Right-Adjoints for Datalog Programs

Balder ten Cate

Institute for Logic, Language, and Computation, University of Amsterdam, The Netherlands

Víctor Dalmau

Department of Information and Communication Technologies, Universitat Pompeu Fabra, Spain

Jakub Opršal

School of Computer Science, University of Birmingham, UK

Abstract

A Datalog program can be viewed as a syntactic specification of a mapping from database instances over some schema to database instances over another schema. We establish a large class of Datalog programs for which this mapping admits a (generalized) right-adjoint. We employ these results to obtain new insights into the existence of, and methods for constructing, homomorphism dualities within restricted classes of instances. From this, we derive new results regarding the existence of uniquely characterizing data examples for database queries in the presence of integrity constraints.

2012 ACM Subject Classification Theory of computation → Logic and databases

Keywords and phrases Datalog, Adjoints, Homomorphism Dualities, Database Constraints, Conjunctive Queries, Data Examples

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.10

Funding *Balder ten Cate*: Supported by the European Union's Horizon 2020 research and innovation programme (MSCA-101031081).
Víctor Dalmau: Supported by the MiCin under grants PID2019-109137GB-C22 and PID2022-138506NB-C22, and the Maria de Maeztu program (CEX2021-001195-M).
Jakub Opršal: Supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No 101034413.

Acknowledgements Thanks to Pierre Bourrhis for detailed feedback and catching mistakes.

1 Introduction

Datalog is a rule-based language for specifying mappings from database instances over an input schema $\mathbf{S}_{in}$, to database instances over an output schema $\mathbf{S}_{out}$.

▶ **Example 1.1.** Consider the Datalog program defined by the following rules:

$$\begin{aligned}
\texttt{Path}(x,y) &:- \texttt{Edge}(x,y).\\
\texttt{Path}(x,y) &:- \texttt{Edge}(x,z), \texttt{Path}(z,y).\\
\texttt{Ans}(x,y) &:- \texttt{Path}(x,y).
\end{aligned}$$

This Datalog program takes as input an instance over an input schema {Edge}, and produces as output an instance over the schema {Ans}, where Ans is the transitive closure of Edge.

We study the existence of right-adjoints and generalized right-adjoints for Datalog programs, where a *right-adjoint* for a Datalog program P is a function Ω from $\mathbf{S}_{out}$-instances to $\mathbf{S}_{in}$-instances, such that for all $\mathbf{S}_{in}$-instances I and $\mathbf{S}_{out}$-instances J, $P(I) \to J$ iff $I \to \Omega(J)$, where "→" denotes the existence of a homomorphism. *Generalized* right-adjoints are defined similarly, loosely speaking, except that we allow Ω to map each $\mathbf{S}_{out}$-instance to a *finite set* of $\mathbf{S}_{in}$-instances, such that, $P(I) \to J$ iff $I \to J'$ for some $J' \in \Omega(J)$. We identify large classes of Datalog programs for which a right-adjoint, respectively, a generalized right-adjoint, exists. For instance, it will follow from our results that the Datalog program from Example 1.1 has a right-adjoint.

Our motivation for studying (generalized) right-adjoints for Datalog programs comes from the fact that they provide us with a means of constructing homomorphism dualities. A homomorphism duality is a pair (F, D) where F and D are sets of instances, such that an arbitrary instance A admits a homomorphism from an instance in F if and only if A does not admit a homomorphism to any instance in D. In other words, homomorphism dualities equate the existence of a homomorphism of one kind to the non-existence of a homomorphism of another kind. Homomorphism dualities have been studied extensively in the literature on constraint satisfaction problems, and have also found several applications in database theory (e.g., for schema mapping design [2], ontology-mediated data access [6], and query inference from data examples [9, 10]). In particular, in [2, 9], homomorphism dualities were used as a tool for studying the unique characterizability, and exact learnability, of schema mappings and of conjunctive queries. Using the same approach, we can use our results on right-adjoints to derive new results on unique characterizations for (unions of) conjunctive queries in the presence of integrity constraints. In the process, we also obtain a new technique for constructing homomorphism dualities within restricted classes of structures, e.g., transitive digraphs.

▶ **Contribution 1** (Section 3)**.** We introduce a new fragment of Datalog called *TAM Datalog (Tree-Shaped Almost-Monadic Datalog).* We characterize TAM Datalog as a fragment of Monadic Second-Order Logic, and we prove that TAM Datalog is closed under composition.

▶ **Contribution 2** (Section 4)**.** We show that every connected TAM Datalog program has a right-adjoint, and that every TAM Datalog program has a generalized right-adjoint. We show by means of counterexamples that each of the syntactic conditions imposed by TAM Datalog is necessary for the existence of generalized right-adjoints.

▶ **Contribution 3** (Section 5)**.** We investigate the relationship between generalized right-adjoints and homomorphism dualities. Generalized right-adjoints can be used for constructing homomorphism dualities. We show that all tree dualities can be accounted for in this way.

▶ **Contribution 4** (Section 6)**.** Following the approach in [2, 9], we derive new results on unique characterizations for (unions of) conjunctive queries in the presence of integrity constraints. In the process, we obtain a new technique for constructing homomorphism dualities within restricted classes of structures, e.g., transitive digraphs.

Some proofs are omitted and can be found in an appendix of the full version of this paper.

Related Work Foniok and Tardif [17] studied the existence of right adjoints to Pultr functors which are themselves right adjoints [22] in the special case of digraphs. Translating into our terms a Pultr functor is an interpretation (of digraphs in digraphs) (ϕ_V, ϕ_E) where ϕ_V and ϕ_E are conjunctive queries (with k and $2k$ free variables, respectively, for some $k \geq 1$) defining the output node-set and edge-set respectively. For the special case where ϕ_V just returns the input node-set, it was shown in [17] that the functor defined by (ϕ_V, ϕ_E) has a right adjoint if ϕ_E is connected and acyclic. The setup and characterization were generalized in [13] to arbitrary relational structures. We extensively build on the framework and concepts in [13], but we permit the interpretation to be specified by an arbitrary Datalog program, so that our setup is able to encompass common types of database dependencies.

To our knowledge, this is the first time that adjoints for functors defined by Datalog programs have been studied. Also, it is the first application of functors with right adjoints in the context of unique characterization of database queries. In a different setting, namely approximate graph coloring, the "arc graph" functor was used in [21] where it is additionally argued that functors with a right adjoint, more generally, can play a role in the design and analysis of reductions between (promise) constraint satisfaction problems. The use of Datalog programs for reductions between such problems, although without reference to adjoints, is discussed in [14].

2 Preliminaries

Schemas, Instances, Homomorphisms A *schema* $\mathbf{S}$ is a finite collection of relation symbols R with specified arity $\text{arity}(R) \geq 0$. An $\mathbf{S}$-*instance* I is a finite set of facts, where a fact is an expression of the form $R(a_1, \dots, a_n)$ with $R \in \mathbf{S}$ and $n = \text{arity}(R)$. Unless specified otherwise, instances are always assumed to be finite. The *active domain* $\text{adom}(I)$ of I is the set of all values a_i occurring in the facts of I. A *homomorphism* $h : I \to J$, where I and J are instances over the same schema $\mathbf{S}$, is a function from $\text{adom}(I)$ to $\text{adom}(J)$ such that the h-image of every fact of I is a fact of J. We will denote by $\text{Inst}[\mathbf{S}]$ the set of all $\mathbf{S}$-instances.

A k-ary *pointed* $\mathbf{S}$-*instance* (for $k \geq 0$) is a pair $(I, \mathbf{a})$ where I is an $\mathbf{S}$-instance and $\mathbf{a}$ a k-tuple of elements of $\text{adom}(I)$, called *distinguished elements*. A homomorphism $h : (I, \mathbf{a}) \to (J, \mathbf{b})$ is a homomorphism $h : I \to J$ such that $h(\mathbf{a}) = \mathbf{b}$.

Incidence Graph, Connectedness, C-Acyclicity The *incidence graph* of an instance I is the bipartite multi-graph whose nodes are the elements and the facts of I, and where there is a distinct (undirected) edge (a, f) for every occurrence of the element a in the fact f. We say that an instance is *connected* if its incidence graph is connected, and an instance is *acyclic* if its incidence graph is acyclic. A pointed instance $(I, \mathbf{a})$ is *c-acyclic* if every cycle in the incidence graph of I contains at least one element from the tuple $\mathbf{a}$.

Conjunctive Queries and Unions of Conjunctive Queries For $\mathbf{S}$ a schema and $k \geq 0$, a *k-ary conjunctive query (CQ) over* $\mathbf{S}$ is an expression of the form

$$q(y_1, \dots, y_k) :- \exists \mathbf{x}(\phi_1 \wedge \cdots \wedge \phi_n) \qquad \text{(Eq. 1)}$$

where each ϕ_i is a relational atomic formula, and such that each variable y_i occurs in at least one conjunct ϕ_j. A *k-ary union of conjunctive queries (UCQ) over* $\mathbf{S}$ is a finite disjunction of k-ary CQs over $\mathbf{S}$. We denote by $q(I)$ the set of tuples $\mathbf{a}$ for which it holds that $I \models q(\mathbf{a})$.

The *canonical instance* of a CQ of the form (Eq. 1) is the pointed instance $(I, \mathbf{y})$ where I is the instance with active domain $\{y_1, \dots, y_k, \mathbf{x}\}$ whose facts are the conjuncts of ϕ, and $\mathbf{y} = y_1 \dots y_k$. Conversely, the *canonical CQ* of a pointed instance $(I, \mathbf{a})$ with $\mathbf{a} = a_1 \dots a_k$, is obtained by associating a unique variable y_a to each $a \in \text{adom}(I)$, letting $\mathbf{x}$ be an enumeration of all variables y_a for $a \in \text{adom}(I) \setminus \{a_1, \dots, a_k\}$, and taking the query $q(y_{a_1}, \dots, y_{a_k}) :- \exists \mathbf{x} \bigwedge_{R(b_1,\dots,b_n) \in I} R(y_{b_1}, \dots, y_{b_n})$. By the well-known Chandra-Merlin theorem, a tuple $\mathbf{a}$ belongs to $q(I)$ if and only if the canonical instance of q homomorphically maps to $(I, \mathbf{a})$.

We call a UCQ q *c-acyclic* if the (pointed) canonical instance of each CQ in q is c-acyclic.

Datalog A Datalog program is specified by a collection of rules, and it defines a mapping from instances over a schema $\mathbf{S}_{in}$ (traditionally known as the EDB schema) to instances over a schema $\mathbf{S}_{out}$ (traditionally known as the IDB schema). The presentation we will give here also allows for auxiliary IDB relations that are not exposed in the output schema.

▶ **Definition 2.1** (Datalog Program). *A* Datalog program *is a tuple* $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \mathbf{S}_{aux}, \Sigma)$ *where* $\mathbf{S}_{in}, \mathbf{S}_{out}, \mathbf{S}_{aux}$ *are mutually disjoint schemas, and* Σ *is a set of rules of the form*

$$S(\mathbf{x}) :- R_1(\mathbf{y}_1), \ldots, R_n(\mathbf{y}_n)$$

where $S \in \mathbf{S}_{out} \cup \mathbf{S}_{aux}$*, each* $R_i \in \mathbf{S}_{in} \cup \mathbf{S}_{aux}$*, and each variable in* $\mathbf{x}$ *occurs in* $\mathbf{y}_i$ *for some* i*.*

If P is a Datalog program, then we will use often use the notation $\mathbf{S}^P_{in}$, $\mathbf{S}^P_{out}$, $\mathbf{S}^P_{aux}$, and Σ^P to refer to the constituents of the tuple P.

The *head* of a rule is the part to the left of the :– sign, and the *body* is the part to the right. The *canonical instance* of a Datalog rule $R_0(\mathbf{x}_0) :- R_1(\mathbf{x}_1), \ldots, R_n(\mathbf{x}_n)$ is the pointed instance whose active domain is $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, whose facts are the conjuncts $R_i(\mathbf{x}_i)$ of the rule body, and whose sequence of distinguished elements is the tuple $\mathbf{x}_0$. We say that a Datalog program P is *connected* if the canonical instance of each rule is connected. We say that a Datalog program P is *non-recursive* if $\mathbf{S}^P_{aux} = \emptyset$.

If P is a Datalog program and I an $\mathbf{S}^P_{in}$-instance, then a *solution* for I with respect to P is an instance J over the schema $\mathbf{S}_{in} \cup \mathbf{S}_{out} \cup \mathbf{S}_{aux}$ such that $I \subseteq J$, and such that all the rules of P are satisfied in J (i.e., whenever the body of a rule is satisfied under a variable assignment, then so is the head). The well-known *chase* procedure provides a method for constructing a solution: given a Datalog program P and an $\mathbf{S}^P_{in}$-instance I, we denote by $\text{chase}_P(I)$ the $\mathbf{S}^P_{in} \cup \mathbf{S}^P_{out} \cup \mathbf{S}^P_{aux}$-instance obtained from I by applying all rules until convergence. More precisely, $\text{chase}_P(I)$ can be defined as the infinite union $\bigcup_{i \geq 0} \text{chase}^i_P(I)$, where $\text{chase}^0_P(I) = I$, and where $\text{chase}^{i+1}_P(I)$ extends $\text{chase}^i_P(I)$ with all facts that can be derived from facts in $\text{chase}^i_P(I)$ using a rule in Σ^P. We refer to [1] for more details.

▶ **Lemma 2.2.** *For all Datalog programs* P *and* $\mathbf{S}^P_{in}$*-instances* I*,* $\text{chase}_P(I)$ *is a solution for* I *with respect to* P*. Moreover, it is the intersection of all solutions for* I *with respect to* P*.*

We denote the $\mathbf{S}^P_{out}$-reduct of $\text{chase}_P(I)$ by $P(I)$. We say that two Datalog programs P, P' with $\mathbf{S}^P_{in} = \mathbf{S}^{P'}_{in}$ and $\mathbf{S}^P_{out} = \mathbf{S}^{P'}_{out}$ are *equivalent* if, for all $\mathbf{S}^P_{in}$-instances I, $P(I) = P'(I)$.

By a *Boolean* Datalog program, we mean a Datalog program P where $\mathbf{S}^P_{out}$ consists of a single zero-ary relation symbol, which is customarily denoted as `Ans`. In such cases, write $P(I) = \mathit{true}$ if $P(I) = \{\texttt{Ans}()\}$ and $P(I) = \mathit{false}$ otherwise (i.e., if $P(I) = \emptyset$).

It is well-known that Datalog programs are monotone with respect to homomorphisms:

▶ **Lemma 2.3.** *Let* P *be any Datalog program, and let* I, I' *be* $\mathbf{S}^P_{in}$*-instances. Every homomorphism* $h : I \to I'$ *yields, when restricted to* $\text{adom}(P(I))$*, a homomorphism from* $P(I)$ *to* $P(I')$*.*

We can think of the above definition of $P(I)$, in terms of the chase, as a bottom-up account of the semantics of a Datalog program. *Unfoldings* (a.k.a. *expansions*) provide a complementary, top-down account. Given a Datalog program P, the set of *derivable rules* of P is the smallest set of rules that (i) contains all rules of P, and (ii) is closed under the operation of substituting occurrences of rule heads by the corresponding rule bodies (renaming variables as necessary). Given a Datalog program P and a relation $R \in \mathbf{S}^P_{out}$, $\text{Unfoldings}(P, R)$ is the set of canonical instances of derivable rules that have R in the rule head and that only have $\mathbf{S}_{in}$-relations in the body. Note that this set is in general infinite.

▶ **Example 2.4.** Let P be the Datalog program consisting of the three rules

$$R(x, y) :- S(x, y) \qquad R(x, x) :- T(x, y) \qquad T(x, y) :- U(x, y), U(y, z)$$

where $\mathbf{S}_{in} = \{U, S\}$, $\mathbf{S}_{out} = \{R\}$, and $\mathbf{S}_{aux} = \{T\}$. Then the derivable rules of P are the rules of P together with the rule $R(x, x) :- U(x, y), U(y, x)$, and $\text{Unfoldings}(P, R)$ consists (up to isomorphism) of the pointed instances $(\{U(a, b), U(b, c)\}, \langle a, a\rangle)$ and $(\{S(a, b)\}, \langle a, b\rangle)$.

▶ **Lemma 2.5** (Cf. [12]). *For all Datalog programs P, instances $I \in \text{Inst}[\mathbf{S}^P_{in}]$, and $\mathbf{S}^P_{out}$-facts $R(\mathbf{a})$ over $\text{adom}(I)$, $R(\mathbf{a}) \in P(I)$ iff, for some $(J, \mathbf{b}) \in \text{Unfoldings}(P, R)$, $(J, \mathbf{b}) \to (I, \mathbf{a})$.*

3 TAM Datalog

TAM Datalog is a fragment of Datalog defined by two requirements: "tree-shaped" and "almost-monadic". We introduce each in isolation first.

Almost-Monadic Datalog Recall that a Datalog program is *monadic* if all relations in $\mathbf{S}_{aux}$ are unary. It is well known that monadic Datalog programs can be expressed in Monadic Second-Order logic (MSO). Formally, by a *k-ary MSO query* over a schema $\mathbf{S}$, we will mean an MSO formula $\phi(x_1, \ldots, x_k)$ over $\mathbf{S}$. We say that a Datalog program $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \mathbf{S}_{aux}, \Sigma)$ together with a designated k-ary relation $R \in \mathbf{S}_{out}$, *defines* an MSO query $\phi_R(\mathbf{x})$ over $\mathbf{S}_{in}$, if for all $\mathbf{S}_{in}$-instances I and $\mathbf{a} \in \text{adom}(I)$, $R(\mathbf{a}) \in P(I)$ iff $I \models \phi_R(\mathbf{a})$. The following is folklore in the database literature (cf. [19] for an explicit proof):

▶ **Theorem 3.1.** *Let P be a monadic Datalog program and $R \in \mathbf{S}^P_{out}$. Then (P, R) defines an MSO query.*

We will now define a weaker restriction, namely that of *almost-monadic* Datalog programs, for which the same holds. These are programs in which every k-ary auxiliary relation has, among its k argument positions, (at most) one specified "*articulation position*", and the syntax of the rules is constrained in such a way that variables occurring in non-articulation positions can only be used to carry information forward, and not to "perform joins". Formally:

▶ **Definition 3.2** (Almost-Monadic Datalog). *An* articulation function*, for a Datalog program P, is a partial function f mapping relations $R \in \mathbf{S}^P_{aux}$ to a number $f(R) \in \{1, \ldots, \text{arity}(R)\}$, which we will call the* articulation position *of R. Each $i \in \{1, \ldots, \text{arity}(R)\}$ other than $f(R)$ is called a* non-articulation position *of R. A Datalog program is* almost-monadic *if there exists an articulation function such that, in every rule, each variable occurring in a non-articulation position of an auxiliary relation in a rule body occurs only once in that rule body, and does not occur in the articulation position of an auxiliary relation in the head.*

Note: the articulation conditions pertain to auxiliary relations and not to output relations.

▶ **Example 3.3.** The Datalog program from Example 1.1 (which outputs all pairs (a, b) for which there is a directed path from a to b) is not monadic but is almost-monadic. The witnessing articulation function assigns to the auxiliary relation `Path` its first position as articulation position. Even if we extend the program with an additional rule $\texttt{Ans}(x, y) :- \texttt{Path}(y, x)$ (so that it computes all pairs (a, b) for which there is a directed path from a to b or from b to a), the resulting program is still almost-monadic. This is because the requirements on the articulation function only pertain to auxiliary relations, not to output relations. On the other hand, if we were to change the rule $\texttt{Path}(x, y) :- \texttt{Edge}(x, z), \texttt{Path}(z, y)$ to $\texttt{Path}(x, y) :- \texttt{Path}(x, z), \texttt{Path}(z, y)$, the program would no longer be almost-monadic (cf. also Example 6.2).

For an example of a Datalog program that is *not* almost-monadic, see Example 3.11 below.

▶ **Proposition 3.4.** *The almost-monadic Datalog program from Example 1.1 is not equivalent to a monadic Datalog program.*

The following result justifies the terminology *almost-monadic*. It shows that almost-monadic Datalog programs can be simulated, in a precise sense, by monadic Datalog programs.

▶ **Theorem 3.5.** *For each almost-monadic Datalog program P and k-ary relation symbol $R \in \mathbf{S}^P_{out}$, there is a Boolean monadic Datalog program P' where $\mathbf{S}^{P'}_{in} = \mathbf{S}^P_{in} \cup \{Q_1, \ldots, Q_k\}$, such that the following are equivalent, for all $\mathbf{S}^P_{in}$-instances I and $a_1, \ldots, a_k \in \text{adom}(I)$:*

1. *$R(a_1, \ldots, a_k) \in P(I)$,*
2. *$P'(I \cup \{Q_1(a_1), \ldots, Q_k(a_k)\}) = true$.*

▶ **Example 3.6.** Let P be the Datalog program from Example 1.1. To satisfy the statement of Theorem 3.5, it suffices to define P' as:

$$\begin{aligned} \texttt{Path'}(x) &:- \texttt{Edge}(x, y), Q_2(y). \\ \texttt{Path'}(x) &:- \texttt{Edge}(x, y), \texttt{Path'}(y). \\ \texttt{Ans}() &:- \texttt{Path'}(x), Q_1(x). \end{aligned}$$

Informally, `Path'`(x) holds if there is a path starting at x that ends at a node satisfying Q_2.

It follows that almost-monadic Datalog is contained in MSO. That is, we have the following analogue of Thm. 3.1 for almost-monadic Datalog programs:

▶ **Corollary 3.7.** *Let P be an almost-monadic Datalog program and $R \in \mathbf{S}^P_{out}$. Then (P, R) defines an MSO query.*

In summary, we have that almost-monadic Datalog forms a strict extension of monadic Datalog that is still contained in MSO. One may be tempted to conjecture that almost-monadic Datalog is expressively complete for the intersection of Datalog and MSO. However, this is not the case. The easiest way to show this, is using the following Lemma, which is interesting in its own right, as it shows that, when the output schema contains only unary relations, almost-monadic Datalog is no more expressive than monadic Datalog:

▶ **Lemma 3.8.** *Let P be any almost-monadic Datalog such that every $R \in \mathbf{S}^P_{out}$ is unary. Then P is equivalent to a monadic Datalog program.*

Using this lemma, we can show:

▶ **Proposition 3.9.** *The unary MSO query "x* lies on a directed R-cycle*" is not definable by an almost-monadic Datalog program.*

▶ Remark 3.10. Prop. 3.9 also shows that almost-monadic Datalog is not closed under composition, because the same query can be expressed as the composition of two almost-monadic Datalog programs, where the first computes the transitive closure R^* of the relation R, and the second program consists of the single non-recursive rule `Ans`(x) :− $R^*(x, x)$. It also shows that almost-monadic Datalog is strictly included in MODEQ (also known as Flag-and-Check), which is another language contained in the intersection of Datalog and MSO [23]. See also [7] for a characterization of the intersection of MSO and Datalog in terms of infinite domain constraint satisfaction problems. As we will soon see, the intersection of *tree-shaped* Datalog and MSO *is* (up to logical equivalence) precisely tree-shaped almost-monadic Datalog.

Tree-shapedness We say that a Datalog program P is *tree-shaped* if the incidence graph of the canonical instance of each rule is acyclic. In particular, since we defined incidence graphs as multigraphs, this implies that no variable occurs twice in the same conjunct in the rule body (but the rule head may contain repeated occurrences of variables). Note that we do not require the incidence graph of the rules to be connected, nor do we make any requirements (say, in the case of binary relations) on the direction of edges. Thus, in tree-shaped Datalog programs, rules such as $T(x) :- R(x,y), S(x,y)$ or $T(x) :- R(x,x)$ are forbidden.

▶ **Example 3.11.** Consider the tree-shaped Datalog program P given by the following two rules (where $\mathbf{S}^P_{in}$ consists of two binary relations, E, F):

$$R(x,y) :- E(x,u), F(u,y) \qquad R(x,y) :- E(x,u), R(u,v), F(v,y) \qquad \texttt{Ans}(x,y) :- R(x,y)$$

Then $\texttt{Ans}^{P(I)}$ contains all pairs (a,b), such that there is a directed path from a to b in I consisting of a number of E-edges followed by an equal number of F-edges.

Observe that P is not TAM Datalog because neither the first position of the relation R qualifies as an articulation position (since v occurs twice in the second rule body) nor the second position (since u occurs twice in the same rule body).

It follows from known facts about MSO (viz. the fact that MSO on words captures the regular languages) that $(P, \texttt{Ans})$ does not define an MSO query. In particular, P is not equivalent to a monadic Datalog program, or even an almost-monadic Datalog program.

▶ **Lemma 3.12.** *Let P be any tree-shaped Datalog program. Then, for each $R \in \mathbf{S}^P_{out}$,* Unfoldings(P,R) *consists of acyclic pointed instances.*

TAM Datalog A *TAM Datalog program* is a tree-shaped, almost-monadic Datalog program. We will give a precise model-theoretic characterization of TAM Datalog in terms of MSO.

We say that an MSO query $\phi(\mathbf{x})$ is *tree-determined* if for each pointed instance $(I,\mathbf{a})$, we have that $I \models \phi(\mathbf{a})$ if and only if there is an acyclic pointed instance $(J,\mathbf{b})$ such that $J \models \phi(\mathbf{b})$ and $(J,\mathbf{b}) \to (I,\mathbf{a})$. Note that J must be finite and that J is not required to be connected.

▶ **Theorem 3.13.** *Let $\phi(x_1,\ldots,x_n)$ be an MSO formula. The following are equivalent:*

1. *ϕ is definable by a TAM Datalog program,*
2. *ϕ is definable by a tree-shaped Datalog program,*
3. *ϕ is tree-determined.*

▶ Remark 3.14. It is worth comparing this to the result in [19] that states that monadic Datalog and MSO have the same expressive power on finite trees. Besides the fact that Thm. 3.13 is a characterization on arbitrary (finite) instances while the result in [19] is restricted to trees, there are a few other important differences: in [19], it is assumed that trees are represented as structures in which the children of each node are ordered; that the signature includes predicates marking the root, leafs, the first child of each node, and the last child of each node; and that each node of the tree is labeled by precisely one of the (other) unary predicates in the signature. These assumptions together imply that every homomorphism between such trees is necessarily an isomorphism, which makes the two results incomparable.

▶ **Corollary 3.15** (TAM Datalog is closed under composition)**.** *For all TAM Datalog programs P_1 and P_2 with $\mathbf{S}^{P_2}_{in} = \mathbf{S}^{P_1}_{out}$, there is a TAM Datalog program $P_3 = (\mathbf{S}^{P_1}_{in}, \mathbf{S}^{P_2}_{out}, \mathbf{S}'_{aux}, \Sigma')$ such that, for all $\mathbf{S}^{P_1}_{in}$-instances I, $P_3(I) = P_2(P_1(I))$.*

We also provide a syntactic normal form for TAM Datalog programs. A TAM Datalog program is *simple* if every rule body contains precisely one occurrence of a relation from $\mathbf{S}_{in}$. For instance the program given in Example 1.1 is a simple TAM Datalog program.

▶ **Theorem 3.16.** *Every (connected) TAM Datalog program can be transformed in polynomial-time into an equivalent (connected) simple TAM Datalog program.*

We make use of this normal form in some of our proofs.

4 Right-Adjoints for TAM Datalog

The notion of adjunction comes from category theory. Although for the most part, we do not assume that the reader has a background in category theory, in order to motivate our definition of generalized right-adjoints for Datalog programs, it is helpful to briefly discuss Datalog programs from a categorical perspective.

Recall that each Datalog program P defines a mapping from $\mathrm{Inst}[\mathbf{S}^P_{in}]$ to $\mathrm{Inst}[\mathbf{S}^P_{out}]$, where $\mathrm{Inst}[\mathbf{S}]$ denotes the set of all $\mathbf{S}$-instances. Recall also that this mapping is monotone with respect to homomorphisms (cf. Lemma 2.3). We view $\mathrm{Inst}[\mathbf{S}^P_{in}]$ and $\mathrm{Inst}[\mathbf{S}^P_{out}]$ as partial (pre)orders, which can consequently be viewed as *thin categories* where the objects are the $\mathbf{S}$-instances and there is an arrow from I to J if there exists a homomorphism $h : I \to J$. The categorical notion of a *functor* is then simply a monotone mapping. In particular, each Datalog program P defines a functor. For functors $F : X \to Y$ and $G : Y \to X$, where X and Y are arbitrary thin categories, it is said that G is a *right-adjoint* for F, and that F is a *left-adjoint* of G, if it holds that $F(I) \to J$ iff $I \to G(J)$.[1]

In this section, we study the existence of right-adjoints for Datalog programs.

▶ **Example 4.1.** Consider the Datalog program $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \emptyset, \Sigma)$, where $\mathbf{S}_{in} = \{R\}$, $\mathbf{S}_{out} = \{S\}$, and Σ consists of the rules $S(x, y) :- R(x, y)$ and $S(x, y) :- R(y, x)$. If we think of an input instance I as a directed graph, $P(I)$ is its *symmetric closure.* For every $\mathbf{S}^P_{out}$-instance J, let $\Omega(J)$ be the $\mathbf{S}^P_{in}$-instance that is the *maximal symmetric sub-instance* of J, that is, $\Omega(J)$ consists of all facts $R(x, y)$ for which it holds that J contains both $S(x, y)$ and $S(y, x)$. It is not hard to see that $P(I) \to J$ iff $I \to \Omega(J)$. Hence, Ω is a right-adjoint of P.

▶ **Example 4.2.** Consider the TAM Datalog program $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \emptyset, \Sigma)$, where $\mathbf{S}_{in} = \{Q_1, Q_2\}$, $\mathbf{S}_{out} = \{Q_3\}$, and Σ consists of the rule $Q_3() :- Q_1(x), Q_2(y)$. This Datalog program does *not* have a right-adjoint in the above sense. Indeed, let J be the empty instance. Then $P(I) \to J$ holds if and only if either I has no Q_1-facts or I has no Q_2-facts, a condition that cannot be equivalently characterized by the existence of a homomorphism from I to any fixed single instance J'. However, it can be shown that $P(I) \to J$ if and only if either $I \to J'_1$ or $I \to J'_2$, where $J'_1 = \{Q_1(a)\}$ and $J'_2 = \{Q_2(a)\}$. If we generalize the notion of right-adjoint by allowing $\Omega(J)$ to be a finite set of instances, then, as we will see later (Thm. 4.5), P *does* admit such a right-adjoint, and the fact that $\Omega(J)$ needs to consist of multiple instances is related to the fact that the program is not connected.

Motivated by the above examples and other considerations that will become clear soon, the precise notion of right-adjoints that we will adopt here is a little more refined:

▶ **Definition 4.3** (Generalized Right-Adjoints). *A* generalized right-adjoint *for a Datalog program P is a function Ω_P that maps each $J \in \mathrm{Inst}[\mathbf{S}^P_{out}]$ to a finite set of pairs (J', ι) where $J' \in \mathrm{Inst}[\mathbf{S}^P_{in}]$ and $\iota : J' \rightharpoonup J$ is a partial function, such that the following holds:*

[1] Note that this coincides with the usual definition of adjoint functors as long as both categories are thin. It is also precisely the notion of right-adjoints for Galois connections over preordered sets. See also Remark 4.11 for why we adopt this "thin" definition of adjoints.

for all $I \in \mathrm{Inst}[\mathbf{S}^P_{in}]$, there is a homomorphism $h : P(I) \to J$ iff there is a homomorphism $h' : I \to J'$ for some $(J', \iota) \in \Omega_P(J)$, and, furthermore, the homomorphism h', respectively h, can be chosen such that the following diagram commutes.[2]

$$\begin{tikzcd}
\mathrm{adom}(P(I)) \arrow[r, "h"] & \mathrm{adom}(J) \\
\mathrm{adom}(I) \arrow[u, harpoon, "id"] \arrow[r, "h'"] & \mathrm{adom}(J') \arrow[u, harpoon, "\iota"']
\end{tikzcd}$$

Here, the "$\to$" arrow refers to homomorphisms, and "$\rightharpoonup$" is used to refer to a partial function. The partial functions ι are needed later on to reason about pointed instances (cf. for instance the proof of Theorem 5.4).

This notion of generalized right-adjoint behaves as one would expect. In particular, if two Datalog programs have generalized right-adjoints, then so does their composition.

▶ **Example 4.4.** Let P be the Datalog program with input schema $\{R_1, R_2\}$ and output schema $\{Q_1, Q_2\}$ and consisting of the rules

$$Q_1(x) :\!-\; R_1(x) \qquad\qquad Q_1(x) :\!-\; R_2(x) \qquad\qquad Q_2(x) :\!-\; R_1(x), R_2(x)$$

This Datalog program indeed has a generalized right-adjoint Ω_P. In fact it has a regular right-adjoint, in the sense that $\Omega_P(J)$ is a singleton for all $J \in \mathrm{Inst}[\mathbf{S}^P_{out}]$, as will follow from Theorem 4.5 below. We will illustrate this with an example. Let J be the $\mathbf{S}^P_{out}$-instance consisting of the single fact $Q_1(a)$. Then a suitable choice for $\Omega_P(J)$ is the singleton set consisting of the pair (J', ι), where J' consists of the facts $R_1(a_1), R_2(a_2)$ and $\iota(a_1) = \iota(a_2) = a$. Indeed, for each $\mathbf{S}^P_{in}$-instance I, $P(I) \to J$ iff $I \to J'$ via homomorphisms that make the diagram in Definition 4.3 commute. Note that, to make the diagram commute, J' must indeed contain facts of the form $R_1(a_1)$ and $R_2(a_2)$ for distinct values a_1, a_2 that are both mapped to a by ι.

Our main result in this section is:

▶ **Theorem 4.5.** *Every TAM Datalog program P has a generalized right-adjoint Ω_P. If P is connected, then $\Omega_P(J)$ is a singleton for all $J \in \mathrm{Inst}[\mathbf{S}^P_{out}]$. Moreover, $\Omega_P(J)$ is computable from J and P in 2Exptime, and in ExpTime whenever the arity of P is bounded.*

Proof. We first consider the special case of connected programs. We may assume without loss of generality that P is simple. This means that every rule is of the following form:

$$R_0(\mathbf{x}_0) :\!-\; E(\mathbf{y}), R_1(\mathbf{x}_1), \ldots, R_m(\mathbf{x}_m) \qquad \text{(Eq. 2)}$$

$$R(\mathbf{x}) :\!-\; E(\mathbf{y}), R_1(\mathbf{x}_1), \ldots, R_m(\mathbf{x}_m) \qquad \text{(Eq. 3)}$$

where E is an input relation, each R_i is an auxiliary relation, and R is an output relation.

To simplify the exposition below, we introduce some further notation. For each atom $R_i(\mathbf{x}_i)$ as in the above rule types, we will denote by $p_i \in \{1, \ldots, n\}$ (with $n = \mathrm{arity}(E)$) the unique number such that y_{p_i} is equal to the articulated variable in $R_i(\mathbf{x}_i)$. It indeed follows from the definition of TAM Datalog and the assumed connectedness and simplicity of P that such an index exists and is unique.

We construct an $\mathbf{S}_{in}$-instance J' consisting of all facts $E((b_1, X_1), \ldots, (b_n, X_n))$ where

[2] By this we, we mean that, for all $a \in \mathrm{adom}(I)$, either $h(a) = \iota(h'(a))$ or both are undefined.

1. Each b_i is an element of $\text{adom}(J) \cup \{\bot\}$ and X_i is a set of $\mathbf{S}_{aux}$-facts over $\text{adom}(J) \cup \{\bot\}$ (not necessarily facts of J) in which b_i occurs in articulation position;
2. For each rule of the form (1) above and for each map $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \to \text{adom}(J) \cup \{\bot\}$, if for each $1 \leq i \leq m$, $R_i(g(\mathbf{x}_i)) \in X_{p_i}$ then $R_0(g(\mathbf{x}_0)) \in X_{p_0}$; and
3. For each rule of the form (2) above and for each map $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \to \text{adom}(J) \cup \{\bot\}$, if for each $1 \leq i \leq m$, $R_i(g(\mathbf{x}_i)) \in X_{p_i}$ then $R(g(\mathbf{x}))$ is a fact of J.

Note that the total number of possible facts $E((b_1, X_1), \dots, (b_n, X_n))$ in J' is double exponential in the combined size of P and J, and is exponential in J if the arity of P is bounded.

Let ι be the natural projection from J' to J, mapping all elements of the form (a, X) to a (and undefined on elements of the form $(\bot, X)$).

▷ Claim. For all $\mathbf{S}_{in}$-instances I, $P(I) \to J$ iff $I \to J'$. Moreover, the witnessing homomorphisms can be constructed so that the following diagram commutes:

$$\begin{tikzcd}
P(I) \arrow[r] & J \\
I \arrow[u, "id"] \arrow[r] & J' \arrow[u, "\iota"']
\end{tikzcd}$$

Proof. [$\Rightarrow$] Let $h : P(I) \to J$. Recall that we denote by $\text{chase}_P(I)$ the $\mathbf{S}_{in} \cup \mathbf{S}_{out} \cup \mathbf{S}_{aux}$-instance that is the chase of I (and of which I and $P(I)$ are the $\mathbf{S}_{in}$-reduct and $\mathbf{S}_{out}$-reduct, respectively). We extend h to the entire active domain of $\text{chase}_P(I)$ by sending every element a that is not in $\text{adom}(P(I))$ to a fresh value $\bot$. With a slight abuse of notation, in what follows, we denote by h the extended map from $\text{adom}(\text{chase}_P(I))$ to $\text{adom}(J) \cup \{\bot\}$. For each $a \in \text{adom}(\text{chase}_P(I))$, let F_a be the set of all $\mathbf{S}_{aux}$-facts of $\text{chase}_P(I)$ in which a occurs in articulation position. We define $h'(a) = (h(a), h(F_a))$.

We claim that h' is a homomorphism from I to J'. Let $E(a_1, \dots, a_n)$ be any fact of I. We must show that the fact $E((h(a_1), h(F_{a_1})), \dots, (h(a_n), h(F_{a_n})))$ belongs to J'. That is, we must show that conditions 1–3 hold.

Clearly, the first requirement is satisfied, namely, $h(X_{a_i})$ consists of facts in which $h(a_i)$ occurs in articulation position.

To see that the second requirement holds, consider a rule of form (1) and any map $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \to \text{adom}(J)$, such that, for each $1 \leq i \leq m$, $R_i(g(\mathbf{x}_i)) \in X_{p_i}$. By construction, this means that each fact $R_i(g(\mathbf{x}_i))$ is the h-image of a fact $R_i(\mathbf{b}_i)$ in $\text{chase}_P(I)$. Now consider the map $\tilde{g} : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \to \text{adom}(I)$ defined by $\tilde{g}(\mathbf{y}) = (a_1 \dots, a_n)$ and $\tilde{g}(\mathbf{x}_i) = \mathbf{b}_i$ for $i \leq i \leq m$. Note that $\tilde{g}$ is a well-defined function. Indeed, for every $i \leq i \leq m$ and every $z \in R(\mathbf{x}_i)$, if z occurs more than once in the rule, then z must necessarily be the variable that appears in the articulation position p_i of $R(\mathbf{x}_i)$. It follows that z occurs in $\mathbf{y}$ at position p_i, and, hence, necessarily, $R_i(g(\mathbf{x}_i))$ belongs to the h-image of $X_{a_{p_i}}$, and, hence, $\tilde{g}(z) = a_{p_i}$.

Since $\text{chase}_P(I)$ is closed under the rules of P, we may conclude that $R_0(\tilde{g}(\mathbf{x}_0))$ belongs to $\text{chase}_P(I)$. Let z be the variable occurring in articulation position in $R_0(\mathbf{x}_0)$. Recall that z occurs in $\mathbf{y}$ at position p_0, and $\tilde{g}(z) = a_{p_0}$. Then $R_0(\tilde{g}(\mathbf{x}_0)) \in F_{a_{p_0}}$, and hence, $h(F_{a_{p_0}})$ contains $R_0(h \circ \tilde{g}(\mathbf{x}_0))$. Note that by definition, $h \circ \tilde{g} = g$. In particular, $R_0(h \circ \tilde{g}(\mathbf{x}_0)) = R_0(g(\mathbf{x}_0))$. Therefore we have that $R_0(g(\mathbf{x}_0)) \in h(F_{a_{p_0}})$, and we are done.

To see that the third requirement holds, consider a rule of form (2) and any map $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \to \text{adom}(J)$, such that, for each $1 \leq i \leq m$, $R_i(g(\mathbf{x}_i)) \in X_{p_i}$. By exactly the same reasoning as before, an h-preimage of the rule head $R(g(\mathbf{x}))$ belongs to $\text{chase}_P(I)$. Hence, it belongs to $P(I)$, therefore, $R(g(\mathbf{x}))$ is a fact of J.

It is also clear from the construction that $h \circ id = \iota \circ h'$, where id is the identity function on $\mathrm{adom}(I) \cap \mathrm{adom}(P(I))$. That is, the diagram commutes.

[$\Leftarrow$] Conversely, let $h : I \to J'$. Note that $\mathrm{adom}(\mathrm{chase}_P(I)) = \mathrm{adom}(I)$, and hence we $h(a)$ is well-defined for all $a \in \mathrm{adom}(\mathrm{chase}_P(I))$. Let $h' : \mathrm{adom}(I) \to \mathrm{adom}(J)$ be the map such that $h'(a) = b$ whenever $h(a) = (b, X)$.

▷ Subclaim 1. For all $a \in \mathrm{adom}(\mathrm{chase}_P(I))$, if $h(a) = (b, X)$, then the h'-image of every $\mathbf{S}_{aux}$-fact of $\mathrm{chase}_P(I)$ in which a occurs in articulation position belongs to X.

▷ Subclaim 2. h' is a homomorphism from $P(I)$ to J.

Subclaim 1 can be proved by induction on the derivation length of the fact in question.

To prove subclaim 2, let $R(\mathbf{a})$ be an $\mathbf{S}_{out}$-fact belonging to $P(I)$. Its derivation must use a rule of the form (2) above, using an assignment g (where $g(\mathbf{x}) = \mathbf{a}$). By Subclaim 1, we have that that $R_i(h'(g(\mathbf{x}_i)))$ belongs to $h(g(y_{p_i}))_2$, for y_{p_i} the articulated variable in $\mathbf{x}_i$. Furthermore, $E(g(\mathbf{y}))$ holds in I, and hence $E(h(g(\mathbf{y})))$ holds in J'. By construction of J', this means that the $R(h'(g(\mathbf{x})))$, that is, $R(h'(\mathbf{a}))$, belongs to J. This concludes the proof for the case of *connected* TAM Datalog programs.

It is also clear from the construction that $\iota \circ h = h' \circ id$, where id is the identity function on $\mathrm{adom}(I) \cap \mathrm{adom}(P(I))$. That is, the diagram commutes. ◁

Finally, we show how to handle non-connected TAM Datalog programs. Let P be a non-connected TAM Datalog program. Let P' be obtained from P by adding a fresh binary input-relation S, and using this relation to make every every rule connected in some arbitrary way (more precisely, whenever the incidence graph of a rule body has multiple connected component, we add S-atoms to the body connecting these components while preserving tree-shapedness and almost-monadicity. For every input instance I, we denote by $\widehat{I}$ the $\mathbf{S}_{in} \cup \{S\}$-instance extending I with all facts of the form $S(a, b)$ for $a, b \in \mathrm{adom}(I)$. Furthermore, given an instance J' over the schema $\mathbf{S}_{in} \cup \{S\}$, by an "$S$-component" of J' we will mean the $\mathbf{S}_{in}$-retract of a fully S-connected sub-instance of J'. Clearly, if J is an $\mathbf{S}_{in}$-instance and J' is a $\mathbf{S}_{in} \cup \{S\}$-instance, then $\widehat{J} \to J'$ iff $J \to J''$ for some S-component J'' of J'. Now we simply define $\Omega_P(J)$ to be the set of all S-components of instance in $\Omega_{P'}(J)$. Then we have: $P(I) \to J$ iff $P'(\widehat{I}) \to J$ iff $\widehat{I} \to \Omega_{P'}(J)$ iff $I \to J'$ for some $J' \in \Omega_P(J)$.

As a side remark, we mention that there is another way present the final argument where we lift the connected case to the general case: we can view the function that sends I to $\widehat{I}$ as a functor that itself has a generalized right-adjoint (sending I to its S-connected components). Thus, we can argue by composition of adjoints. ◀

We note that the proof of thm:tam-adjoint makes crucial use of both the tree-shapedness and the almost-monadicity of the Datalog program. Indeed, both properties are important for the existence of generalized right-adjoints as the following two propositions show:

▶ **Proposition 4.6.** *The tree-shaped Datalog program P in Example 3.11 (which is not almost-monadic) does not admit a generalized right-adjoint.*

Proof. Assume towards a contradiction that P has a generalized right-adjoint Ω_P. Let J be the two-element $\{R\}$-instance consisting of the facts $R(0, 1)$ and $R(1, 0)$.

For $n \geq 1$, let C_n be the $\{E, F\}$-instance consisting of the facts $E(v_0, v_1), \ldots, E(v_{n-1}, v_n)$, $E(v_n, v_0)$, that is, the directed E-cycle of length n. Trivially, $P(C_n) \to J$ for all $n \geq 1$. Therefore, for each $n \geq 1$, we have $C_n \to J'$ for some $J' \in \Omega_P(J)$. For every $J' \in \Omega_P(J)$ and for each element b of J', let us define $n_{J',b}$ to be an arbitrarily chosen value such that

$(C_n, v_0) \to (J', b)$, or undefined, if no such value exists. It follows from our earlier observation that $n_{J',b}$ is defined for at least one pair (J', b) with $J' \in \Omega_P(J)$. Let m be a common multiple of all defined $n_{J',b}$'s.

For each pair of positive integers $e \leq f$, let $I_{e,f}$ be the $\{E, F\}$-instance depicted as follows:

$$u_0 \xrightarrow{E} v_0 \xrightarrow{F} u_1 \xrightarrow{E} v_1 \xrightarrow{F} u_2 \xrightarrow{\substack{\text{sequence of}\\ e\ E\text{-edges}}} z \xrightarrow{\substack{\text{sequence of}\\ f\ F\text{-edges}}} u_0$$

▷ Claim 1. For all $1 \leq e \leq f$, the following are equivalent:

1. $I_{e,f} \to J'$ for some $J' \in \Omega_P(J)$
2. $e \neq f$.

Proof. If $e = f$ then $P(I_{e,f})$ contains an R-cycle of odd length, viz. $u_0 \xrightarrow{R} u_1 \xrightarrow{R} u_2 \xrightarrow{R} u_0$, and therefore $P(I_{e,f}) \not\to J$. Hence, $I_{e,f} \not\to J'$ for all $J' \in \Omega_P(J)$. On the other hand, if $e < f$, then $P(I_{e,f})$ is a disjoint union of R-paths, and, clearly, $P(I_{e,f}) \to J$. Therefore, $I_{e,f} \to J'$ for some $J' \in \Omega_P(J)$. ◁

Now, let e be larger than the universe of all instances in $\Omega_P(J)$ and let $f = e + m$. By Claim 1, there is a homomorphism $h : I_{e,f} \to J'$ for some $J' \in \Omega_P(J)$. We will show that h can be extended to a homomorphism $h' : I_{f,f} \to J'$, which contradicts Claim 1. Let

$$u_2 = x_0 \xrightarrow{E} x_1 \cdots \xrightarrow{E} x_e = z$$

be the sub-instance of $I_{e,f}$ consisting of the E-edges joining u_2 and z. Similarly, let

$$u_2 = x_0 \xrightarrow{E} x_1 \cdots \xrightarrow{E} x_e \xrightarrow{E} x_{e+1} \ldots \xrightarrow{E} x_f = z$$

be the sub-instance of $I_{f,f}$ consisting of the E-edges joining u_2 and z. Recall that $f = e + m$. Since e is larger than the domain size of J', it must be the case that $h(x_i) = h(x_j) = b$ for some $i < j \leq e$, and for some element b of J'. This means that b lies on a directed E-cycle in J', and hence, in particular, it lies on a directed E-cycle of length m, say, $b = b_0 \xrightarrow{E} b_1 \cdots \xrightarrow{E} b_m = b$. The mapping $h' : I_{f,f} \to J'$ can be constructed simply by extending h and mapping x_{e+i} to b_i for $1 \leq i \leq m$. ◀

▶ **Proposition 4.7.** *The monadic Datalog program given by the single rule* $\texttt{Ans}(x)$:– $E(x, x)$ *does not admit a generalized right-adjoint.*

Proof. Let P be the Boolean Datalog program in question. Note that Unfoldings(P, `Ans`) consists of a pointed structure that is c-acyclic but not acyclic. It does not admit a generalized right-adjoint: let J be the empty $\mathbf{S}^P_{out}$-instance, and suppose for the sake of contradiction that there is a finite set $\{J_1, \ldots, J_n\}$ such that, for all $\mathbf{S}^P_{in}$-instances I, $P(I) \to J$ iff $I \to J_i$ for some $i \leq n$. Let I_c be the instance consisting of a single reflexive `Ans`-edge of the form $\texttt{Ans}(a, a)$. Clearly, $P(I_c) \not\to J$, and therefore, $I_c \not\to J_i$. That is, $J_1, \ldots, J_n$ do not contain a reflexive `Ans`-edge. Next, let I_n be the instance that is an (irreflexive) `Ans`-clique of size n, where n is any number greater than the size of each J_i. Then, $I_n \not\to J_i$ (because if there was a homomorphism, J_i would contain a reflexive `Ans`-edge), but, trivially, $P(I_n) \to J$. ◀

In the special case of Boolean non-recursive programs, there is a converse to Thm. 4.5:

▶ **Theorem 4.8.** *A Boolean non-recursive Datalog programs has a generalized right-adjoint iff it is equivalent to a TAM Datalog program.*

This follows from results that we will prove in Section 5 (specifically, Thm. 5.4 together with Thm. 5.3). It also follows from a known result about Pultr functors, namely [17, Theorem 2.5], since Boolean non-recursive Datalog programs define Pultr functors.

▶ Remark 4.9. Thm. 4.8 does not hold for *recursive* Boolean Datalog programs. Indeed, consider the Boolean Datalog program with input schema $\{R\}$ consisting of the rules

$$\begin{array}{lll} \texttt{Ans}() & :- & \texttt{OddLengthPath}(x,x) \\ \texttt{OddLengthPath}(x,y) & :- & R(x,y) \\ \texttt{OddLengthPath}(x,y) & :- & R(x,z), R(z,u), \texttt{OddLengthPath}(u,y) \end{array}$$

It follows from well-known results in the CSP literature, (combined with Thm. 5.7 below), that P admits a generalized right-adjoint and that P is not equivalent to a monadic Datalog program. It follows by Lem. 3.8 that P is not equivalent to a TAM Datalog program.

▶ Remark 4.10. We note that the right adjoint $\Omega_P(\cdot)$ of a TAM Datalog program P is in general not definable by a Datalog program. This follows trivially from the fact that $\Omega_P(J)$ might consist of more than one structure if P is non connected and, in addition, It is not definable by a Datalog program even when P is connected as, in general, the domains of J and $\Omega_P(J)$ do not need to be related. For instance, Example 4.4 contains an example where the domain of $\Omega_P(J)$ is necessarily larger than that of J.

▶ Remark 4.11. Our definition of right-adjoints treats Datalog programs as functors in a flat category, while Lemma 2.3 naturally allows us to view Datalog programs as functors even in the non-flat category of instances and homomorphisms. This natually raises a question of which functors between the 'non-flat' categories allow right adjoints. This question has been answered by Pultr [22] up to some technical details, who described pairs of adjoint functors between categories of relational structures. Using either Pultr's description, or by simple categorical methods, it can be seen that a Datalog program will rarely allow a proper right adjoint.

▶ Remark 4.12. While we are specifically interested in right-adjoints in this paper, one may also wonder what it means for a Datalog program to admit a (generalized) left-adjoint. Generalized left-adjoints for Datalog programs are closely related to query rewritings, as studied in the literature on data integration and data exchange. A Datalog program P has a generalized left-adjoint iff P is equivalent to a non-recursive Datalog program. Indeed, if P has a generalized left-adjoint Θ, then, for each $R \in \mathbf{S}^P_{out}$, the $\mathbf{S}^P_{in}$-instances in $\Theta(\{R(a_1,\ldots,a_n)\})$ correspond to the members of Unfoldings(P,R) (cf. [17, 13]).

5 Right Adjoints and Homomorphism Dualities

For any set of instances X, let $X^\uparrow = \{A \mid B \to A \text{ for some } B \in X\}$, and let $X^\downarrow = \{A \mid A \to B \text{ for some } B \in X\}$. A *homomorphism duality* is a pair of sets of instances (F, D), such that $F^\uparrow$ is the complement of $D^\downarrow$. The same definition extends naturally to pointed instances. By a *finite* homomorphism duality, we mean a homomorphism duality (F, D) where F and D are finite sets. By a *tree duality*, we mean a homomorphism duality (F, D) where F is a (possibly infinite) set of (not-necessarily-connected) acyclic instances, and D is finite.

The study of such dualities originated in combinatorics (see [20]) motivated by its links to the structure of the homomorphism partial order, and the complexity of deciding the existence of homomorphism between graphs and, more generally, relational structures (a.k.a. constraint satisfaction problems or CSPs). Indeed, dualities have played an important role in the study of CSPs. In particular, it was shown [3] that the CSPs definable in FO are precisely those whose template is the right-hand side of a finite duality. In a similar vein, the CSPs solvable by the well-known *arc-consistency* algorithm are precisely those whose template is the right-hand side of a tree duality. More generally, the CSPs solvable by local consistency methods are those whose template is the right-hand side of a homomorphism duality whose left-hand side consists of instances of bounded treewidth. See [8] for a survey on the connections between duality and consistency algorithms. In database theory, homomorphism dualities are used in the study of the unique characterizability and exact learnability of schema mappings and database queries [2, 9], closed-world rewritings of open-world queries [6], and extremal fitting algorithms [10].

▶ **Example 5.1.** Let $\mathbf{S} = \{R\}$, where R is a binary relation symbol, and let $n \geq 1$. Let L_n be the finite linear order of length n, and let P_{n+1} be the directed path of length $n+1$. Then $(\{P_{n+1}\}, \{L_n\})$ is a finite homomorphism duality.

▶ **Example 5.2.** Let $\mathbf{S} = \{P_0, P_1, E\}$, where P_0 and P_1 are unary and E is binary, and consider the two-element $\mathbf{S}$-instance $I = \{P_0(0), P_1(1), E(0,0), E(1,1)\}$ (without distinguished elements). For all $\mathbf{S}$-instances J, $J \to I$ holds if and only if no connected component of J contains both a P_0-fact and a P_1-fact. This can be expressed in the form of a tree duality: let F be the set of all (acyclic) instances consisting of an oriented path that connects a P_0-node to a P_1-node (where, by an *oriented path* we mean a path $a_1, a_2, \ldots, a_n$ where, for each $1 \leq i < n$, either $E(a_i, a_{i+1})$ or $E(a_{i+1}, a_i)$). Then $(F, \{I\})$ is a homomorphism duality.

▶ **Theorem 5.3** ([16, 9]). *Fix a schema $\mathbf{S}$ and $k \geq 0$. Let F be any finite set of pairwise homomorphically incomparable k-ary pointed instances over $\mathbf{S}$. The following are equivalent:*

1. *There is a finite set of k-ary pointed instances D over $\mathbf{S}$ such that (F, D) is a homomorphism duality.*
2. *Each pointed instance in F is homomorphically equivalent to a c-acyclic pointed instance.*

Moreover (for fixed $\mathbf{S}$ and k), given a set F of c-acyclic pointed instances, such a set D can be computed in ExpTime.[3]

The following theorem establishes a close relationship between generalized right-adjoints and homomorphism dualities:[4]

▶ **Theorem 5.4.** *Let P be any Datalog program that has a generalized right-adjoint. Then, for each $R \in \mathbf{S}^P_{out}$, there is a finite set of pointed $\mathbf{S}_{in}$-instances D such that* $(\text{Unfoldings}(P, R), D)$ *is a homomorphism duality.*

Proof. We may assume without loss of generality that $\mathbf{S}^P_{out} = \{R\}$. Let J be the $\mathbf{S}^P_{out}$-instance with $\text{adom}(J) = \{b_1, \ldots, b_k, c\}$ (for $k = \text{arity}(R)$) containing all R-facts over $\text{adom}(J)$ except $R(b_1, \ldots, b_k)$. Let $D = \{(J', \mathbf{b}') \mid (J', \iota) \in \Omega_P(J), \mathbf{b}' \in \text{adom}(J')^k, \iota(\mathbf{b}') = \mathbf{b}\}$, where $\mathbf{b} = b_1, \ldots, b_k$. We claim that $(\text{Unfoldings}(P, R), D)$ is a homomorphism duality. Let $(C, \mathbf{c})$ be any $\mathbf{S}_{in}$-instance with k distinguished elements. Then an instance in $\text{Unfoldings}(P, R)$ homomorphically maps to $(C, \mathbf{c})$ iff $R(\mathbf{c}) \in P(C)$ iff $(P(C), \mathbf{c}) \not\to (J, \mathbf{b})$ iff (by the adjoint property) $(C, \mathbf{c}) \not\to (J', \mathbf{b}')$ for all $(J', \iota) \in \Omega_P(J)$ and $\mathbf{b}'$ with $\iota(\mathbf{b}') = \mathbf{b}$. ◀

Thm. 5.4 shows that generalized right-adjoints can be used to construct duals. This approach was first used in [17] and [13], where right-adjoints of Pultr functors are applied to derive the dual of a tree. Thm. 5.4 can be viewed as extending results in [13] to a more general class of functors defined by recursive Datalog programs.

[3] The ExpTime bound is not explicitly stated in [9] but follows from results in that paper.

[4] Recall that $\text{Unfoldings}(P, R)$ can be viewed as an infinite union of (canonical instances of) conjunctive queries that defines the output relation R (cf. Lemma 2.5).

For TAM Datalog programs P, the proof of Thm. 5.4 yields a ExpTime algorithm to compute D from (P, R) provided the arity of the relations in P is bounded. Since $\text{Unfoldings}(P, R)$ consists of acyclic instances whenever P is a TAM Datalog program, this gives us a systematic way of constructing tree-dualities. In fact, every tree-duality can be obtained in this way:

▶ **Corollary 5.5.** *Let F be any set of acyclic pointed instances. The following are equivalent:*
1. *There is a finite set of pointed instances D such that (F, D) is a homomorphism duality*
2. *$F^{\uparrow} = \text{Unfoldings}(P, R)^{\uparrow}$ for some TAM Datalog program P and $R \in \mathbf{S}^P_{out}$.*

Proof. From 1 to 2: It is well known that, for any finite set of pointed instances D, there is an MSO formula ϕ that defines $D^{\downarrow}$. Hence, by duality, $\neg\phi$ defines $F^{\uparrow}$. Furthermore, the fact that F consists of acyclic pointed instances implies that $\neg\phi$ is tree-determined. Therefore, the direction 1 to 2 follows from Thm. 3.13. The direction from 2 to 1 follows from Thm. 5.4. ◀

It is possible to strengthen Corollary 5.5 by showing that the above conditions (1) and (2) are, in turn, equivalent to the fact that $F^{\uparrow} = G^{\uparrow}$ for some regular set G of acyclic queries (where "regular" needs to be defined in a suitable way, as in [15]). This follows from the fact that Thm. 3.13 uses tree-automata as an intermediate step in the proof. We note that the special case of this equivalence for Boolean CQs over digraphs was proven in [15].

Thm. 5.4 also implies that every finite set of acyclic pointed instances F is the left-hand side of a finite homomorphism duality: it suffices to let P be the TAM Datalog program containing a single non-recursive rule for each $(I, \mathbf{a}) \in F$, whose canonical instance is $(I, \mathbf{a})$. Then, the unfoldings of P are, up to isomorphism, precisely the pointed instances in F. It follows from Thm. 5.4 that there is a finite set D such that (F, D) is a homomorphism duality. This provides an alternative proof of the characterization of left-hand sides of finite dualities given in [16] (i.e., the special case of Thm. 5.3 for structures without distinguished elements).

▶ Remark 5.6. In light of Thm. 5.3, it is natural to ask whether the above "dualities through adjoints" technique can be used to construct a finite homomorphism duality for any c-acyclic pointed instance. Prop. 4.7 shows that this is not possible. Note that the canonical instance of the rule of the program in Prop. 4.7 is $(\{E(a, a)\}, a)$, which is c-acyclic (but not acyclic).

For Boolean Datalog programs, the relationship between adjoints and dualities is tighter:

▶ **Theorem 5.7.** *For Boolean Datalog programs P, the following are equivalent:*
1. *P admits a generalized right-adjoint,*
2. *There is a finite set of pointed $\mathbf{S}^P_{in}$-instances D such that $(\text{Unfoldings}(P, \texttt{Ans}), D)$ is a homomorphism duality.*

Proof. For every $\mathbf{S}^P_{in}$-instance I, $P(I)$ is either the empty instance, which we may denote as J_0 or the instance consisting of the zero-ary fact $\texttt{Ans}()$, which we may denote as J_1. Let $\Omega_P(J_0)$ be the set of all pairs (J', ι) with $J' \in D$ and ι the empty partial function; and let $\Omega_P(J_1) = \{(J', \iota)\}$ where J' is a single-element fully-connected $\mathbf{S}^P_{in}$-instance and ι is the empty partial function. It is easy to see that Ω_P is then a generalized right-adjoint for P. ◀

6 An Application: Generating Data Examples for Database Queries

We will now show-case one application of our results, which is concerned with the problem of generating data examples for database queries. A *data example* for a database query, informally, consists of a database instance I together with information about the output of the query when evaluated on I. Data examples can be a helpful tool in query debugging, query refinement, interactive query specification, and query learning. In each of these settings, the question naturally comes up as to whether, for a given database query q, there exists a finite collection of data examples, such that, modulo logical equivalence, q is the *only* query (within some given class of queries) that fits the data examples. When this happens, we say that the collection of data examples in question *uniquely characterizes* the query q.

It was shown in [16, 9] that every "c-acyclic" union of conjunctive queries (UCQ) is indeed uniquely characterized by a finite collection of data examples. In fact, a UCQ is uniquely characterizable by a finite collection of data examples, if and only if it is equivalent to a c-acyclic UCQ. While this gives a precise answer to the question of unique characterizability, it can be cumbersome to use in practice. One of the reasons for this is that the data examples in question tend to look unnatural to a user. In particular, the existing algorithms for constructing data examples do not take integrity constraints into consideration. We will show here that the aforementioned results from [16, 9] can be adapted to the setting with integrity constraints, in such a way that all generated data examples satisfy the integrity constraints, provided that the integrity constraints are from a suitable, well-behaved class.

We will do this in three steps. First, we propose a suitable class of integrity constraints. Second, we study the existence of homomorphism dualities relative to a set of integrity constraints. Finally, we use this to construct uniquely characterizing examples for c-acyclic UCQs.

6.1 Tame sets of full TGDs

One of the most important classes of integrity constraints in databases is the class of *tuple-generating dependencies (TGDs).* The results we will present will be concerned with a subclass of TGDs called *full TGDs.* A full TGD is a TGD without existential quantifiers. More precisely, a full TGD is a first-order sentence of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \to \psi(\mathbf{x}))$, where $\phi(\mathbf{x})$ and $\psi(\mathbf{x})$ are conjunctions of relational atomic formulas, and each variable in $\mathbf{x}$ occurs in ϕ.

Every finite set of full TGDs naturally gives rise to a Datalog program. More precisely, for any set Σ of full TGDs over a schema $\mathbf{S}$, we will denote by P_Σ the Datalog program with $\mathbf{S}^P_{in} = \{R_{in} \mid R \in \mathbf{S}\}$, $\mathbf{S}^P_{out} = \{R_{out} \mid R \in \mathbf{S}\}$, and $\mathbf{S}^P_{aux} = \mathbf{S}$, consisting of the full TGDs in Σ as Datalog rules (where $\forall \mathbf{x}(\phi(\mathbf{x}) \to \psi(\mathbf{x}))$ becomes $\psi(\mathbf{x}) :- \phi(\mathbf{x})$), plus the "copy constraints" $R(\mathbf{x}) :- R_{in}(\mathbf{x})$ and $R_{out}(\mathbf{x}) :- R(\mathbf{x})$ for each $R \in \mathbf{S}$.

Although the input and output schemas of P_Σ are renamings of $\mathbf{S}$, we will write $P_\Sigma(I)$ even when I is an $\mathbf{S}$-instance instead of an $\mathbf{S}_{in}$, with the understanding that relation symbols are renamed in the obvious way; and similarly, we will freely treat the $\mathbf{S}_{out}$-instance $P_\Sigma(I)$ as an $\mathbf{S}$-instance. The Datalog program P_Σ then "captures" Σ in the following sense:

▶ **Lemma 6.1.** *Let Σ be any finite set of full TGDs. For all instances I, $P_\Sigma(I)$ is the unique minimal instance J with $I \subseteq J$ such that $J \models \Sigma$. In particular, $P_\Sigma(I) = I$ when $I \models \Sigma$.*

We say that a finite set Σ of full TGDs is *tame* (or, TAM-equivalent) if P_Σ is equivalent to a TAM Datalog program. A full TGD is *monadic* (*tree-shaped*) if the associated Datalog program P_Σ (where Σ consists of the full TGD in question) is monadic (resp. tree shaped).

▶ **Example 6.2.** The following sets of full TGDs are tame:

- $\Sigma_1 = \{\forall xyz(R(x,y) \wedge R(y,z) \to R(x,z))\}$. To see that P_{Σ_1} has a generalized right-adjoint, observe that it consists of the rules depicted on the left:

$$\begin{array}{lcl}
R(x,y) & :- & R_{in}(x,y) \\
R(x,z) & :- & R(x,y), R(y,z) \\
R_{out}(x,y) & :- & R(x,y)
\end{array}
\qquad\qquad
\begin{array}{lcl}
R(x,y) & :- & R_{in}(x,y) \\
R(x,z) & :- & R(x,y), R_{in}(y,z) \\
R_{out}(x,y) & :- & R(x,y)
\end{array}$$

P_{Σ_1} is not a TAM Datalog program. However, it is equivalent to the program P' consisting of the rules depicted on the right. Note how we have replaced one occurrence of R by R_{in}. The equivalence of P_{Σ_1} and P' is easy to show. Furthermore, P' is a TAM Datalog program (where the articulation position of R is the second position).

- $\Sigma_2 = \{\forall xyzu(R(x,y) \wedge R(y,z) \wedge R(z,u) \rightarrow R(x,u)), \forall xy(R(x,y) \rightarrow R(y,x))\}$. Although P_{Σ_4} is not a TAM Datalog program, it can be rewritten as one, using the same strategy as for Σ_1. We will return to this example later, in Remark 6.6.
- Every finite set Σ of monadic tree-shaped TGDs. Indeed, P_Σ is a TAM Datalog program.

▶ Remark 6.3. The above example involves adhoc arguments. We leave it as an open problem to define a large syntactic class of (sets of) TGDs that have a generalized right-adjoint, which includes Σ_1. Thm. 4.5 with Thm. 3.13 does imply that, for finite sets of tree-shaped TGDs Σ, if P_Σ is MSO-definable then Σ has a generalized right-adjoint.

Our main result in this section, namely Thm. 6.4, will apply to tame sets of full TGDs. The general strategy we develop here, however, can be extended to a larger class of integrity constraints that includes inclusion dependencies, as we will discuss in the conclusion section. This is beyond the scope of the present paper as it requires considering ∃Datalog programs (i.e., Datalog programs where existential quantifiers are allowed in rule heads).

6.2 Homomorphism dualities within restricted categories

Let $\mathcal{C}$ be a class of instances over some schema (e.g., the class of transitive digraphs). We say that a pair (F, D), with $F, D \subseteq \mathcal{C}$, is a *homomorphism duality within* $\mathcal{C}$ if $F^\uparrow \cap \mathcal{C}$ is the complement of $D^\downarrow \cap \mathcal{C}$ relative to $\mathcal{C}$. In what follows we will also speak of homomorpism dualities *with respect to a theory* Σ. By this, we mean homomorphism dualities w.r.t. the class of instances defined by Σ. The next result shows how to obtain finite homomorphism dualities within $\mathcal{C}$, for classes $\mathcal{C}$ that are definable by a tame set of full TGDs.

▶ **Theorem 6.4.** *Let Σ be a tame set of full TGDs. Let F be any finite set of pointed instances. If each member of F is of the form $(P_\Sigma(A), \mathbf{a})$ for some c-acyclic pointed instance $(A, \mathbf{a})$, then F is the left-hand side of a finite duality w.r.t. Σ.*

Proof. Let F' be the finite set of c-acyclic pointed instances such that $F = \{(P_\Sigma(I), \mathbf{a}) \mid (I, \mathbf{a}) \in F'\}$. Let D be the finite set such that (F', D) is a homomorphism duality, given by Thm. 5.3. Let $D' = \{(P_\Sigma(B'), \mathbf{b}') \mid (B, \mathbf{b}) \in D, (B', \iota) \in \Omega_{P_\Sigma}(B), \iota(\mathbf{b}') = \mathbf{b}\}$. Note that D' consists of pointed instances satisfying Σ. We will show that (F, D') is a homomorphism duality w.r.t. Σ. Let $(C, \mathbf{c})$ be a pointed instance with $C \models \Sigma$. Then:

$$(C, \mathbf{c}) \in F^\uparrow \quad \Leftrightarrow \quad (C, \mathbf{c}) \in F'^\uparrow \quad \Leftrightarrow \quad (C, \mathbf{c}) \notin D^\downarrow \quad \Leftrightarrow \quad (C, \mathbf{c}) \notin D'^\downarrow$$

The first equivalence holds by Lem. 2.3 and the fact that $P_\Sigma(C) = C$. The second equivalence holds by the duality assumption. It remains to prove the third equivalence.

From left to right: By contraposition. Suppose that $(C, \mathbf{c}) \rightarrow (P_\Sigma(B'), \mathbf{b}')$ for some $(B, \mathbf{b}) \in D$, $(B', \iota) \in \Omega_{P_\Sigma}(B)$, and $\iota(\mathbf{b}') = \mathbf{b}$. Trivially, we have $id : (B', \mathbf{b}') \rightarrow (B', \mathbf{b}')$. It follows by the generalized adjoint property that $(P_\Sigma(B'), \mathbf{b}') \rightarrow (B, \iota(\mathbf{b}'))$. Therefore, by transitivity, and since $\iota(\mathbf{b}') = \mathbf{b}$, we have $(C, \mathbf{c}) \rightarrow (B, \mathbf{b})$ and therefore $(C, \mathbf{c}) \in D^\downarrow$.

From right to left: Again, by contraposition. Assume $(C, \mathbf{c}) \in D^\uparrow$. Since $P_\Sigma(C) = C$, it follows that $(P_\Sigma(C), \mathbf{c}) \rightarrow (B, \mathbf{b})$ for some $(B, \mathbf{b}) \in D$. It follows by the adjoint property that $(C, \mathbf{c}) \rightarrow (B', \mathbf{b}')$ for some $(B, \mathbf{b}) \in D$, $(B', \iota) \in \Omega_{P_\Sigma}(B)$, and $\mathbf{b}' \in \iota^{-1}(\mathbf{b})$. Then also $(C, \mathbf{c}) \rightarrow (P_\Sigma(B'), \mathbf{b}')$. This means that $(C, \mathbf{c}) \in D'^\downarrow$. ◀

Regarding complexity, consider the case where Σ is a fixed tame set of full TGDs (not treated as part of the input). Then the proof of Thm. 6.4 yields a 2ExpTime algorithm for computing the dual set D from F, assuming F is specified by the underlying set of c-acyclic instances $(A, \mathbf{a})$. Thus, for instance, for the class of transitive digraphs (which, as we saw earlier, is captured by a TAM Datalog program), we have a 2ExpTime-algorithm for constructing duals for digraphs that are specified as the transitive closure of an acyclic digraph.

The only prior results regarding homomorphism dualities for restricted classes of structures that we are aware of, are for undirected graphs and for finite algebras. An undirected graph can be viewed as an instance over a schema $\mathbf{S}$ consisting of a single binary relation symbol E, satisfying the integrity constraints $\forall xy(E(x,y) \rightarrow E(y,x))$ and $\forall x \neg E(x,x)$. It is known that the category of undirected graphs and homomorphisms has no finite dualities, up to homomorphic equivalence, other than the trivial duality $(\{K_2\},\{K_1\})$, where K_1 and K_2 are the 2-element clique and the empty graph, respectively (cf. [20]). Similarly, a finite algebra of a similarity type σ can be viewed as an $\mathbf{S}$-instance, with $\mathbf{S} = \{R_f \mid f \in \sigma\}$ satisfying $\Sigma = \{\forall \mathbf{x} \exists y R_f(\mathbf{x}, y), \forall \mathbf{x} yz(R_f(\mathbf{x}, y) \wedge R_f(\mathbf{x}, z) \rightarrow y = z) \mid f \in \sigma\}$, and, again, it is known that, in the category of finite algebras, no non-trivial finite dualities exist [4]. Note that both in the case of undirected graphs (viewed as symmetric and irreflexive relational structures) and in the case of finite algebras, all non-trivial structures in question are cyclic.

For the special case of monadic tree-shaped TGDs, we can prove a converse to Thm. 6.4:

▶ **Theorem 6.5.** *Let Σ be any set of monadic tree-shaped TGDs. Let F be any finite set of pairwise homomorphically-incomparable pointed instances $(A, \mathbf{a})$ with $A \models \Sigma$. Then, the following are equivalent:*

1. *F is the left hand side of a finite duality w.r.t. Σ,*
2. *Each $(A, \mathbf{a}) \in F$ is homomorphically equivalent to $(P_\Sigma(A'), \mathbf{a})$ for some c-acyclic $(A', \mathbf{a})$.*

▶ Remark 6.6. Thm. 6.5 cannot be lifted to arbitrary tame sets of full TGDs. Consider again the tame set of full TGDs $\Sigma = \{\forall xyzu(R(x,y) \wedge R(y,z) \wedge R(z,u) \rightarrow R(x,u)), \forall xy(R(x,y) \rightarrow R(y,x))\}$ from Example 6.2. Let A be the instance (without distinguished elements) $\{R(a,a)\}$, and let B be the instance $\{R(a,b), R(b,a)\}$. Then $(\{A\},\{B\})$ is a homomorphism duality w.r.t. Σ. Indeed, let C be an instance satisfying Σ and assume that $A \not\rightarrow C$ (i,e, C has no loop). Since C satisfies Σ it follows that C has no odd cycle and, hence, is homomorphic to B. However, it is easy to see that every instance A' satisfying $P_\Sigma(A') = A$ must have a cycle.

6.3 Uniquely Characterizing Examples for Database Queries

By a *collection of labeled examples* for a k-ary query, we mean a pair (E^+, E^-) of finite sets of pointed instances with k distinguished elements.[5] A UCQ q *fits* such (E^+, E^-) if $\mathbf{a} \in q(A)$ for all $(A, \mathbf{a}) \in E^+$, and $\mathbf{a} \notin q(A)$ for all $(A, \mathbf{a}) \in E^-$. We say that two UCQs q, q' (over the same schema $\mathbf{S}$) are *equivalent w.r.t.* Σ, where Σ is a first-order theory, if for all $I \in \text{Inst}[\mathbf{S}]$ with $I \models \Sigma$, $q(I) = q'(I)$. A collection of labeled examples (E^+, E^-) *uniquely characterizes* a UCQ q w.r.t. Σ, if q fits (E^+, E^-), and every UCQ that fits (E^+, E^-) is equivalent to q w.r.t. Σ.

▶ **Lemma 6.7.** *Let $\mathbf{S}$ be any schema and Σ a FO theory over $\mathbf{S}$. Let q be a UCQ over $\mathbf{S}$, and let E^+, E^- be finite sets of pointed instances $(I, \mathbf{a})$ with $I \models \Sigma$. The following are equivalent:*

1. *The collection of labeled examples (E^+, E^-) uniquely characterizes q w.r.t. Σ*

[5] Data examples can also be defined as pairs $(I, q(I))$ of an input instance and the complete query output. This would not change our results. Note that such a data example $(I, q(I))$ can be equivalently represented by all positive examples $(I, \mathbf{a})$ for $\mathbf{a} \in q(I)$ and negative examples $(I, \mathbf{a})$ for $\mathbf{a} \in \text{adom}(I)^k \setminus q(I)$.

2. *q fits (E^+, E^-) and (E^+, E^-) is a finite homomorphism duality w.r.t. Σ.*

▶ **Theorem 6.8.** *Let Σ a tame set of full TGDs over a schema $\mathbf{S}$. Every c-acyclic UCQ q over $\mathbf{S}$ is uniquely characterized w.r.t. Σ by a collection of labeled examples satisfying Σ.*

Proof. Let E^+ be the set of pointed instances $(P_\Sigma(I), \mathbf{a})$, for $(I, \mathbf{a})$ a (c-acyclic) canonical instance of a CQ in q. By Thm. 6.4, there is a finite set E^- such that (E^+, E^-) is a homomorphism duality w.r.t. Σ. By Lem. 6.7, (E^+, E^-) uniquely characterizes q w.r.t. Σ. ◀

The proof of Thm. 6.8 yields a 2ExpTime upper bound for computing uniquely characterizing examples for a given c-acyclic UCQ, relative to a fixed tame set of full TGDs. It is not known whether this is optimal. In the absence of integrity constraints, uniquely characterizing examples for a c-acyclic UCQ can be constructed in ExpTime and this is known to be optimal since they are in general exponential in size. In the case of c-acyclic CQs, uniquely characterizing examples can be constructed in polynomial time [9].

7 Conclusion

We introduced a new fragment of Datalog, TAM Datalog, that is semantically well-behaved (closed under composition and having a natural semantic characterization) and admits generalized right-adjoints. We used this result to obtain a method for constructing uniquely characterizing data examples for c-acyclic UCQs in the presence of integrity constraints (where the data examples are required to satisfy the integrity constraints). Generalized right-adjoints for Datalog programs seem potentially useful in other contexts as well, such as in tasks involving reasoning about a hidden database instance based on an exposed view (cf. [5]).

In a companion paper (cf. [11]), we further extend our study to ∃Datalog (the extension of Datalog where existential quantifiers are allowed in rule heads), and we show that linear ∃Datalog programs have right-adjoints. This is then used to extend our results on uniquely characterizing data examples to the case with (a weakly acyclic set of) inclusion dependencies.

We leave as open problems for future research: (i) obtaining tight complexity bounds for the task of constructing uniquely characterizing data examples for CQs and UCQs in the presence of a tame set of full TGDs; (ii) identifying better syntactic criteria that guarantees that a given finite set of full TGDs is tame; (iii) extending our results to the case with functional dependencies[6]; and (iv) studying which logic (or, more generally, formalism) is necessary to define the right adjoint $\Sigma_P(\cdot)$ (see also Remark 4.10).

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley, 1995.

2 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, December 2011. doi:10.1145/2043652.2043656.

3 Albert Atserias. On digraph coloring problems and treewidth duality. *Eur. J. Comb.*, 29(4):796–820, 2008. doi:10.1016/j.ejc.2007.11.004.

[6] Recent results in [18] show that $\mathcal{ELI}$*-concepts* (equivalently: unary connected acyclic CQs over a schema with only binary relations) are uniquely characterizable in the presence of functional dependencies, and that uniquely characterizing examples can be computed for them in polynomial time.

4 Richard N. Ball, Jaroslav Nešetřil, and Aleš Pultr. Dualities in full homomorphisms. *European Journal of Combinatorics*, 31(1):106–119, 2010. doi:10.1016/j.ejc.2009.04.004.

5 Michael Benedikt, Pierre Bourhis, Balder Ten Cate, Gabrieled Puppis, and Michael Vanden Boom. Inference from visible information and background knowledge. *ACM Trans. Comput. Logic*, 22(2), jun 2021. doi:10.1145/3452919.

6 Meghyn Bienvenu, Balder Ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through Disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4), December 2015. doi:10.1145/2661643.

7 Manuel Bodirsky, Simon Knäuer, and Sebastian Rudolph. Datalog-expressibility for monadic and guarded second-order logic. In *ICALP 2021*, pages 120:1 – 120:17, 2021.

8 Andrei A. Bulatov, Andrei A. Krokhin, and Benoît Larose. Dualities for constraint satisfaction problems. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints - An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*, volume 5250 of *LNCS*, pages 93–124. Springer, 2008. doi:10.1007/978-3-540-92800-3_5.

9 Balder ten Cate and Victor Dalmau. Conjunctive queries: Unique characterizations and exact learnability. *ACM Trans. Database Syst.*, 47(4), nov 2022. doi:10.1145/3559756.

10 Balder ten Cate, Victor Dalmau, Maurice Funk, and Carsten Lutz. Extremal fitting problems for conjunctive queries. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS'23*, 2023. doi:10.1145/3584372.3588655.

11 Balder ten Cate, Víctor Dalmau, and Jakub Opršal. Right-adjoints for datalog programs, and homomorphism dualities over restricted classes, 2023. arXiv:2302.06366.

12 Surajit Chaudhuri and Moshe Y Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *Journal of Computer and System Sciences*, 54(1):61–78, 1997. doi:10.1006/jcss.1997.1452.

13 Victor Dalmau, Andrei Krokhin, and Jakub Opršal. Functors on relational structures which admit both left and right adjoints, 2023. doi:10.48550/arXiv.2302.13657.

14 Víctor Dalmau and Jakub Opršal. Local consistency as a reduction between constraint satisfaction problems, 2023. doi:10.48550/arXiv.2301.05084.

15 Péter L. Erdös, Dömötör Pálvölgyi, Claude Tardif, and Gábor Tardos. Regular families of forests, antichains and duality pairs of relational structures. *Comb.*, 37(4):651–672, 2017. doi:10.1007/s00493-015-3003-4.

16 Jan Foniok, Jaroslav Nešetřil, and Claude Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.*, 29(4):881–899, 2008.

17 Jan Foniok and Claude Tardif. Digraph functors which admit both left and right adjoints. *Discrete Mathematics*, 338(4):527–535, 2015. doi:10.1016/j.disc.2014.10.018.

18 Maurice Funk, Jean Christoph Jung, and Carsten Lutz. Frontiers and exact learning of ELI queries under DL-Lite ontologies. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-22*, 2022. doi:10.24963/ijcai.2022/364.

19 Georg Gottlob and Christoph Koch. Monadic Datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, jan 2004. doi:10.1145/962446.962450.

20 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications.* Oxford University Press, 2004.

21 Andrei A. Krokhin, Jakub Opršal, Marcin Wrochna, and Stanislav Živný. Topology and adjunction in promise constraint satisfaction. *SIAM Journal on Computing*, 52(1):38–79, 2023. doi:10.1137/20M1378223.

22 Aleš Pultr. The right adjoints into the category of relational systems. In *Reports of the Midwest Category Seminar IV*, volume 137 of *Lecture Notes in Mathematics*, pages 100–113. Springer, 1970.

23 Sebastian Rudolph and Markus Krötzsch. Flag & check: Data access with monadically defined queries. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, page 151–162. ACM, 2013. doi:10.1145/2463664.2465227.

A Proofs for Section 3

▶ **Proposition 3.4.** *The almost-monadic Datalog program from Example 1.1 is not equivalent to a monadic Datalog program.*

Proof. Suppose, for the sake of a contradiction, that there was an equivalent monadic Datalog program P. Let n be the maximum number of variables in any `Ans` rule of P. Consider the $\mathbf{S}_{in}$-instance I consisting of the facts $R(a_0, a_1), R(a_1, a_2), \dots, R(a_n, a_{n+1})$ as well as the facts $R(b_0, b_1), R(b_1, b_2), \dots, R(b_n, b_{n+1})$. Then $\texttt{Ans}(a_0, a_{n+1})$ is a fact of $P(I)$ while $\texttt{Ans}(a_0, b_{n+1})$ is not. A simple isomorphism argument shows that, for all $i \leq n+1$ and for all $S \in \mathbf{S}_{aux}$, $S(a_i)$ belongs to $\text{chase}_P(I)$ if and only if $S(b_i)$ belongs to $\text{chase}_P(I)$. It is then easy to see that any derivation of $\texttt{Ans}(a_0, a_{n+1})$ using a rule of P implies also the existence of a derivation of $\texttt{Ans}(a_0, b_{n+1})$ using the same rule, a contradiction. ◀

▶ **Theorem 3.5.** *For each almost-monadic Datalog program P and k-ary relation symbol $R \in \mathbf{S}^P_{out}$, there is a Boolean monadic Datalog program P' where $\mathbf{S}^{P'}_{in} = \mathbf{S}^P_{in} \cup \{Q_1, \dots, Q_k\}$, such that the following are equivalent, for all $\mathbf{S}^P_{in}$-instances I and $a_1, \dots, a_k \in \text{adom}(I)$:*

1. $R(a_1, \dots, a_k) \in P(I)$,
2. $P'(I \cup \{Q_1(a_1), \dots, Q_k(a_k)\}) = \mathit{true}$.

Proof. To simplify the exposition, we may assume that the articulation position of each relation (if it has one) is the first position. Let $\mathcal{Q} = \{Q_1, \dots, Q_k\}$. For each relation $S \in \mathbf{S}^P_{out} \cup \mathbf{S}^P_{aux}$ with $\text{arity}(S) > 0$, and for each partial function $f : \{1, \dots, \text{arity}(S)\} \rightharpoonup \mathcal{Q}$, we create a unary relation S^f. The intuitive meaning of $S^f(x)$ is:

$$\exists y_1 \dots y_k (S(y_1, \dots, y_k) \wedge x = y_1 \wedge \bigwedge_{f(i)=Q_j} Q_j(y_i)) \ .$$

Let $\mathbf{S}'_{aux}$ be the set of all these new unary relations. Finally, we define the set $\Sigma^{P'}$ of rules of our new program P'. Take any rule in $\rho \in \Sigma^P$. Without loss of generality, we can we can assume that ρ is of the form

$$R_0(\mathbf{x}_0) :- R_1(\mathbf{x}_1), \dots, R_n(\mathbf{x}_n), E_{n+1}(\mathbf{x}_{n+1}), \dots, E_{n+m}(\mathbf{x}_{n+m})$$

where each $R_i \in \mathbf{S}^P_{aux} \cup \mathbf{S}^P_{out}$ and each $E_i \in \mathbf{S}_{in}$. For $0 \leq i \leq n$, let $f_i : \{1, \dots, \text{arity}(R_i)\} \rightharpoonup \mathcal{Q}$ be a partial function, such that the following consistency requirement is satisfied: whenever a variable occurs in multiple $\mathbf{S}_{aux} \cup \mathbf{S}_{out}$-atoms in the above rule, say, in the j-th argument position of the atom $R_i(\mathbf{x}_i)$ and in the j'-th argument position of the atom $R_{i'}(\mathbf{x}_{i'})$, then $f_i(j) = f_{i'}(j')$ (we allow here that $f_i(j)$ and $f_{i'}(j')$ are both undefined).

For each rule $\rho \in \Sigma^P$ and for each choice of partial functions $f_0, \dots, f_n$, satisfying the above consistency requirement, we add to $\Sigma^{P'}$ the rule

$$R_0^{f_0}(x_{0,1}) :- R_1^{f_1}(x_{1,1}), \dots, R_n^{f_n}(x_{n,1}), E_{n+1}(\mathbf{x}_{n+1}), \dots, E_{n+m}(\mathbf{x}_{n+m}), \bigwedge_{f_0(i)=Q_j} Q_j(x_{0,i})$$

where $x_{i,j}$ stands for the j-th variable in the tuple of variables $\mathbf{x}_i$.

Finally we add the rule

$$\texttt{Ans}() :- R^f(x)$$

where R is the relation mentioned in the statement of the proposition, and $f : \{1, \dots, \text{arity}(R)\} \to \mathcal{Q}$ is the total function given by $f(i) = Q_i$. This concludes the definition of the monadic Datalog program P'.

Let I be any $\mathbf{S}^P_{in}$-instance, and let $I' = I \cup \{Q_1(a_1), \dots, Q_k(a_k)\}$.

▷ Claim. The following are equivalent, for all $R^f \in \mathbf{S}'_{aux}$ and $c \in \text{adom}(I)$:

1. $R^f(c) \in \text{chase}_{P'}(I)$
2. $R(b_1, \ldots, b_n) \in \text{chase}_P(I)$ for some $b_1, \ldots, b_n$ such that $b_1 = c$ and, for all $i \leq n$, if $f(i) = Q_j$, then $b_i = a_j$.

Both directions of this claim can be proved by an induction on the length of derivations. We note that the fact that P is almost-monadic is required in the direction (1) ⇒ (2).

It follows from this claim that $\texttt{Ans}() \in P'(I')$ iff $R(a_1, \ldots, a_k) \in P(I)$. ◀

▶ **Corollary 3.7.** *Let P be an almost-monadic Datalog program and $R \in \mathbf{S}^P_{out}$. Then (P, R) defines an MSO query.*

Proof. Let P' be as in Thm. 3.5. By Thm. 3.1, there is an MSO sentence ϕ such that, for all $\mathbf{S} \cup \{Q_1, \ldots, Q_k\}$-instances I, $\texttt{Ans}() \in P'(I)$ iff $I \models \phi$. Let

$$\psi(x_1, \ldots, x_k) = \exists Q_1 \ldots Q_k(\phi \wedge \bigwedge_i \forall z(Q_i(z) \leftrightarrow z = x_i))$$

Then, for all $\mathbf{S}^P_{in}$-instances I, $I \models \psi(a_1, \ldots, a_k)$ iff $R(a_1, \ldots, a_k) \in P(I)$. ◀

▶ **Lemma 3.8.** *Let P be any almost-monadic Datalog such that every $R \in \mathbf{S}^P_{out}$ is unary. Then P is equivalent to a monadic Datalog program.*

Proof. To simplify the presentation, we may assume without loss of generality that $\mathbf{S}^P_{out} = \{\texttt{Ans}\}$. Let f be the articulation function.

We introduce a unary auxiliary relation $Q_{R,j}$ for every $R \in \mathbf{S}^P_{aux}$ and $j \leq \text{arity}(R)$ with $j \neq f(R)$. Intuitively, a fact of the form $Q_{R,j}(a)$ will be used to indicate that, for every fact $R(a_1, \ldots, a_n) \in P(I)$ with $a_{f(R)} = a$, $\texttt{Ans}(a_j)$ should be derived.

We now construct a monadic Datalog program P' with $\mathbf{S}^{P'}_{in} = \mathbf{S}^P_{in}$, $\mathbf{S}^{P'}_{out} = \mathbf{S}^P_{out}$, and $\mathbf{S}^{P'}_{aux}$ is the set of auxiliary relations of the form $Q_{R,j}$ as defined above, as well as unary auxiliary relations of the form $\widehat{R}$ for $R \in \mathbf{S}^P_{aux}$. For any atom $R(\mathbf{x})$ with $\mathbf{x} = x_1, \ldots, x_n$, we will use the denote by $\widehat{R(\mathbf{x})}$ its projection to the articulation variable, that is, $\widehat{R}(x_{f(R)})$.

Without loss of generality, we can we can assume that each rule in P is of one of the following two forms:

$$(i) \quad \texttt{Ans}(x) :- R_1(\mathbf{x}_1), \ldots, R_n(\mathbf{x}_n), E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m})$$

$$(ii) \quad R_0(\mathbf{x}_0) :- R_1(\mathbf{x}_1), \ldots, R_n(\mathbf{x}_n), E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m})$$

where each $R_i \in \mathbf{S}^P_{aux}$, and each $E_i \in \mathbf{S}^P_{in}$. In what follows, we will write $x_{i,j}$ to denote the j-th element of the tuple $\mathbf{x}_i$.

For each rule of type (i) and every occurrence $x_{i,j}$ of x in its body we add to P' the rule with body

$$\widehat{R_1(\mathbf{x}_1)}, \ldots, \widehat{R_n(\mathbf{x}_n)}, E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m})$$

and head $Q_{R_i,j}(x_{i,f(S_i)})$ whenever $i \leq n$ and $\texttt{Ans}(x)$ otherwise.

For each rule of type (ii), we add in P' the rule

$$\widehat{R_0(\mathbf{x}_0)} :- \widehat{R_1(\mathbf{x}_1)}, \ldots, \widehat{R_n(\mathbf{x}_n)}, E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m})$$

Also, for each rule of type (ii), each $k \leq \text{arity}(R_0)$ and every occurrence $x_{i,j}$ of $x_{0,k}$ in its body we add to P' the rule with body

$$\widehat{R_1(\mathbf{x}_1)}, \ldots, \widehat{R_n(\mathbf{x}_n)}, E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m}), Q_{R_0,k}(x)$$

and head $Q_{R_i,j}(x_{i,f(S_i)})$ whenever $i \leq n$ and $\texttt{Ans}(x_{0,k})$ otherwise.

▷ Claim. For every $\mathbf{S}_{in}$-instance I, $P(I) = P'(I)$.

Proof. $(P(I) \supseteq P'(I))$. It follows immediately from the following fact. For every $a \in \text{adom}(I)$:

1. If $\widehat{R}(a) \in \text{chase}_{P'}(I)$ then $R(a_1, \ldots, a_n) \in \text{chase}_P(I)$ for some tuple $a_1, \ldots, a_n$ with $a_{f(R)} = a$, and
2. If $Q_{R,j}(a) \in \text{chase}_{P'}(I)$ then for all $R(a_1, \ldots, a_n) \in \text{chase}_P(I)$ with $a_{f(R)} = a$, $\texttt{Ans}(a_j) \in P(I)$

This fact can be proved by induction on the derivation order of $P'(I)$.

$(P(I) \subseteq P'(I))$ It follows from the following fact. Let

$$\texttt{Ans}(x) :- R_1(\mathbf{x}_1), \ldots, R_n(\mathbf{x}_n), E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m})$$

be any *derivable* rule of P and let X be the canonical instance of

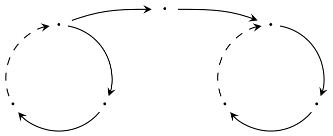
$$\widehat{R_1(\mathbf{x}_1)}, \ldots, \widehat{R_n(\mathbf{x}_n)}, E_{n+1}(\mathbf{x}_{n+1}), \ldots, E_{n+m}(\mathbf{x}_{n+m})$$

Then for every occurrence $x_{i,j}$ of x in the body, $\text{chase}_{P'}(X)$ contains $Q_{R_i,j}(x_{f(R_i)})$ if $i \leq n$ and $\texttt{Ans}(x)$ otherwise (here we are abusing slightly notation since X is not necessarily a $\mathbf{S}^P_{in}$-instance but the meaning is clear). This is proved by structural induction on the derivable rules of P. ◁

◀

▶ **Proposition 3.9.** *The unary MSO query "x lies on a directed R-cycle" is not definable by an almost-monadic Datalog program.*

Proof. By Lem. 3.8, it suffices to show that the query in question is not definable by a monadic Datalog program. Suppose, for the sake of a contradiction, that the query is defined by a monadic Datalog program P, with answer relation $\texttt{Ans}$, and let n be greater than the maximum number of atoms in a rule body of P. Consider the following instance I, where the two cycles have length n.



Clearly, $P(I)$ should contain $\texttt{Ans}(a)$ for all elements that lie on one of the two cycles, and not for the intermediate element. However, a straightforward induction argument shows that after each iteration of applying the rules of P, (i) all elements satisfy the same facts, i.e., whenever a fact of the form $S(a)$ is derived, with $S \in \mathbf{S}^P_{aux} \cup \mathbf{S}^P_{out}$, then $S(a')$ is also derived, for all other elements a'; and (ii) for every two elements a, a', their neighborhoods of diameter n are homomorphically equivalent. In particular, if $\texttt{Ans}(a)$ is derived from some element a, then $\texttt{Ans}(a')$ is derived for all elements a'. ◀

▶ **Lemma 3.12.** *Let P be any tree-shaped Datalog program. Then, for each $R \in \mathbf{S}^P_{out}$,* $\text{Unfoldings}(P, R)$ *consists of acyclic pointed instances.*

Proof. Recall that $\text{Unfoldings}(P, R)$ consists of canonical instances of derivable rules, where a derivable rule is a rule can be obtained from the rules of P through the operation of substituting occurrences of a rule head by the corresponding rule body. It is easy to see that this substitution operation preserves tree-shapedness, and therefore every derived rule is tree-shaped. It follows that every $\text{Unfoldings}(P, R)$ consists of acyclic pointed instances. ◀

▶ **Theorem 3.13.** *Let $\phi(x_1, \ldots, x_n)$ be an MSO formula. The following are equivalent:*

1. *ϕ is definable by a TAM Datalog program,*
2. *ϕ is definable by a tree-shaped Datalog program,*
3. *ϕ is tree-determined.*

The proof is deferred to Appendix C.

▶ **Corollary 3.15** (TAM Datalog is closed under composition)**.** *For all TAM Datalog programs P_1 and P_2 with $\mathbf{S}^{P_2}_{in} = \mathbf{S}^{P_1}_{out}$, there is a TAM Datalog program $P_3 = (\mathbf{S}^{P_1}_{in}, \mathbf{S}^{P_2}_{out}, \mathbf{S}'_{aux}, \Sigma')$ such that, for all $\mathbf{S}^{P_1}_{in}$-instances I, $P_3(I) = P_2(P_1(I))$.*

Proof. The composition of P_1 and P_2 is clearly expressible as a tree-shaped Datalog program: we may assume that $\mathbf{S}^{P_1}_{aux}$ and $\mathbf{S}^{P_2}_{aux}$ are disjoint. Let $P_3 = (\mathbf{S}^{P_1}_{in}, \mathbf{S}^{P_2}_{out}, \mathbf{S}^{P_1}_{aux} \cup \mathbf{S}^{P_1}_{out} \cup \mathbf{S}^{P_2}_{aux}, \Sigma^{P_1} \cup \Sigma^{P_2})$. Then P_3 defines the composition of P_1 and P_2. Note that P_3 is tree-shaped but no longer necessarily almost-monadic. As we will show, however, P_3 is nevertheless equivalent to a TAM Datalog program.

For each k-ary relation $R \in \mathbf{S}^{P_1}_{out}$, let $\phi_R(x_1, \ldots, x_k)$ be the MSO query over schema $\mathbf{S}^{P_1}_{in}$ defined by (P_1, R). Similarly, for each k-ary relation $S \in \mathbf{S}^{P_2}_{out}$, let $\phi_S(x_1, \ldots, x_k)$ be the MSO query over schema $\mathbf{S}^{P_2}_{in}$ defined by (P_2, S). We can substitute, in ϕ_S, all occurrences of relation symbols $R \in \mathbf{S}^{P_1}_{out}$ by their defining formula ϕ_R. In this way, we obtain, for each $S \in \mathbf{S}^{P_2}_{out}$, an MSO query ϕ'_S over the schema $\mathbf{S}^{P_1}_{in}$. Note that ϕ'_S is precisely the MSO query defined by (P_3, S).

It follows by Thm. 3.13 that, for each $S \in \mathbf{S}^{P_2}_{out}$, the MSO query ϕ'_S is definable by a TAM Datalog program. As a last step, we merge the TAM Datalog programs in question to obtain a single TAM Datalog program that is equivalent to P_3. ◀

▶ **Theorem 3.16.** *Every (connected) TAM Datalog program can be transformed in polynomial-time into an equivalent (connected) simple TAM Datalog program.*

Proof. First, we will show how to ensure that each rule contains at most one occurrence of a relation from $\mathbf{S}_{in}$. Consider any rule whose body has two or more conjuncts involving relations from $\mathbf{S}_{in}$. Since the program is tree-shaped, the incidence graph of the rule body is acyclic. It follows that the rule in question can be written (by re-ordering the atoms in the body as needed) as follows:

$$R_0(\mathbf{x}_0) :- R_1(\mathbf{x}_1), \ldots, R_i(\mathbf{x}_i), R_{i+1}(\mathbf{x}_{i+1}), \ldots, R_n(\mathbf{x}_n)$$

where one of the relations $R_1, \ldots, R_i$ is in $\mathbf{S}_{in}$, one of the relations $R_{i+1}, \ldots, R_n$ is in $\mathbf{S}_{in}$, and the intersection $\{\mathbf{x}_1, \ldots, \mathbf{x}_i\} \cap \{\mathbf{x}_{i+1}, \ldots, \mathbf{x}_n\}$ contains at most one variable z. Indeed, if the program is connected, such a variable z must exist.

Let $\mathbf{u}$ be an enumeration of the variables in $\{\mathbf{x}_{i+1}, \ldots, \mathbf{x}_n\}$ without duplicates, and starting with z, or otherwise starting with any variable occurring in an input-relation atom in $R_{i+1}(\mathbf{x}_{i+1}), \ldots, R_n(\mathbf{x}_n)$.

We can replace the above rule by the following two rules:

$$\begin{aligned} R_0(\mathbf{x}_0) &:- R_1(\mathbf{x}_1), \ldots, R_i(\mathbf{x}_i), R'(\mathbf{u}) \\ R'(\mathbf{u}) &:- R_{i+1}(\mathbf{x}_{i+1}), \ldots, R_n(\mathbf{x}_n) \end{aligned}$$

where R' is a fresh auxiliary relation of suitable arity, whose articulation position is the first position. Observe that both new rules have strictly fewer occurrences of relations from $\mathbf{S}_{in}$ than the original rule, and that this construction preserves connectedness. If we repeat this process, we will end up with at most a linear number of rules, each of size no greater than the size of the original rule. Furthermore, this can clearly be performed in polynomial time.

Next, we explain how to ensure that each rule body contains at least one (hence, exactly one) relation from $\mathbf{S}_{in}$. Here we use the fact that every tuple that is derived into a defined relation must consist of values originating from facts of the input instance. Specifically, given a rule

$$R_0(\mathbf{x}_0) :- R_1(\mathbf{x}_1), \ldots, R_n(\mathbf{x}_n)$$

not containing any relation from $\mathbf{S}_{in}$. Let x be any variable occurring in the articulation position of one of the atoms in the rule body, and replace the rule by all possible rules that extend its body with an additional atom $R'(\mathbf{u}, x, \mathbf{v})$ with $R' \in \mathbf{S}_{in}$ and $\mathbf{u}, \mathbf{v}$ distinct, fresh variables. Again, connectedness is preserved. ◀

B Proofs for Section 6

By the *c-girth* of a pointed instance $(A, \mathbf{a})$ we will mean the length of the smallest cycle in the incidence graph of A that does not pass through any element in $\mathbf{a}$ (or ∞ if no such cycle exists). Observe that a pointed instance is c-acyclic if and only if its c-girth is ∞.

▶ **Lemma B.1** (Sparse Incomparability Lemma with Designated Elements). *For every pointed instance $(I, \mathbf{a})$ and $m > 0$, there is a pointed instance $(I', \mathbf{a})$ of c-girth at least m, such that $(I', \mathbf{a}) \to (I, \mathbf{a})$ and such that, for all pointed instances $(J, \mathbf{b})$ of size at most m, $(I, \mathbf{a}) \to (J, \mathbf{b})$ iff $(I', \mathbf{a}) \to (J, \mathbf{b})$.*

Proof. Let $(I, \mathbf{a})$ be given, with $\mathbf{a} = a_1 \ldots a_k$, and where I is an instance over schema $\mathbf{S}$. Let $\widehat{I}$ be the instance over schema $\widehat{\mathbf{S}} = \mathbf{S} \cup \{Q_1, \ldots, Q_k\}$ that extends I with the unary facts $Q_i(a_i)$. By the standard version of the sparse incomparability lemma, there is an $\widehat{\mathbf{S}}$-instance I'' of girth at least m such that $I'' \to \widehat{I}$ and such that, for all $\widehat{\mathbf{S}}$-instances J of size at most m, $I'' \to J$ iff $\widehat{I} \to J$. Now, let I' be the $\mathbf{S}$-instance obtained from I'' by (i) replacing every element satisfying a unary predicate Q_i by a_i, and (ii) dropping the unary predicates Q_i. This operation may introduce new cycles but it is not hard to see that any such newly introduced short cycle must pass through one of the designated elements. Therefore, $(I', \mathbf{a})$ has c-girth at least m. Furthermore, for all $\mathbf{S}$-instances $(J, \mathbf{b})$ of size at most m, we have that $(I, \mathbf{a}) \to (J, \mathbf{b})$ iff $\widehat{I} \to \widehat{J}$ iff $I'' \to \widehat{J}$ iff $(I', \mathbf{a}) \to (J, \mathbf{b})$. ◀

▶ **Theorem 6.5.** *Let Σ be any set of monadic tree-shaped TGDs. Let F be any finite set of pairwise homomorphically-incomparable pointed instances $(A, \mathbf{a})$ with $A \models \Sigma$. Then, the following are equivalent:*

1. *F is the left hand side of a finite duality w.r.t. Σ,*
2. *Each $(A, \mathbf{a}) \in F$ is homomorphically equivalent to $(P_\Sigma(A'), \mathbf{a})$ for some c-acyclic $(A', \mathbf{a})$.*

Proof. The 2 to 1 direction follows immediately from Theorem 6.4. From 1 to 2: suppose F has a finite duality w.r.t. Σ, consisting of $(D_1, \mathbf{d}_1), \ldots, (D_n, \mathbf{d}_n)$ where $D_i \models \Sigma$, and let $(A, \mathbf{a}) \in F$. Then $(A, \mathbf{a}) \not\to (D_i, \mathbf{d}_i)$ for all $i \leq n$. Let $m = s \cdot t$, where s is the number of facts in A, and t is the maximum number of conjuncts in the body of a TGD in Σ. By Lem. B.1, there is a pointed instance $(A', \mathbf{a})$ of c-girth at least m, such that $(A', \mathbf{a}) \to (A, \mathbf{a})$ and $(A', \mathbf{a}) \not\to (D_i, \mathbf{d}_i)$ for all $i \leq n$. Since $(A', \mathbf{a}) \to (A, \mathbf{a})$ and $A \models \Sigma$, we have $(P(A'), \mathbf{a}) \to (P(A), \mathbf{a}) \to (A, \mathbf{a})$ (by Lem. 2.3 and Lem. 6.1(2)).

We may assume without loss of generality that A' contains all unary facts belonging to $P(A')$. This is because (i) adding these unary facts does not change the c-girth of the instance, and (ii) when extending A' with facts from $P(A')$, the condition that $(A', \mathbf{a}) \to (A, \mathbf{a})$ is preserved, because $P(A', \mathbf{a}) \to (A, \mathbf{a})$, (iii) the condition that $(A', \mathbf{a}) \not\to (D_i, \mathbf{d}_i)$ is clearly also preserved when extending A' with additional facts.

We already observed that $(P_\Sigma(A'), \mathbf{a}) \to (A, \mathbf{a})$. Furthermore, $(P_\Sigma(A'), \mathbf{a}) \not\to (D_i, \mathbf{d}_i)$ (because, otherwise, since $(A', \mathbf{a}) \subseteq (P_\Sigma(A'), \mathbf{a})$, we would have $(A', \mathbf{a}) \to (D_i, \mathbf{d}_i)$). Since $P_\Sigma(A') \models \Sigma$ (by Lem. 6.1(1)) and $(P_\Sigma(A'), \mathbf{a}) \not\to (D_i, \mathbf{d}_i)$, by the duality assumption, some pointed instance in F maps homomorphically to $(P_\Sigma(A'), \mathbf{a})$. In fact, the pointed instance in question must be $(A, \mathbf{a})$ (otherwise we would obtain a contradiction with the fact that the members of F are pairwise homomorphically incomparable). Let $h : (A, \mathbf{a}) \to (P_\Sigma(A'), \mathbf{a})$.

Let $(B, \mathbf{a})$ be the sub-instance of $(P_\Sigma(A'), \mathbf{a})$ that is the image of $(A, \mathbf{a})$ under h. Since all unary facts in $P_\Sigma(A')$ already belong to A', and Σ is monadic, every fact in $P_\Sigma(A')$ either belongs to A' or else can be derived from facts in A' by a single rule application. It follows that there is a sub-instance B' of A of size at most $|B| \cdot t$, such that $B \subseteq P_\Sigma(B')$. Since $|B| \leq s$, it follows that B' is c-acyclic. Furthermore, $(A, \mathbf{a}) \to (P_\Sigma(B'), \mathbf{a})$, and $(P_\Sigma(B'), \mathbf{a}) \subseteq (P_\Sigma(A'), \mathbf{a}) \to (A, \mathbf{a})$, hence also $(P_\Sigma(B'), \mathbf{a}) \to (A, \mathbf{a})$. Therefore, $(A, \mathbf{a})$ is homomorphically equivalent to $(P_\Sigma(B'), \mathbf{a})$. ◀

▶ **Lemma 6.7.** *Let $\mathbf{S}$ be any schema and Σ a FO theory over $\mathbf{S}$. Let q be a UCQ over $\mathbf{S}$, and let E^+, E^- be finite sets of pointed instances $(I, \mathbf{a})$ with $I \models \Sigma$. The following are equivalent:*

1. *The collection of labeled examples (E^+, E^-) uniquely characterizes q w.r.t. Σ*
2. *q fits (E^+, E^-) and (E^+, E^-) is a finite homomorphism duality w.r.t. Σ.*

Proof. The following proof does not in fact depend on Σ being a *first-order* theory. The requirement that Σ is a FO theory, in the statement of the lemma, merely stems from the way we set up the definitions of the notions involved.

From 1 to 2, if (E^+, E^-) uniquely characterizes q w.r.t. Σ, then, by definition, q fits (E^+, E^-). Furthermore, it follows that no pointed instance in E^+ admits a homomorphism to a pointed instance in E^- (otherwise, it would follow by monotonicity of UCQs that q does not fit the negative examples). Next, assume for the sake of a contradiction that (E^+, E^-) is not a homomorphism duality with respect to Σ. Then there is a pointed instance $(I, \mathbf{a})$ with $I \models \Sigma$ that neither belongs to $E^{+\uparrow}$, not to $E^{-\downarrow}$. Let q_1 be the union of the canonical CQs of E^+ and let q_2 be the union of the canonical CQs of $E^+ \cup \{(I, \mathbf{a})\}$. Then q_1 and q_2 are not equivalent under Σ, and both fit (E^+, E^-), a contradiction.

From 2 to 1, let q' be any UCQ that fits (E^+, E^-). We must show that q' is equivalent to q w.r.t. Σ. Consider any pointed instance $(I, \mathbf{a})$ with $I \models \Sigma$. If $\mathbf{a} \in q(I)$ then $(I, \mathbf{a}) \in E^{+\uparrow}$, therefore $\mathbf{a} \in q'(I)$. If, on the other hand, $\mathbf{a} \notin q(I)$, then $(I, \mathbf{a}) \notin E^{+\uparrow}$, hence $(I, \mathbf{a}) \in E^{-\downarrow}$, hence $\mathbf{a} \notin q'(I)$. ◀

C Expressive completeness of TAM Datalog (Proof of Thm. 3.13)

Fix a schema $\mathbf{S}$ and let $\mathbf{X} = \{X_1, \ldots, X_n\}$ (which we can consider to be schema consisting of unary predicates). Following [13] we consider the set of formal "tree-terms" defined inductively from the following operators.

- for every $S \subseteq \mathbf{X}$, $\bullet_S$ is a tree-term.
- for every $R \in \mathbf{S}$, for all tree-terms $t_1, \ldots, t_k$ with $k = \text{arity}(R)$, and for each $i \in [k]$, $\blacktriangledown_i^R(t_1, \ldots, t_k)$ is a tree-term.

We define for each tree-term t an associated pointed tree $(T(t), r(t))$ inductively as follows.

- If $t = \bullet_S$ then $T(t)$ is the tree containing only one node v (hence $r(t) = v$) and facts $X_i(v)$ for every $X_i \in S$.
- If $t = \blacktriangledown_i^R(t_1, \ldots, t_k)$ then $T(t)$ is the tree obtained by taking the disjoint union of $T(t_1), \ldots, T(t_k)$ and adding fact $f = R(r(t_1), \ldots, r(t_k))$. Furthermore, $r(t) = r(t_i)$.

▶ **Lemma C.1.** *For every finite connected acyclic pointed* $(\mathbf{S} \cup \mathbf{X})$*-instance* (I, a)*, there is a tree-term* t *such that* $(T(t), r(t))$ *is isomorphic to* (I, a)*.*

Proof. The proof is by induction on the size of instance I, as counted by the number of $\mathbf{S}$-facts. The base case of the induction is where I does not contain any $\mathbf{S}$-facts. In this case, it follows from connectedness that I must be a single-element structure containing only some $\mathbf{X}$-facts. In this case, the statement clearly holds: it suffices to take t to be the term $\bullet_S$ where S is the set of all $X_i \in \mathbf{X}$ appearing in I.

If I contains n $\mathbf{S}$-facts, with $n > 0$, then, by connectedness, a must appear in at least one $\mathbf{S}$-fact, that is, I contains a fact of the form $R(a_1, \ldots, a_n)$ where, say, $a_i = a$. Let I' be the sub-instance of I where the fact $R(a_1, \ldots, a_n)$ is removed. For each $j \leq n$, let I_j be the connected component of I' containing a_j. By induction, there is a term t_j such that $(T(t_i), r(t_i))$ is isomorphic to (I_j, a_j). Let $t = \blacktriangledown_i^R(t_1, \ldots, t_n)$. Then it is easy to see that $(T(t), r(t))$ is isomorphic to (I, a). ◀

An automaton, for present purposes, is a tuple $(\mathbf{S}, \mathbf{X}, Q, F, \delta)$ consisting of:

- schemas $\mathbf{S}$, $\mathbf{X}$.
- A finite set Q of states, with a distinguished subset $F \subseteq Q$
- For every operator o of the form $\bullet_S$ or $\blacktriangledown_i^R$, of arity, say, r (where we view $\bullet_S$ as a zero-ary operator), a transition relation $\delta_o \subseteq Q^r \times Q$

Acceptation is as one would expect. A tree-term t is accepted if we can associate a state $q_{t'}$ to each one of its subterms t' such that $q_t \in F$ and the mapping $t' \mapsto q_{t'}$ respects the transition relation (meaning, that if $t' = o(t'_1, \ldots, t'_r)$ then $(q_{t'_1}, \ldots, q_{t'_r}, q_{t'}) \in \delta_o$.

Given a MSO formula $\phi(x_1, \ldots, x_n)$ with schema $\mathbf{S}$ we shall consider the following associated formula ϕ' defined to be the MSO-sentence with schema $\mathbf{S} \cup \mathbf{X}$ defined as

$$\exists x_1, \ldots, x_n(\phi(x_1, \ldots, x_n) \wedge \bigwedge_{i=1\ldots n} X_i(x_i)) \qquad \text{(Eq. 4)}$$

▶ **Lemma C.2.** *If* ϕ *is monotone then* ϕ' *is monotone as well.*

Proof. Assume that $h : A \to B$, where A and B are $\mathbf{S} \cup \mathbf{X}$-instances and assume that A satisfies ϕ'. Let $x_i \mapsto a_i$ be the instantiation witnessing it. Since the predicates of $\mathbf{X}$ do not appear in ϕ it follows that the $\mathbf{S}$-reduct of A satisfies $\phi(a_1, \ldots, a_n)$. Since ϕ is monotone it follows that the $\mathbf{S}$-reduct of B satisfies $\phi(h(a_1), \ldots, h(a_n))$. Since $h(a_i) \in B^{X_i}$ for each $1 \leq i \leq n$, it follows that B satisfies ϕ'. ◀

▶ **Theorem C.3.** *Let* ϕ' *be a MSO-sentence with schema* $\mathbf{S} \cup \mathbf{X}$*. Then there is a finite automaton that accepts the set of all tree-terms* t *such that* $T(t)$ *satisfies* ϕ'*.*

Proof. The proof is entirely standard. For the sake of completeness, we spell out the construction, but we will omit the correctness argument. As is customary in the literature on automata theory and MSO, we will simplify things by assuming a syntactic normal form for MSO-formulas, in which all quantification is second-order. More precisely, we consider formulas built up from atomic formulas of the form

- $R(X_1, \ldots, X_n)$, treated as a shorthand for $\exists x_1, \ldots, x_n(R(x_1, \ldots, x_n) \wedge X_1(x_1) \wedge \cdots \wedge X_n(x_n))$,

- $X_1 \subseteq X_2$, treated as shorthand for $\forall y(X_1(y) \to X_2(y))$, and
- $Singleton(X)$, treated as shorthand for $\exists x(X(x) \land \forall y(X(y) \to y = x))$

using disjunction, negation, and existential second-order quantification. It is easy to construction an automaton for each of the above atomic formulas. The connectives are handled by the following standard closure operations on automata:

- The union of two automata $(\mathbf{S}, \mathbf{X}, Q^i, F^i, \delta^i)$ $i = 1, 2$ (assume that Q^1 and Q^2 are disjoint) is the automaton $(\mathbf{S}, \mathbf{X}, Q^1 \cup Q^2, F^1 \cup F^2, \delta)$ where $\delta_o = \delta_o^1 \cup \delta_o^2$.
- The complement of an automaton $(\mathbf{S}, \mathbf{X}, Q, F, \delta)$ is the (deterministic) automaton $(\mathbf{S}, \mathbf{X}, 2^Q, F', \delta')$, where $F' = \{Q' \subseteq Q \mid F \cap Q' = \emptyset\}$ and where $(Q_1, \ldots, Q_r, Q_{r+1}) \in \delta'_o$ iff $Q_{r+1} = \{q \in Q \mid (q_1, \ldots, q_r, q) \in \delta_o$ for some $q_1 \in Q_1, \ldots, q_r \in Q_r\}$.
- The projection of $(\mathbf{S}, \mathbf{X}, Q, F, \delta)$ to $\mathbf{X}' \subseteq \mathbf{X}$ is defined to be $(\mathbf{S}, \mathbf{X}', Q, F, \delta')$, where δ' is obtained by modifying δ in the following way. For every $S' \subseteq \mathbf{X}'$, $\delta'_{\bullet_{S'}} = \bigcup_{S \cap \mathbf{X}' = S'} \delta_{\bullet_S}$. ◀

▶ **Theorem C.4.** *Let $A = (\mathbf{S}, \mathbf{X}, Q, F, \delta)$ be an automaton. There is a Boolean monadic tree-shaped Datalog program P with $\mathbf{S}^P_{in} = \mathbf{S} \cup \mathbf{X}$ such that for all $(\mathbf{S} \cup \mathbf{X})$-instances I, the following are equivalent:*

1. $\mathtt{Ans}() \in P(I)$.
2. *There is some tree-term t accepted by A such that $T(t) \to I$.*

Proof. For every state q, $\mathbf{S}^P_{aux}$ has a unary symbol E_q. Let us describe the rules in P:

- For every $o = \bullet_S$ and every $q \in \delta_o$, Σ^P contains the rule with head $E_q(x)$ and whose body contains $X_i(x)$ for every $X_i \in S$.
- For every $o = \blacktriangledown^R_i$ and every $(q_1, \ldots, q_k, q) \in \delta_o$, Σ^P contains the rule

$$E_q(x_i) :- R(x_1, \ldots, x_k), E_{q_1}(x_1), \ldots, E_{q_k}(x_k)$$

- For every $q \in F$, we introduce the rule

$$\mathtt{Ans}() :- E_q(x)$$

Let I be any $(\mathbf{S} \cup \mathbf{X})$-instance. The correctness of the construction follows from the following claim:

▷ Claim. The following are equivalent for each $a \in \text{adom}(I)$ and $q \in Q$:

1. $E_q(a) \in \text{chase}_P(I)$
2. There exists some tree-term t such that (i) $(T(t), r(t)) \to (I, a)$ and (ii) there is a run of A on input t that finishes at state q

We omit the proof as it is fairly standard. The (1) → (2) direction is proved by induction on the derivation length and the (2) → (1) direction is by structural induction on t. ◀

▶ **Theorem C.5.** *Let P be a Boolean monadic tree-shaped Datalog program where $\mathbf{X} \subseteq \mathbf{S}^P_{in}$. There exists a TAM Datalog program P' with $\mathbf{S}^{P'}_{in} = \mathbf{S}^P_{in} \setminus \mathbf{X}$ and $\mathbf{S}^{P'}_{out} = \{R\}$ where R has arity n such that for every $\mathbf{S}^{P'}_{in}$-instance I and every $a_1, \ldots, a_n \in \text{adom}(I)$ the following are equivalent:*

1. $R(a_1, \ldots, a_n) \in P'(I)$
2. $P(I \cup \{X_1(a_1), \ldots, X_n(a_n)\}) = true$

Proof. Let U be the set of all tuples (of possibly arity 0) $\mathbf{u}$ whose entries are non repeated integers from $[n]$. Let us assume that the output predicate of P is $\texttt{Ans}$.

We shall define a TAM Datalog program P' satisfying the requirements of the lemma. The set $\mathbf{S}^{P'}_{aux}$ contains for every $S \in \mathbf{S}_{out} \cup \mathbf{S}^{P}_{aux} \cup \mathbf{X}$ and every $\mathbf{u} \in U$ a predicate $Q_{S,\mathbf{u}}$ or arity r is $S = Ans$ and $1 + r$ otherwise where r is the arity of $\mathbf{u}$. The intended meaning of $Q_{S,\mathbf{u}}$ is the following:

$$Q_{S,\mathbf{u}}(\mathbf{a},\mathbf{b}) \in \text{chase}_{P'}(I) \Leftrightarrow S(\mathbf{a}) \in \text{chase}_{P}(I \cup \{X_{\mathbf{u}_1}(\mathbf{b}_1),\dots,X_{\mathbf{u}_r}(\mathbf{b}_r)\})$$

We note that here and elsewhere in the proof $\mathbf{a}$ is a 0-ary array whenever $S = Ans$ and an 1-ary array elsewhere.

We include in P' the following rules. First, for every $i \in [n]$ we include the rule

$$Q_{X_i,i}(x,x) :\!-\ \text{(empty body)}$$

We note that although this rule is unsafe (that is, the variable in the head does not occur in the body) this can be easily fixed extending the rule body with an $\mathbf{S}^{P'}_{in}$-atom containing x in one position and fresh variables in all other positions of the atom (there are multiple ways to do this, and we add all safe rules that can be obtained in this way).

Secondly, for every rule ρ in P we add to P' a collection of rules constructed in the following way. First, ρ can be written (by re-ordering the atoms in the body as needed) as follows:

$$R_0(\mathbf{x}_0) :\!-\ R_1(x_1),\dots,R_k(x_k),E_{k+1}(\mathbf{x}_{k+1}),\dots,E_m(\mathbf{x}_m)$$

where $R_0 \in \mathbf{S}^{P}_{out} \cup \mathbf{S}^{P}_{aux}$, each $R_i \in \mathbf{S}^{P}_{aux} \cup \mathbf{X}$ and each $E_i \in \mathbf{S}^{P}_{in}$. Then, for every $\mathbf{u}_1,\dots,\mathbf{u}_k \in U$ having no common elements we add to P' the rule

$$Q_{R_0,\mathbf{u_0}}(\mathbf{x}_0,\mathbf{y}_0) :\!-\ Q_{R_1,\mathbf{u}_1}(x_1,\mathbf{y}_1),\dots,Q_{R_k,\mathbf{u}_k}(x_k,\mathbf{y}_k),E_{k+1}(\mathbf{x}_{k+1}),\dots,E_m(\mathbf{x}_m)$$

where $\mathbf{y}_i (i \in [k])$ are tuples of different fresh variables, $\mathbf{u_0} = (\mathbf{u}_1 \dots, \mathbf{u}_k)$, and $\mathbf{y_0} = (\mathbf{y}_1 \dots, \mathbf{y}_k)$.

Further, for every $Q_{S,\mathbf{v}} \in \mathbf{S}^{P'}_{aux}$ and every $\mathbf{u} \subseteq \mathbf{v}$ we add the rule

$$Q_{S,\mathbf{v}}(\mathbf{x},\mathbf{z}) :\!-\ Q_{S,\mathbf{u}}(\mathbf{x},\mathbf{y})$$

where $\mathbf{x}$ and $\mathbf{z}$ do not have repeated elements or elements in common and for every i, j

$$\mathbf{u}_i = \mathbf{v}_j \Rightarrow \mathbf{y}_i = \mathbf{z}_j.$$

Note that $\mathbf{y}$ is completely determined from $\mathbf{z}$ and that although this rule is again unsafe but it might be turned into a safe one as mentioned above.

Finally, we add the rule

$$R(y_1,\dots,y_n) :\!-\ Q_{\texttt{Ans},(1,\dots,n)}(y_1,\dots,y_n).$$

It only remains to show that P' satisfies the requirements of the Lemma. It is immediate from the construction of P' that it is TAM Datalog program. The rest of the proof follows immediately from the following claim which can be proved by induction on the derivation order.

▷ Claim. Let I be a $\mathbf{S}^{P'}_{in}$-instance, let $Q_{S,\mathbf{u}} \in \mathbf{S}^{P'}_{aux}$, let $r = \text{arity}(\mathbf{u})$, and let $\mathbf{a},\mathbf{b}$ tuples of variables in $\text{adom}(I)$. The following are equivalent:

1. $Q_{S,\mathbf{u}}(\mathbf{a},\mathbf{b}) \in \text{chase}_{P'}(I)$
2. $S(\mathbf{a}) \in \text{chase}_P(I \cup \{X_{\mathbf{u}_1}(\mathbf{b}_1), \ldots, X_{\mathbf{u}_r}(\mathbf{b}_r)\})$

Proof. As mentioned both implications are easily proved by induction on the derivation order. However, it is convenient to note that, since P is tree-shaped, direction (2) ⇒ (1) only needs to be proven in the case I (and hence instance $I \cup \{X_{\mathbf{u}_1}(\mathbf{b}_1), \ldots, X_{\mathbf{u}_r}(\mathbf{b}_r)\}$) is acyclic. ◁

◀

▶ **Theorem 3.13.** *Let $\phi(x_1, \ldots, x_n)$ be an MSO formula. The following are equivalent:*
1. *ϕ is definable by a TAM Datalog program,*
2. *ϕ is definable by a tree-shaped Datalog program,*
3. *ϕ is tree-determined.*

Proof. From 1 to 2 is immediate. From 2 to 3 follows immediately from Lem. 3.12 and Lem. 2.5. In the remainder, we prove (3) → (1).

Assume that $\phi(x_1, \ldots, x_n)$ is an MSO formula over $\mathbf{S}_{in}$ that satisfies (3).

Let S be a fresh binary relation symbol not in $\mathbf{S}_{in}$. In particular, S does not occur in ϕ. Let $\mathbf{S} = \mathbf{S}_{in} \cup \{S\}$. For the purpose of the next steps of the proof, we will view ϕ as a formula over $\mathbf{S}$. This also means that we will be constructing a corresponding Datalog program over the input schema $\mathbf{S}$. Afterwards, we will deal with eliminating the relation S from the schema. The reason for extending the schema is that it will help us bridge the gap between connected instances and arbitrary (possibly disconnected) instances. Specifically, we will make use the fact that every $\mathbf{S}_{in}$-instance is the $\mathbf{S}_{in}$-reduct of a connected $\mathbf{S}$-instance.

Let ϕ' be the MSO sentence over $\mathbf{S} \cup \mathbf{X}$ as defined as in Eq. 4 (where $\mathbf{X} = \{X_1, \ldots, X_n\}$). Let A be the automaton corresponding to ϕ' as in Theorem C.3, let P as the Boolean connected tree-shaped monadic Datalog program as in Theorem C.4 and let P' by the TAM Datalog program corresponding to P as in Thm. C.5. Recall that $\mathbf{S}^{P'}_{out}$ consists of a single n-ary relation symbol R.

▷ Claim 2. (P', R) is equivalent to ϕ over connected $\mathbf{S}$-instances. That is, for all connected $\mathbf{S}$-instances I and for all tuples $\mathbf{a} \in \text{adom}(I)^n$, $R(\mathbf{a}) \in P'(I)$ iff $I \models \phi(\mathbf{a})$.

Proof. Assume that P' on a connected $\mathbf{S}$-instance I produces $R(a_1, \ldots, a_n)$. Let $\widehat{I}$ be the connected $(\mathbf{S} \cup \mathbf{X})$-instance extending I with $X_1(a_1), \ldots, X_n(a_n)$. Then, it follows that $P(\widehat{I}) = \mathit{true}$. Then, there is some tree-term t accepted by A such that $T(t) \to \widehat{I}$. It follows that $T(t)$ satisfies ϕ'. Consequently, we have that $\widehat{I}$ satisfies ϕ'. It follows that I satisfies $\phi(a_1, \ldots, a_n)$. Note that for this direction we do not use the full condition of tree-determinacy, only monotonicity.

Conversely, assume that I satisfies $\phi(a_1, \ldots, a_n)$. Then by (3)

$$(J, b_1, \ldots, b_n) \to (I, a_1, \ldots, a_n)$$

for some J and $b_1, \ldots, b_n$ such that J satisfies $\phi(b_1, \ldots, b_n)$. Let $\widehat{J}$ be the $(\mathbf{S} \cup \mathbf{X})$-instance extending J with $X_1(b_1), \ldots, X_n(b_n)$. Let t be a tree-term such that $T(t)$ is isomorphic to $\widehat{J}$, as given by Lem. C.1. It follows that A accepts t. Consequently $P(\widehat{I}) = \mathit{true}$. It follows that $R(a_1, \ldots, a_n)$ belongs to $P'(I)$. ◁

Finally, let P'' be the TAM Datalog program obtained from P' by dropping all occurrences of the relation S from the body of every rule of P'. The operation of dropping all occurrences of S might make some rules unsafe. That is, one or more variable x occurring in the head of a rule might not occur in the body anymore. This can, however, be easily fixed by extending the rule body with an $\mathbf{S}_{in}$-atom containing x and with fresh variables in all other positions of the atom (there are multiple ways to do this, and we add all safe rules that can be obtained in this way). Then it follows from Claim 2 that P'' is equivalent to ϕ: take any $\mathbf{S}_{in}$-instance I and let I' be $\mathbf{S}$-instance extending I with all possible R-facts over $\text{adom}(I)$. Since S does not occur in ϕ, we have that $I \models \phi(a_1, \ldots, a_n)$ iff $I' \models \phi(a_1, \ldots, a_n)$ iff $R(a_1, \ldots, a_n) \in P'(I')$ iff $R(a_1, \ldots, a_n) \in P''(I)$. ◀