, a_\ell)$ in $G_{n,m}$ with $a_0 = (1, 1)$ and $a_\ell \neq a_0$. We claim that for each $P = (a_0, a_1, \dots, a_\ell) \in \mathcal{W}$ and each $a \neq a_\ell$ in $\textsc{vert}(G_{n,m})$, the tuple $h^P(a)$ always belongs to the domain of $\mathcal{I}_{TP^*}$. Note that $x^P_{e^a_1} + \cdots + x^P_{e^a_{d_a}} = |\{e \text{ in } P : e \text{ is incident to } a\}| \pmod 2$. For $a \neq a_0$, we have

$|\{e \text{ in } P : e \text{ is incident to } a\}| = 2 \cdot |\{i : 0 < i < \ell \text{ and } a_i = a\}| = 0$ (mod 2), and hence $h^P(a) = (\Psi(a), x^P_{e^a_1}, \ldots, x^P_{e^a_{d_a}})$ belongs to $\mathcal{I}_{TP^*}$ (as $\Psi(a) \neq (1,1)$). On the other hand, for $a = a_0$, we have $|\{e \text{ in } P : e \text{ is incident to } a\}| = 1 + 2 \cdot |\{i : 0 < i < \ell \text{ and } a_i = a\}| = 1$ (mod 2), and hence $h^P(a) = (\Psi(a), x^P_{e^a_1}, \ldots, x^P_{e^a_{d_a}})$ belongs to $\mathcal{I}_{TP^*}$ (as $\Psi(a) = (1,1)$).

Thus we can define for each walk $P = (a_0, a_1, \ldots, a_\ell) \in \mathcal{W}$ a partial mapping h^P from $\mathcal{I}^{grid}_{n,m}$ to $\mathcal{I}_{TP^*}$ with domain $\textsc{vert}(G_{n,m}) \backslash \{a_\ell\}$. By definition of TP^* and since h^P is defined from a 0/1 vector $(x^P_e)_{e \in \textsc{edges}(G_{n,m})}$, we have that h^P is actually a partial homomorphism.

We define a non-empty collection $\mathcal{H}$ of partial homomorphisms from $\mathcal{I}^{grid}_{n,m}$ to $\mathcal{I}_{TP^*}$ as follows. For $1 \leq p \leq n$ and $1 \leq q \leq m$, we denote by $C_{p,q}$ the (p,q)-*cross* of $G_{n,m}$ defined as $C_{p,q} := \{(p,j) : 1 \leq j \leq m\} \cup \{(i,q) : 1 \leq i \leq n\}$. For every non-empty subset $S \subseteq \textsc{vert}(G_{n,m})$ with $|S| \leq k$ (recall that $2 \leq k < \min\{n,m\}$), and every walk $P = (a_0, \ldots, a_\ell) \in \mathcal{W}$ such that there are p, q with $a_\ell \in C_{p,q}$ and $C_{p,q} \cap S = \emptyset$, we add to $\mathcal{H}$ the restriction $h^P|_S$. We prove that $\mathcal{H}$ is a winning strategy for the Duplicator and then $\mathcal{I}^{grid}_{n,m} \rightarrow_k \mathcal{I}_{TP^*}$ as required. Condition (1) of Fact 5 holds by definition, so we focus on condition (2). Let $h^P|_S \in \mathcal{H}$ for some S with $|S| < k$ and walk $P = (a_0, \ldots, a_\ell) \in \mathcal{W}$ such that $a_\ell \in C_{p,q}$ and $C_{p,q} \cap S = \emptyset$ for some p, q. Let $a \in \textsc{vert}(G_{n,m}) \setminus S$ and $S' = S \cup \{a\}$. Since $k < \min\{n,m\}$, there exist p', q' such that $C_{p',q'} \cap S' = \emptyset$. Moreover, since $C_{p,q}$ is connected and $|C_{p,q} \cap C_{p',q'}| \geq 2$, there is a walk $P'' = (a_\ell, a_{\ell+1}, \ldots, a_{\ell+r})$ such that $a_{\ell+i} \in C_{p,q}$, for all $0 \leq i \leq r$, $a_{\ell+r} \in C_{p',q'}$ and $a_{\ell+r} \neq a_0$. Let $P' = (a_0, \ldots, a_{\ell+r})$ be the concatenation of P and P''. Then $h^{P'}|_{S'} \in \mathcal{H}$. Finally, observe that $h^P(b) = h^{P'}(b)$, for every $b \in S$, since x^P_e and $x^{P'}_e$ can only differ for edges $e = \{b', b''\} \subseteq C_{p,q}$ and $C_{p,q} \cap S = \emptyset$. It follows that $h^P|_S \subseteq h^{P'}|_{S'}$, and hence condition (2) holds. ■

□

Proof of Theorem 9

Recall the statement:

> There is no integer-valued function F such that for all $Q, \mathbf{V}$ such that $\mathbf{V}$ and Q are in Datalog and Q is monotonically determined over V, there is a separator of Q over $\mathbf{V}$ that runs in time $F(\mathbf{V}(I))$.

We now give the proof of Theorem 9. We assume the opposite, aiming for a contradiction. We use the following fact, which is a consequence of the time hierarchy theorem:

> For any computable function F there is a deterministic Turing machine M_F which halts on all of its inputs, and such that no Turing machine running in time F can decide the same language as M_F.

Fix such a machine M for F.

Let $\Sigma_{\textsc{input}}$ be the input alphabet of M, and Σ_M be a suitable alphabet for encoding configurations of M.

We consider a base signature with relations $\textsc{Succ}(x,y)$ and $U_a(x) : a \in \Sigma_{\textsc{input}}$ for the input signature of M along with symbols

$$\textsc{Succ}'(x,y), U'_a(x) : a \in \Sigma_M$$

for the configuration signature of M.

A *pre-run-string* is a string in the regular language formed by intersecting $\sigma_{\textsc{InpBegin}}\ (\Sigma_{\textsc{input}})^* \sigma_{\textsc{InpEnd}} (\Sigma^*_M;)^+ \sigma_{\textsc{RunEnd}}$ with a regular expression enforcing that the last maximal segment of Σ_M strings that does not contain ; encodes a halting state. Above:

- $\sigma_{\textsc{InpBegin}}$ is a marker designating the beginning of the input while
- $\sigma_{\textsc{InpEnd}}$ designates the end of the input;
- ; is a marker indicating the separator between configurations, while
- $\sigma_{\textsc{RunEnd}}$ marks the end of the run.

A *well-shaped string* will consist of an initial letter with a special symbol $\sigma_{\textsc{InpBegin}}$ and ending with $\sigma_{\textsc{InpEnd}}$, followed by a code for a run of M, ending with a special symbol $\sigma_{\textsc{RunEnd}}$. A string is *badly-shaped* if it is not well-shaped. It is easy to see that a badly-shaped string w has at least one of the following *bad properties*: w is not a pre-run string, w contains a sub-string $;c_i;c_{i+1};$ where c_{i+1} does not encode a next configuration after c_i, w contains a string $\sigma_{\textsc{InpBegin}} w_{in} \sigma_{\textsc{InpEnd}} c_1;$ such that c_1 does not encode initial configuration of M with input w_{in}.

A *pre-run instance* will be a relational encoding of a homomorphic image of a pre-run string using relations $\textsc{Succ}(x,y)$ and $U_a(x) : a \in \Sigma_{\textsc{input}}$ for the coding of the initial segment, symbols $\textsc{Succ}'(x,y), U'_a(x) : a \in \Sigma_M$ for the remaining part of the run, and additional symbols for the separators. That is, in the relational encoding we allow the same element to represent different places in the string. A *well-shaped string instance* will be a relational encoding of a homomorphic image of a well-shaped string, again using the relations $\textsc{Succ}(x,y), U_a : a \in \Sigma_{\textsc{input}}$ for the initial segment and the primed copies for the remaining segments. We define a badly-shaped string instance analogously.

A standard argument shows

Proposition 11. *There is a Datalog query whose approximations are (up to isomorphism) exactly the badly-shaped string instances.*

Note that if we had enforced that codings were *alternating*, with every other configuration reversed, then we could use a PDA to detect bad properties on a string and a context-free path query to detect it on the encoding. With the power of general Datalog, no alternation is needed.

Our views $\mathbf{V}$ will include:

- the *input views*, with one binary view returning exactly $\textsc{Succ}(x,y)$, and for each $a \in \Sigma_{input}$ a unary view returning $U_a(x)$.
- a nullary view $V^{\textsc{badly-shaped}}$ which returns $\textsc{True}$ whenever the instance contains a badly-shaped string instance. That is, $V^{\textsc{badly-shaped}}$ returns $\textsc{True}$ when the input contains the homomorphic image of a relational encoding of a string starting with the symbol $\sigma_{\textsc{InpBegin}}$ and ending with the symbol $\sigma_{\textsc{InpEnd}}$ which has one of the bad properties. By Proposition 11, a Datalog view with this property exists.
- a unary view $V^{\textsc{pre-run}}(x)$ which holds for x if there is a subinstance that is a pre-run instance in which the occurrence of $\sigma_{\textsc{InpEnd}}$ corresponds to x.

Our query Q will be the sentence obtained from $V^{\textsc{badly-shaped}}$ disjoined with $Q^{\textsc{Accept}}$, where $Q^{\textsc{Accept}}$ returns true exactly when we detect a relational encoding of a pre-run string that ends in an accept state.

We now argue that Q is monotonically determined over $\mathbf{V}$.

Consider instances $\mathcal{I}_1$ and $\mathcal{I}_2$ with $\mathbf{V}(\mathcal{I}_1) \subseteq \mathbf{V}(\mathcal{I}_2)$ and $Q(\mathcal{I}_1)$ being true.

$Q(\mathcal{I}_1)$ could be true because $V^{\textsc{badly-shaped}}$ holds, in this case, $Q(\mathcal{I}_2)$ also holds since $V^{\textsc{badly-shaped}}$ is one of the views. So we can assume that $V^{\textsc{badly-shaped}}$ does not hold in $\mathcal{I}_1$ or $\mathcal{I}_2$, since if it does hold then $\mathcal{I}_2$ satisfies Q.

$Q(\mathcal{I}_1)$ could also be true because $V^{\textsc{badly-shaped}}$ fails but $Q^{\textsc{Accept}}$ holds. We know there is a relational encoding of some string

$$\sigma_{\textsc{InpBegin}} w \sigma_{\textsc{InpEnd}} w_0; w_1; \ldots; w_n \sigma_{\textsc{RunEnd}}$$

witnessing that $Q^{\textsc{Accept}}$ holds in $\mathcal{I}_1$. Let x be the element corresponding to the label $\sigma_{\textsc{InpEnd}}$ in this encoding. Note that $V^{\textsc{pre-run}}$ must hold of x in $\mathcal{I}_1$, hence in $\mathcal{I}_2$. The latter must be witnessed via a relational encoding of some string of the form

$$\sigma_{\textsc{InpBegin}} w'_0 \sigma_{\textsc{InpEnd}} w'_1; \ldots; w'_k$$

with $\sigma_{\textsc{InpBegin}} w'_0 \sigma_{\textsc{InpEnd}}$ relationally encoded in the unprimed signature, the w'_i encoded in the primed signature, with the element labelled by $\sigma_{\textsc{InpEnd}}$ corresponding to x. Note that by the definition of pre-run, w'_k must include a halting state.

Since we have views for all of the input signature elements, and $\mathbf{V}(\mathcal{I}_1) \subseteq \mathbf{V}(\mathcal{I}_2)$, we know that we also have an encoding of a string $\sigma_{\textsc{InpBegin}} w_0 \sigma_{\textsc{InpEnd}}$ in $\mathcal{I}_2$, with the encoding done in the unprimed signature, with the

We now consider the string

$$s = \sigma_{\textsc{InpBegin}}\ w_0\ \sigma_{\textsc{InpEnd}}\ w'_1; \ldots; w'_k\ \sigma_{\textsc{RunEnd}}$$

s begins with the input string, and ends with a halting state. Note that since a relational encoding of $\sigma_{\textsc{InpBegin}}\ w_0\ \sigma_{\textsc{InpEnd}}$ lies in $\mathcal{I}_1$, the encoding of s must lie in $\mathcal{I}_2$, due to the input views. Since $V^{\textsc{badly-shaped}}$ is false in $\mathcal{I}_2$, we know that in $\mathcal{I}_2$:

- For every relational encoding of a string of form:

  $$\sigma_{\textsc{InpBegin}}\ w_0\ \sigma_{\textsc{InpEnd}}\ w'_1;$$

  with $\sigma_{\textsc{InpBegin}}\ w_0\ \sigma_{\textsc{InpEnd}}$ encoded in the unprimed signature and w'_1 encodes a state with tape configuration w_0 and state the initial state of M, under the transition relation of M.
- For every relational encoding a string of the form:

  $$w'_1; w'_2;$$

  with the encoding being in the primed signature, w'_2 must encode a state that is a successor in the transition relation of M of the state encoded by w'_1.

From this we infer that s is an encoding of a run of M on w_0, ending at a halting state.

But since M is deterministic, s must be the same as

$$\sigma_{\textsc{InpBegin}}\ w_0\ \sigma_{\textsc{InpEnd}} w_1; \ldots; w_n \sigma_{\textsc{RunEnd}}$$

which ends in an acceptance state.

Since a relational encoding of s lies in $\mathcal{I}_2$, we can conclude that Q holds in $\mathcal{I}_2$. This completes the argument for monotonic determinacy of Q with respect to $\mathbf{V}$.

Now, suppose Q has a separator R that runs in time F. Then R will allow us to check in time F whether M accepts or rejects on its input, a contradiction. Thus we have completed the proof of Theorem 9.