Two Programs, one Distribution? - On Distinguishing Structures of Probabilistic Logic Programs

Kilian Rückschloß [1], Felix Weitkämper [2]

Abstract: We present a criterion for two structures of probabilistic logic programs, i.e. programs with unspecified probabilities, to induce different sets of distributions. To this end we associate to each program structure a Datalog program, which we call the Pearl identifier. We then obtain that faithfully indistinguishable program structures yield equivalent Pearl identifiers. Moreover, we argue that the notion of faithful indistinguishability is sufficiently close to the notion of two program structures inducing the same set of distributions. Hence, we reduce our initial problem to the equivalence of Datalog programs, a semi-decidable problem.

Keywords: Probabilistic Logic Programming; Equivalence of Probabilistic Logic Programs; Faithful Indistinguishability; Markov Equivalence; Causal Structure Discovery

1 Introduction

In causal structure discovery one first learns a probabilistic logic program from observational data with the help of a structure learning algorithm. In the next step one tries to find all causal explanations for the distribution induced by the program **P**, i.e. one is interested in the space of all programs that yield the same distribution [RW22].

Example 1 *Assume for instance we are given a database of individuals (denoted Ind), which contains the information whether Ind_1 is a parent of Ind_2 denoted $parent(Ind_1, Ind_2)$. Further, we say that Ind_1 is an ancestor of Ind_N and write $ancestor(Ind_1, Ind_N)$ if there exists a chain $parent(Ind_1, Ind_2), parent(Ind_2, Ind_3), ..., parent(Ind_{N-1}, Ind_N)$. Finally, our data describes how wealth (denoted $wealth/1$) passes along the ancestor relation. From a structure learning algorithm we obtain that this mechanism is described by the following LPAD-clause RC_1 [VV03]:*

```
wealth(Ind2) : 0.7 :- wealth(Ind1), ancestor(Ind1,Ind2).
```

Hence, if Ind_1 is wealthy and Ind_1 is an ancestor of Ind_2, we expect Ind_2 to be wealthy with a probability of 0.7*. This reflects the intuition that wealth passes from ancestors to descendants. However, we obtained this description only from observations, so it is*

[1] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538 München, Deutschland kilian.rueckschloss@lmu.de

[2] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538 München, Deutschland felix.weitkaemper@lmu.de

generally not permissible to conclude that the LPAD-clause RC_1 represents the correct causal mechanism.

In our situation an alternative causal hypothesis would be that wealth passes from descendants to ancestors. In this case we would find a probability _ such that the following clause RC_2 induces the same probability distribution as RC_1:

```
wealth(Ind1) : _ :- wealth(Ind2), ancestor(Ind1,Ind2).
```

In this contribution we study the problem, whether two expressions like RC_2 in Example 1 induce the same set of distributions. This also extends work on the equivalence of Datalog programs [Ha01] and thus we believe that this problem is of interest in its own right.

Our main idea is to ground probabilistic logic programs to Bayesian networks. These Bayesian networks we can then investigate for faithful indistinguishability, a notion that is slightly more sensitive than the equivalence of the induced distributions. However, faithful indistinguishability can be decided considering only the graph underlying a Bayesian network. In this contribution we lift this graphical criterion to probabilistic logic programming in order to refute that two given program structures ground to families of faithfully indistinguishable Bayesian networks.

2 Preliminaries

Here, we recall the basic notions from the theory of Pearl's causal models and Bayesian networks. In order to apply this theory in our context we then introduce the FCM-semantics for probabilistic logic programming [RW22]. First, we recall the definition of a functional causal model [Pe00, 1.4.1]:

Definition 1 (Functional Causal Model) *A **functional causal model** or **causal model** $\mathcal{M}$ on a set of variables $\mathbf{V}$ is a system of equations that consists of one **defining equation** of the form $X := f_X(\mathrm{Pa}(X), \mathrm{error}(X))$ for each variable $X \in \mathbf{V}$. Here the **parents** $\mathrm{Pa}(X) \subseteq \mathbf{V}$ of X form a subset of the set of variables $\mathbf{V}$, the **error term** $\mathrm{error}(X)$ of X is a vector of random variables and f_X is a function defining X in terms of the parents $\mathrm{Pa}(X)$ and the error term $\mathrm{error}(X)$ of X.*

Example 2 *In the situation of Example 1 we obtain a causal model, which consists of the following equations:*

$$wealth(Ind_2) := \bigvee_{ancestor(Ind_1, Ind_2)} wealth(Ind_1) \wedge u(Ind_1, Ind_2)$$

Here $u(Ind1, Ind2)$ are our error terms that are true with probability 0.7. In particular we assume that they are distinct mutually independent random variables.

Bayesian networks are a widespread representation for probability distributions: A given distribution is stored in a directed acyclic graph G on the set of involved random variables, where each node is equipped with parameters that represent the distribution of the corresponding random variable conditioned on the set of its parents in G. We refer to [Pe00, §1.2.2] for a more detailed introduction into Bayesian networks. Further, let us recall the relation between causal models and Bayesian networks:

Definition 2 (Acyclic & Markovian Causal Models) *For each causal model $\mathcal{M}$ on $\mathbf{V}$ we define the **causal diagram** $\mathrm{graph}(\mathcal{M})$ to be the directed graph on $\mathbf{V}$, which is given by drawing an arrow $X \longrightarrow Y$ if and only if $X \in \mathrm{Pa}(Y)$. Further, we call $\mathcal{M}$ an **acyclic** causal model if its causal diagram $\mathrm{graph}(\mathcal{M})$ is a directed acyclic graph. Finally, an acyclic causal model is said to be **Markovian** if the error terms $\mathrm{error}(X)$ are mutually independent for all $X \in \mathbf{V}$.*

We find that every Markovian causal model $\mathcal{M}$ gives rise to a probability distribution that can be stored in a Bayesian network on its causal diagram $\mathrm{graph}(\mathcal{M})$ [Pe00, 1.4.2].

Example 3 *In the situation of Example 2 we assume that our database contains four individuals $p1 - p4$ with $parent(p3, p4)$, $parent(p2, p4)$ and $parent(p1, p2)$. In this we obtain the following causal diagram:*

$$\begin{tikzcd}
 & wealth(p2) \arrow[dl] & \\
wealth(p4) & & wealth(p1) \arrow[ul] \arrow[ll] \\
 & wealth(p3) \arrow[ul] &
\end{tikzcd}$$

Moreover, it is easy to see that the causal model of Example 2 is Markovian, i.e. the induced distribution can be stored in a Bayesian network on the graph above.

The graph underlying a Bayesian network gives rise to a conditional independence oracle: Two random variables A, B are independent conditioned on a subset $\mathbf{Z}$ of random variables if they are **d-separated**, i.e. if there exists no d-connecting path between them in the underlying graph G. Here a d-connecting or directionally connecting path is a special kind of undirected path in the graph G [GVP90]. As it turns out, d-separation yields a correct [VP90a] but incomplete independence oracle [Mc95, 1.4]. If a distribution π can be stored in a Bayesian network on a graph G such that d-separation on G yields a complete conditional independence oracle, we say that the distribution π is **(causally) faithful** to the graph G. Fortunately, we find that faithfulness holds for almost all discrete Bayesian networks in the following sense [Me95]:

Theorem 1 (Obstruction to Causal Faithfulness) *Let G be a directed acyclic graph and let $\theta \in [0,1]^n$ be a vector which determines the conditional distributions that turn G into a discrete Bayesian network representing the distribution π. In this case we obtain finitely many non-trivial polynomial equations such that π is faithful to G unless θ solves one of these equations.*

For an in-depth discussion of causal faithfulness we refer the reader to [SGS00] and [We18]. Further, two directed acyclic graphs $G_{1/2}$ are said to be **faithfully indistinguishable** if every distribution faithful to G_1 is also faithful to G_2. Given two faithfully indistinguishable graphs $G_{1/2}$, Theorem 1 tells us that the set of parameters for G_1, which yield to a distribution that cannot be represented in a Bayesian network on G_2, is a set of Lebesque-measure zero in the parameter space. In this sense, faithful indistinguishability approximates the notion of two graphs representing the same set of distributions. Finally, we give a criterion to check two directed acyclic graphs for faithful indistinguishability [VP90b, Theorem 1]:

Theorem 2 *Let $G_{1/2}$ be two directed acyclic graphs on a set of random variables $\mathbf{V}$. In this case the graphs $G_{1/2}$ are faithfully indistinguishable if and only if they share the same adjacencies and the same unshielded colliders.*

Here, two nodes A, B in a directed graph G are said to be **adjacent** if there exists an edge $A \rightarrow B$ or an edge $A \leftarrow B$. Moreover, an **unshielded collider** is a subgraph $A \rightarrow B \leftarrow C$, where A and C are not adjacent.

Next, we give a semantics for probabilistic logic programs in terms of causal models. This allows us to ground a program to a Bayesian network. Before giving our semantics we shall fix the syntax we are working with.

Let us fix a **query language**, i.e. a language $\mathfrak{Q} \supseteq \mathfrak{L} \supseteq \mathfrak{E}$ with equality $\doteq$ in three parts with an **external vocabulary** $\mathfrak{E}$ and a **logical vocabulary** $\mathfrak{L}$. Here $\mathfrak{Q}$ is a finite multisorted relational vocabulary, i.e. it consists of a finite set of sorts, a finite set of relation symbols and a finite set of constants in those sorts, as well as a countably infinite set of variables in every sort. For every variable or constant x we will denote its sort by $\mathfrak{s}(x)$. Further, $\mathfrak{L}$ is a subvocabulary of $\mathfrak{Q}$, which contains all of the sorts, variables and constants of $\mathfrak{Q}$ as well as a (possibly empty) subset of the relation symbols of $\mathfrak{Q}$. Moreover, $\mathfrak{E}$ is a subvocabulary of $\mathfrak{L}$, which satisfies the same properties in $\mathfrak{L}$ as $\mathfrak{L}$ does regarding $\mathfrak{Q}$.

Example 4 *To model the situation in Example 1 we need a query language $\mathfrak{Q}$ that consists of one sort $\mathfrak{ind}$ and no constants. The external alphabet $\mathfrak{E}$ is given by the predicate $parent/2$, the logical alphabet is given by the predicates $parent/2$ and $ancestor/2$. Finally, the query language $\mathfrak{Q}$ is given by the predicates $parent/2$, $ancestor/2$ and $wealth/1$.*

As usual, an **atom** is an expression of the form $r(t_1, \dots, t_n)$ or $t_1 \doteq t_2$, where r is a relation symbol and t_1 to t_n are constants or variables of the correct sorts and a **literal** is an expression of the form A or $\neg A$ for an atom A. It is called an **external atom** or **literal** if r is in $\mathfrak{E}$, **logical atom** or **literal** if r is in $\mathfrak{L}$, **internal atom** or **literal** if r lies in $\mathfrak{L} \setminus \mathfrak{E}$ and a **random atom** or **literal** if r is in $\mathfrak{Q} \setminus \mathfrak{L}$. A literal of the form A is called **positive** and a literal of the form $\neg A$ is called **negative**.

Formulas, as well as **existential** and **universal formulas** are defined as usual in first-order logic. Moreover, we will use the operator var(_) to refer to the variables in an expression. A **substitution** is a function θ that assigns to each variable in a set $\mathrm{dom}(\theta)$ a variable or constant of the same sort. We say that θ is a substitution on $\mathbf{V}$ is $\mathbf{V} \subseteq \mathrm{dom}(\theta)$. Substitutions are extended to formulas as usual in first-order logic.

The logical vocabulary will be used to formulate constraints and conditions for our probabilistic logic program. The purpose of a probabilistic logic program, however, is to define distributions for the random variables, determined by the query language $\mathfrak{Q}$. This is done by so called random clauses, i.e. LPAD-clauses with one random atom in the head [VV03].

Definition 3 (Random Clause) *A **generalized random clause** RC is an expression*

$$(R : \_ \leftarrow R_1, ..., R_m, L_1, ..., L_n) = (\mathrm{effect}(RC) : \_ \leftarrow \mathrm{causes}(RC) \cup \mathrm{cond}(RC))\,,$$

*which is given by a random atom $R := \mathrm{effect}(RC)$, called the **effect** of RC, a finite set of random literals $\mathrm{causes}(RC) := \{R_1, ..., R_m\}$, called the **causes** of RC and a finite set of logical literals $\mathrm{cond}(RC) := \{L_1, ..., L_n\}$, called the **condition** of RC with no variable occurring twice. Further, we obtain a **random clause** $RC^\pi := (R : \pi(RC) \leftarrow R_1, ..., R_m, L_1, ..., L_n)$ from the generalized random clause RC by choosing a **probability** $\pi(RC) \in [0, 1]$.*

Note that the assumption that no variable occurs twice in a generalized random clause is not a restriction since we can express coinciding variable names as equality:

Example 5 *In the introduction we need to rewrite the generalized random clause RC_1 as*

```
wealth(Ind1) : _ :- wealth(Ind2), ancestor(Ind3,Ind4), Ind1 = Ind4.
```

From a query alphabet we construct program structures and (lifted) programs as follows:

Definition 4 (Lifted Program & Program Structure) *A **program structure** $\boldsymbol{P} := (\boldsymbol{R}, \boldsymbol{I}, \boldsymbol{C})$ is a triple, which consists of a finite set of integrity constraints $\boldsymbol{C}$ of the form $\bot \leftarrow L_1, ..., L_k$ for logical literals L_i with $1 \leq i \leq k$, a finite set of normal clauses $\boldsymbol{I}$ of the from $H \leftarrow B_1, ..., B_m$ with logical literals $B_1, ..., B_m$ and an internal atom H, and a finite set of generalized random clauses $\boldsymbol{R}$. We call $\boldsymbol{C}$ the **constraints**, $\boldsymbol{I}$ the **internal part** and $\boldsymbol{R}$ the **random part** of $\boldsymbol{P}$. Finally, we say that $\boldsymbol{P}$ is **stratified** if its internal part $\boldsymbol{I}$ is a stratified set of normal clauses.*

*A **choice of parameters** for the program structure $\boldsymbol{P}$ is a function $\pi : \boldsymbol{R} \to [0, 1]$ that assigns a probability to each generalized random clause in the random part of $\boldsymbol{P}$. The program structure $\boldsymbol{P}$ and a choice of parameters π yield a **(lifted) program** $\boldsymbol{P}^\pi := (\boldsymbol{R}^\pi, \boldsymbol{I}, \boldsymbol{C})$, where*

$$\boldsymbol{R}^\pi := \{\mathrm{effect}(RC) : \pi(RC) \leftarrow \mathrm{causes}(RC) \cup \mathrm{cond}(RC) \mid RC \in \boldsymbol{R}\}.$$

Example 6 *We aim to represent the causal model in Example 2 by a program which is obtained from the following program structure $\boldsymbol{P}_1$ by choosing probability 0.7 for its only generalized random clause:*

```
%random part
wealth(Ind1) : _ :- wealth(Ind2), ancestor(Ind3,Ind4), Ind1 = Ind4.
%internal part
ancestor(Ind1,Ind2) :- parent(Ind1,Ind2).
ancestor(Ind1,Ind3) :- ancestor(Ind2,Ind3), parent(Ind1,Ind2).
%constraints
false :- ancestor(Ind1,Ind2), ancestor(Ind2,Ind1).
```

Here, the constraint reflects that there are no cycles in family trees. The second hypothesis for the causal mechanism in Example 1 can be formulated by the program structure $\boldsymbol{P}_2$, which is obtained by replacing the random part of the program above with the generalized random clause RC_2.

After having established the necessary syntax we introduce the corresponding semantics. Let us begin with the logical expressions. We define $\mathfrak{L}$-structures as usual in first-order logic. The semantics of logical expressions are given in a straightforward way under the usage of the unique names assumption, i.e. different constants denote different individuals. Whether a logical formula is **satisfied** by a given $\mathfrak{L}$-structure (under a given **interpretation** of its free variables) is determined by the usual rules of first-order logic. Finally, note that the semantics of external expressions is defined analogously.

Notation 1 *Let $\boldsymbol{P} := (\boldsymbol{R}, \boldsymbol{I}, \boldsymbol{C})$ be a stratified program structure and let $\mathcal{E}$ be an $\mathfrak{E}$-structure. Due to the unique names assumption we may assume without loss of generality that $\mathcal{E}$ is a Herbrand model with respect to an alphabet $\mathfrak{E}^*$, which extends the external alphabet $\mathfrak{E}$ by constants. Further, denote by $\mathfrak{L}^*$ respectively $\mathfrak{Q}^*$ the extension of the languages $\mathfrak{L}$ and $\mathfrak{Q}$ by the new constants in $\mathfrak{E}^*$. We write $\mathcal{E}^{\boldsymbol{I}} := \{L \text{ ground atom of } \mathfrak{L}^* : \boldsymbol{I} \cup \mathcal{E} \models L\}$ for the minimal Herbrand model of $\boldsymbol{I} \cup \mathcal{E}$ [Br07]. This is also the result of applying the stratified Datalog program $\boldsymbol{I}$ to $\mathcal{E}$.*

*An $\mathfrak{E}$-structure $\mathcal{E}$ is called an **external database** for the program structure $\boldsymbol{P}$ if we find that $\mathcal{E}^{\boldsymbol{I}} \models \boldsymbol{C}$. A **ground variable** is a ground atom $G := r(x_1, ..., x_n)$ of $\mathfrak{Q}^*$ with a random predicate $r \in \mathfrak{Q}$. Finally, we write $\mathcal{G}(\mathcal{E})$ for the set of all ground variables.*

The term ground variable indicates that we expect G to become a proper random variable under our semantics. To obtain a well-defined semantics, we need the following assumption:

Assumption 1 *From now on we fix a stratified program structure $\boldsymbol{P} := (\boldsymbol{R}, \boldsymbol{I}, \boldsymbol{C})$ in a query language $\mathfrak{Q} \supseteq \mathfrak{L} \supseteq \mathfrak{E}$.*

Example 7 *Note that the $\mathfrak{E}$-structure $\mathcal{E}$ described in Example 3 yields an external database for the program structure $\boldsymbol{P}_1$ of Example 6.*

Now we are in the position to define the desired semantics:

Definition 5 (FCM-Semantics for Lifted Programs) *Let π be a choice of parameters and let $\mathcal{E}$ be an external database for $\boldsymbol{P}$. The* ***grounding*** $\boldsymbol{P}^{(\pi,\mathcal{E})}$ *of the program structure $\boldsymbol{P}$ with respect to π and $\mathcal{E}$ is the Boolean functional causal model on the set of ground variables $\mathcal{G}(\mathcal{E})$, which is given by the following equation for every ground variable $G \in \mathcal{G}(\mathcal{E})$:*

$$G := \bigvee_{\substack{RC \in \boldsymbol{R} \\ \kappa \textit{ interpretation on } \operatorname{var}(RC) \\ \operatorname{effect}(RC)^{\kappa} = G \\ \left(\mathcal{E}^{\boldsymbol{I}}, \kappa\right) \models \operatorname{cond}(RC)}} \left(\bigwedge_{C \in \operatorname{causes}(RC)} C^{\kappa} \wedge u\,(RC, \kappa) \right). \tag{1}$$

Here we have that the ***error term*** *$u\,(RC, \kappa)$ is a distinct Boolean random variable that is true with probability $\pi(RC)$ for every random clause $RC \in \boldsymbol{R}$ and every variable interpretation κ on $\operatorname{var}(RC)$. In addition, the error terms $u\,(RC, \kappa)$ are assumed to be mutually independent.*

Example 8 *As intended the causal model of Example 2 is the grounding of the program structure $\boldsymbol{P}_1$ in Example 6 with respect to the given choice of parameters and the external database described in Example 3.*

Further, the ground graph yields the causal diagram of the grounding of a program structure:

Definition 6 (Ground Graph & Acyclicity) *Let $\mathcal{E}$ be an external database for $\boldsymbol{P}$. We define the* ***ground graph*** $\operatorname{Graph}_{\mathcal{E}}(\boldsymbol{P})$ *to be the directed graph on the set of ground variables $\mathcal{G}(\mathcal{E})$, which is obtained by drawing an edge $G_1 \to G_2$ if and only if there exists a generalized random clause $RC \in \boldsymbol{R}$, a cause $C \in \operatorname{causes}(RC)$ and a variable interpretation ι on $\operatorname{var}(C) \cup \operatorname{var}(\operatorname{effect}(RC))$ such that $\left(\mathcal{E}^{\boldsymbol{I}}, \iota\right) \models \exists_{\operatorname{var}(\operatorname{cond}(RC)) \setminus (\operatorname{var}(C) \cup \operatorname{var}(\operatorname{effect}(RC)))} \operatorname{cond}(RC)$, $C^{\iota} \in \{G_1, \neg G_1\}$ and such that $\operatorname{effect}(RC)^{\iota} = G_2$. In this case we say that the edge $G_1 \to G_2$ is* ***induced*** *by RC. Moreover, we call $\boldsymbol{P}$ an* ***acyclic*** *program structure if $\operatorname{Graph}_{\mathcal{E}}(\boldsymbol{P})$ is a directed acyclic graph for every external database $\mathcal{E}$.*

From the definition of the ground graph we immediately obtain that for every choice of parameters π of an acyclic program structure $\mathbf{P}$ and for every external database $\mathcal{E}$ the causal model $\mathbf{P}^{(\pi,\mathcal{E})}$ is Markovian with the ground graph $\operatorname{Graph}_{\mathcal{E}}(\mathbf{P})$ as causal diagram.

Example 9 *The graph in Example 3 yields the ground graph of the program structure $\boldsymbol{P}_1$ in Example 6 with respect to the described external database $\mathcal{E}$. Further, note that the* *corresponding ground graph of the program structure* $\boldsymbol{P}_2$ *is obtained by reversing all arrows.*

Assumption 2 *From now on we assume the program structure* $\boldsymbol{P} := (\boldsymbol{R}, \boldsymbol{I}, \boldsymbol{C})$ *to be acyclic.*

In this case it is easy to see that the grounding $\mathbf{P}^{(\pi,\mathcal{E})}$ yields a unique distribution on the ground variables $\mathcal{G}(\mathcal{E})$, which is shown to consistently generalize the distribution semantics [Ri20, Chapter 1 §2.1] in [RW22].

Proposition 1 *For every external database* $\mathcal{E}$ *and for every choice of parameters* π *the grounding* $\boldsymbol{P}^{(\pi,\mathcal{E})}$ *yields a distribution, which can be stored in a Boolean Bayesian network on the ground graph* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})$. □

3 Main Results

Our initial goal was to study the notion of equivalence for program structures, i.e. we want to refute that two program structures $\mathbf{P}_{1/2}$ induce the same set of distributions. However, here we restrict ourselves to the study of faithful indistinguishability:

Definition 7 (Faithfulness and Faithful Indistinguishability) *A program structure* $\boldsymbol{P}$ *is said to be* ***faithful*** *with respect to a choice of parameters* π *and an external database* $\mathcal{E}$ *if the induced distribution is faithful to the ground graph* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})$ *of* $\boldsymbol{P}$.

Let $\boldsymbol{P}_{1/2}$ *be two programs structures sharing the same external databases. Further, let* $\mathcal{E}$ *be any external database and let* π_1 *be a choice of parameters arbitrarily chosen with the property that* $\boldsymbol{P}_1$ *is faithful with respect to* π_1 *and* $\mathcal{E}$*. We call* $\boldsymbol{P}_{1/2}$ ***faithfully indistinguishable*** *if we can choose a choice of parameters* π_2 *such that the causal models* $\boldsymbol{P}_{1/2}^{(\pi_{1/2},\mathcal{E})}$ *induce the same distribution and such that* $\boldsymbol{P}_2$ *is also faithful with respect to* π_2 *and* $\mathcal{E}$*.*

Note that in light of Theorem 1 the gap between faithful indistinguishability and equivalence of program structures does not seem too big. Further, to use Theorem 2 we need the following assumption, which also seems harmless considering Theorem 1:

Assumption 3 *From now on all mentioned program structures* $\boldsymbol{P}$ *satisfy Assumption 1 and Assumption 2. Moreover, for every external database* $\mathcal{E}$ *there exists a choice of parameters* π *such that* $\boldsymbol{P}$ *is faithful with respect to* π *and* $\mathcal{E}$*.*

Remark 1 *A precise justification of our restriction to faithful indistinguishability and of Assumption 3 requires the generalization of Theorem 1 to the context of probabilistic logic programming, which we propose as an interesting direction for future work.*

Next, we invoke Assumption 3 and Theorem 2 to obtain the following result, which becomes our key tool for distinguishing program structures:

Proposition 2 *If two program structures $\boldsymbol{P}_{1/2}$ are faithfully indistinguishable, for every external database the ground graphs* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P}_{1/2})$ *share the same adjacencies and the same unshielded colliders.* □

Let us now state the problem, which is solved in this work:

Problem 1 *Given two program structures $\boldsymbol{P}_{1/2}$ sharing the same external databases, we aim to refute that $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ are faithfully indistinguishable.*

With Proposition 2 we can correctly reformulate Problem 1 to:

Problem 2 *Given two program structures $\boldsymbol{P}_{1/2}$ sharing the same external databases, we aim to refute that for every external database $\mathcal{E}$ the ground graphs* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P}_{1/2})$ *share the same adjacencies and the same unshielded colliders.*

Example 10 *Proposition 2 yields that the program structures $\boldsymbol{P}_{1/2}$ of Example 6 are not faithfully indistinguishable: The ground graph* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P}_1)$ *of $\boldsymbol{P}_1$ in Example 3 with respect to the external database $\mathcal{E}$ of Example 7 has an unshielded collider at $wealth(p4)$, which disappears after reversing the arrows. Hence, we can conclude with Example 9 that the ground graphs* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P}_{1/2})$ *do not share the same unshielded colliders.*

In order to generalize the reasoning of Example 10 we need to classify all adjacencies and all unshielded colliders that may occur in a ground graph of a program structure. To do so it is for sure sufficient to classify all subgraphs of cardinality three that may occur in a ground graph of a program structure.

More generally, we even aim to classify all subgraphs of cardinality $n \in \mathbb{N}_{\geq 1}$ that may occur in the ground graphs of a given acyclic, stratified program structure $\mathbf{P} := (\mathbf{R}, \mathbf{I}, \mathbf{C})$. We introduce the n-th generic dependence graph as the classifying object for these subgraphs:

Definition 8 (n-th Generic Dependence Graph) *Let $n \in \mathbb{N}_{\geq 1}$ be a natural number. For every sort $\mathfrak{s}$ of $\mathfrak{Q}$ we denote by $k(\mathfrak{s})$ the maximal number of occurrences of $\mathfrak{s}$ in the arity of any random predicate of $\mathfrak{Q}$. Further, we fix $m(\mathfrak{s}) = k(\mathfrak{s}) \cdot n$ distinct variables $S_1, ..., S_{m(\mathfrak{s})}$ for every sort $\mathfrak{s}$ of $\mathfrak{Q}$ and set $\mathcal{V}_n^{\mathfrak{s}} := \{S_1, ..., S_{m(\mathfrak{s})}\}$. Finally, we denote by $\mathcal{R}_n(\mathfrak{Q})$ the set of all random atoms R of $\mathfrak{Q}$ with* $\mathrm{var}(R) \subseteq \bigcup_{\mathfrak{s}\ sort} \mathcal{V}_n^{\mathfrak{s}}$ *in which no variable occurs twice.*

*The **n-th generic dependence graph** n-* $\mathrm{Graph}(\boldsymbol{P})$ *is the directed labelled multigraph on the set of random atoms $\mathcal{R}_n(\mathfrak{Q})$ which is obtained by the following procedure:*

For every generalized random clause $RC \in \boldsymbol{R}$, for every cause $C \in \mathrm{causes}(RC)$ of RC and for each substitution θ on $var(C) \cup \mathrm{var}(\mathrm{effect}(RC))$ such that C^θ and $\mathrm{effect}(RC)^\theta$ share no common variables we draw an edge $E := \left(R^\theta \to \mathrm{effect}(RC)^\theta\right)$ with label $\lambda(E) := (RC, C, \theta)$, where R is the random atom underlying $C \in \{R, \neg R\}$. An edge of this kind is said to be ***induced*** *by RC, C and θ. Moreover, we associate to the edge E its* ***projection condition*** $\mathrm{p\_cond}\,(E) = \left(\exists_{\mathrm{cond}(RC)\setminus(\mathrm{var}(\mathrm{effect}(RC))\cup\mathrm{var}(C))}\,\mathrm{cond}(RC)\right)^\theta$.

Finally, let $E := R \to S$ be an edge of the n-th generic dependence graph $n\text{-}\mathrm{Graph}(\boldsymbol{P})$ and let ι be an interpretation on $\mathrm{var}(R) \cup \mathrm{var}(S)$ in an external database $\mathcal{E}$. We say that E ***projects*** *with respect to ι to an edge $E^\iota = (R^\iota \to S^\iota)$ if we find that $(\mathcal{E}, \iota) \models \mathrm{p\_cond}(E)$.*

For a subgraph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ of the n-th generic dependence graph $n\text{-}\mathrm{Graph}(\boldsymbol{P})$ we define its ***projecting condition*** *by $\mathrm{p\_cond}(\mathbb{G}) := \bigcup_{E \in \mathbb{E}} \mathrm{p\_cond}(E)$. Further, let ι be an interpretation on the variables occurring in the random atoms of $\mathbb{V}$ in an external database $\mathcal{E}$. We say that ι* ***projects*** *$\mathbb{G}$ if every edge $E \in \mathbb{E}$ projects, i.e. if $(\mathcal{E}, \iota) \models \mathrm{p\_cond}(\mathbb{G})$.*

Example 11 *The third generic dependence graph $3\text{-}\mathrm{Graph}(\boldsymbol{P}_1)$ of the program structure $\boldsymbol{P}_1$ in Example 6 is given by three nodes $wealth(Ind_1)$-$wealth(Ind_3)$, where each pair of nodes is connected by an edge $E := wealth(Ind_i) \to wealth(Ind_j)$ for $i \neq j$ with label $\lambda(E) := (RC_1, ancestor(Ind_i, Ind_j), \theta)$ with the substitution $\theta(Ind_1) = Ind_i$ and $\theta(Ind_2) = Ind_j$. Finally, observe that the third generic dependence graph $3\text{-}\mathrm{Graph}(\boldsymbol{P}_2)$ of the program structure $\boldsymbol{P}_2$ is obtained by reversing the directions of the edges in $3\text{-}\mathrm{Graph}(\boldsymbol{P}_1)$.*

From Definition 6 we immediately obtain the following result:

Proposition 3 *Let $\mathbb{G} := (\mathbb{V}, \mathbb{E})$ be a subgraph of the n-th generic dependence graph $n\text{-}\mathrm{Graph}(\boldsymbol{P})$, which projects with respect to an interpretation ι in an external database $\mathcal{E}$ on the variables occurring in a random atom of $\mathbb{V}$. In this case we obtain a morphism of directed graphs $\mathbb{G} \xrightarrow{\iota} \mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})$, $V \mapsto V^\iota$.* □

Definition 9 *In the situation of Proposition 3 we call the map $\mathbb{G} \xrightarrow{\iota} \mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})$ the* ***projection*** *of the subgraph $\mathbb{G}$ with respect to ι. Further, we say that $\mathbb{G}$ grounds with respect to ι if the projection yields a monomorphism of directed graphs. In this case we also say that $\mathbb{G}$* ***grounds*** *to the subgraph $\mathrm{im}(\mathbb{G} \xrightarrow{\iota} \mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})) \subseteq \mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})$ and that $\mathrm{im}(\mathbb{G} \xrightarrow{\iota} \mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P}))$* ***lifts*** *to $\mathbb{G}$.*

We obtain the following logical condition for a subgraph of the n-th generic dependence graph to ground:

Definition 10 *For a subgraph* $\mathbb{G} := (\mathbb{V}, \mathbb{E})$ *of the n-th generic dependence graph* $n\text{-Graph}(\boldsymbol{P})$ *we define its* ***grounding condition*** *by*

$$\text{g\_cond}(\mathbb{G}) := \text{p\_cond}(\mathbb{G}) \cup \left\{ \bigwedge_{\substack{r(X_1,\ldots,X_k)\in\mathbb{V} \\ r(Y_1,\ldots,Y_k)\in\mathbb{V}}} \left(\bigvee_{i=1}^{k} \neg(X_i \doteq Y_i) \right) \right\}.$$

As desired we obtain the following results that classify the subgraphs of cardinality n that may occur in the ground graph of the program structure $\mathbf{P}$:

Theorem 3 (Grounding in the n-th Generic Dependence Graph) *Fix* $n \in \mathbb{N}_{\geq 1}$ *and a subgraph* $\mathbb{G} := (\mathbb{V}, \mathbb{E})$ *of the n-th generic dependence graph* $n\text{-Graph}(\boldsymbol{P})$. *Let* ι *be an interpretation in an external database* $\mathcal{E}$ *on the variables occurring in a random atom of* $\mathbb{V}$. *We find that* $\mathbb{G}$ *grounds with respect to* ι *if and only if* $(\mathcal{E}, \iota) \models \text{g\_cond}(\mathbb{G})$.

Theorem 4 (Lifting to the n-th Generic Dependence Graph) *Fix* $n \in \mathbb{N}_{\geq 1}$. *Let* $\mathcal{E}$ *be an external database and let G be a subgraph of the ground graph* $\text{Graph}_{\mathcal{E}}(\boldsymbol{P})$ *with not more than n nodes. In this case we find a subgraph* $\mathbb{G}$ *of the n-th generic dependence graph* $n\text{-Graph}(\boldsymbol{P})$ *such that no variable occurs twice in the nodes of* $\mathbb{G}$ *and such that G lifts to* $\mathbb{G}$. *Obviously, we obtain all those lifts of G by relabelling the variables occurring in* $\mathbb{G}$.

Finally, we apply the Theorems 3 and 4 to measure the adjacencies and the unshielded colliders that may occur in the ground graphs of a given acyclic, stratified program structure $\mathbf{P} := (\mathbf{R}, \mathbf{I}, \mathbf{C})$ with the help of a Datalog program.

Definition 11 (Pearl Identifier) *The* ***Pearl identifier*** $\text{ID}(\boldsymbol{P})$ *of the program structure* $\boldsymbol{P}$ *is the Datalog program, which is given by the internal part* $\boldsymbol{I}$, *the constraints* $\boldsymbol{C}$ *and the following clauses:*

$adjacent(R, S) \leftarrow \text{g\_cond}(R \rightarrow S)$ *for every edge* $R \rightarrow S$ *in the second generic dependence graph* $2\text{- Graph}(\boldsymbol{P})$ *with R and S not sharing common variables*

$adjacent(S, R) \leftarrow \text{g\_cond}(R \rightarrow S)$ *for every edge* $R \rightarrow S$ *in the second generic dependence graph* $2\text{- Graph}(\boldsymbol{P})$ *with R and S not sharing common variables*

$immorality(B, C, D) \leftarrow \text{g\_cond}(\mathbb{G}), \neg\,\text{p\_cond}(E_1), \ldots, \neg\,\text{p\_cond}(E_n)$ *for every collider* $\mathbb{G} := (B \rightarrow C \leftarrow D)$ *in the third generic dependence graph* $3\text{- Graph}(\boldsymbol{P})$ *and edges* $E_1,\ldots,E_n$ *shielding the collider* $\mathbb{G}$ *in the third generic dependence graph* $3\text{- Graph}(\boldsymbol{P})$ *such that the nodes* A, B, C *do not share common variables.*

Note that at this place by abuse of notation we silently omit the existential quantifiers in the expressions g_cond(_) *and* p_cond(_).

Example 12 *The Pearl identifier of the program structure $\boldsymbol{P}_1$ in Example 6 is given by the following Datalog program:*

```
adjacent(Ind1, Ind2) :- ancestor(Ind1,Ind2).
adjacent(Ind2, Ind1) :- ancestor(Ind1,Ind2).
immorality(Ind1, Ind2, Ind3) :- ancestor(Ind1,Ind2), ancestor(Ind3,Ind2),
        \+ancestor(Ind1,Ind3),\+ancestor(Ind3,Ind1).
ancestor(Ind1,Ind2) :- parent(Ind1,Ind2).
ancestor(Ind1,Ind3) :- ancestor(Ind2,Ind3), parent(Ind1,Ind2).
false :- ancestor(Ind1,Ind2), ancestor(Ind2,Ind1).
```

As desired Theorem 2, Theorem 3 and Theorem 4 yield the following result:

Theorem 5 *If two program structures $\boldsymbol{P}_{1/2}$ are faithfully indistinguishable, their Pearl identifiers* $\mathrm{ID}(\boldsymbol{P}_{1/2})$ *yield the same answers to the Datalog queries $adjacent(X_1, X_2)$, $immorality(X_1, X_2, X_3)$ and $\perp$.*

Hence, if we want to refute the faithful indistinguishability of two program structures $\mathbf{P}_{1/2}$, we compute their Pearl identifiers $\mathrm{ID}(\mathbf{P}_{1/2})$ and prove that they yield non-equivalent Datalog programs.

4 Conclusion

First, we lift the notion of causal faithfulness to structures of probabilistic logic programs. Further, for each program structure we construct the Pearl identifier, a Datalog program that classifies the adjacencies and the unshielded colliders occurring in the ground graphs. Together with the theory of Bayesian networks we then reduce the task of refuting the faithful indistinguishability of program structures to the equivalence of Datalog programs: Two program structures are not faithfully indistinguishable if their Pearl identifiers yield non-equivalent Datalog programs.

In order to precisely justify our restriction to the study of faithful indistinguishability we still need to investigate causal faithfulness of program structures in greater detail. In particular, we require an analogue of Theorem 1 in probabilistic logic programming. Furthermore, we also aim to establish the converse of Theorem 5, i.e. we want to conclude faithful indistinguishability of program structures from the equivalence of the corresponding Pearl identifiers.

Acknowledgements

The research leading to this publication was supported by LMUexcellent, funded by the Federal Ministry of Education and Research (BMBF) and the Free State of Bavaria under the Excellence Strategy of the Federal Government and the Länder.

The authors also wish to thank Kailin Sun for proof-reading the text before submission.

Bibliography

[Br07] Bry, François; Eisinger, Norbert; Eiter, Thomas; Furche, Tim; Gottlob, Georg; Ley, Clemens; Linse, Benedikt; Pichler, Reinhard; Wei, Fang: Foundations of Rule-Based Query Answering. In: Reasoning Web: Third International Summer School 2007, Dresden, Germany, September 3-7, 2007, Tutorial Lectures. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–153, 2007.

[GVP90] Geiger, D.; Verma, T.; Pearl, J.: Identifying Independence in Bayesian Networks. Networks, 20(5):507–534, 1990.

[Ha01] Halevy, Alon Y.; Mumick, Inderpal Singh; Sagiv, Yehoshua; Shmueli, Oded: Static Analysis in Datalog Extensions. J. ACM, 48(5):971–1012, sep 2001.

[Mc95] McDermott, Michael: Redundant Causation. The British Journal for the Philosophy of Science, 46(4):523–544, 1995.

[Me95] Meek, Christopher: Strong Completeness and Faithfulness in Bayesian Networks. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. UAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 411–418, 1995.

[Pe00] Pearl, J.: Causality. Cambridge University Press, Cambridge, UK, 2 edition, 2000.

[Ri20] Riguzzi, F.: Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning. River Publishers, 2020.

[RW22] Rückschloß, Kilian; Weitkämper, Felix: Exploiting the Full Power of Pearl's Causality in Probabilistic Logic Programming. In: Proceedings of the International Conference on Logic Programming 2022 Workshops co-located with the 38th International Conference on Logic Programming (ICLP 2022), Haifa, Israel, July 31st - August 1st, 2022. volume 3193 of CEUR Workshop Proceedings. CEUR-WS.org, 2022.

[SGS00] Spirtes, P.; Glymour, C.; Scheines, R.: Causation, Prediction, and Search. MIT press, 2nd edition, 2000.

[VP90a] Verma, T.; Pearl, J.: Causal Networks: Semantics and Expressiveness. In: Uncertainty in Artificial Intelligence, volume 9 of Machine Intelligence and Pattern Recognition, pp. 69–76. North-Holland, 1990.

[VP90b] Verma, T.; Pearl, J.: Equivalence and Synthesis of Causal Models. In: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence. UAI '90, Elsevier Science Inc., USA, p. 255–270, 1990.

[VV03] Vennekens, J.; Verbaeten, S.: Logic Programs with Annotated Disjunctions. Technical Report CW 368, Katholieke Universiteit Leuven, Department of Computer Science, 2003.

[We18] Weinberger, Naftali: Faithfulness, Coordination and Causal Coincidences. Erkenntnis, 83(2):113–133, 2018.

Proof Appendix

Fix an acyclic stratified program structure $\mathbf{P} := (\mathbf{R}, \mathbf{I}, \mathbf{C})$ together with an external database $\mathcal{E}$ and a natural number $n \in \mathbb{N}_{\geq 1}$. Here, we do not directly project from the n-th generic dependence graph n - Graph$(\mathbf{P})$ to the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P})$. In the interest of a transparent presentation we introduce the meta ground graph as a book keeping object between the n-th generic dependence graph n-Graph$(\mathbf{P})$ and the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P})$.

Definition 12 (Meta Ground Graph) *The* ***meta ground graph*** $\mathrm{Graph}^{\mathcal{E}}(\boldsymbol{P})$ *is the directed labelled multigraph on the set of ground variables $\mathcal{G}(\mathcal{E})$, which is obtained by the following procedure:*

For every generalized random clause $RC \in \boldsymbol{R}$, for every cause $C \in \mathrm{causes}(RC)$ of RC and for every variable interpretation ι on $var(C) \cup \mathrm{var}(\mathrm{effect}(RC))$ we draw an edge $E := (R^{\iota} \to \mathrm{effect}(RC)^{\iota})$ with the label $\lambda(E) := (RC, C, \iota)$ where $C \in \{R, \neg R\}$ if and only if we find that $(\mathcal{E}^{\boldsymbol{I}}, \iota) \models \exists_{\mathrm{cond}(RC)\setminus(\mathrm{var}(\mathrm{effect}(RC))\cup\mathrm{var}(C))} \mathrm{cond}(RC)$. An edge of this kind is said to be ***induced*** *by RC, C and ι.*

Remark 2 *By construction each edge in the meta ground graph is uniquely determined by its label.*

Next, we investigate how subgraphs of the meta ground graph $\mathrm{Graph}^{\mathcal{E}}(\mathbf{P})$ correspond to subgraphs of the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P})$.

Proposition 4 *For two ground variables $G_{1/2} \in \mathcal{G}(\mathcal{E})$ the following statements are equivalent:*

a) *The edge $G_1 \to G_2$ exists in the ground graph* $\mathrm{Graph}_{\mathcal{E}}(\boldsymbol{P})$.

b) *There exists an edge $G_1 \to G_2$ in the meta ground graph* $\mathrm{Graph}^{\mathcal{E}}(\boldsymbol{P})$.

Proof.

a) $\Rightarrow$ b) Assume we find the edge $G_1 \to G_2$ in the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P})$. In this case this edge is necessarily induced by a generalized random clause $RC \in \mathbf{R}$, i.e. there exists a cause $C \in \mathrm{causes}(RC)$ and a variable interpretation ι on $\mathrm{var}(C) \cup \mathrm{var}(\mathrm{effect}(RC))$ such that the following assertions hold:

i) $(\mathcal{E}^{\mathbf{I}}, \iota) \models \exists_{\mathrm{cond}(RC)\setminus(\mathrm{var}(\mathrm{effect}(RC))\cup\mathrm{var}(C))} \mathrm{cond}(RC)$

ii) $C^{\iota} \in \{G_1, \neg G_1\}$

iii) $\text{effect}(RC)^\iota = G_2$

From here Definition 12 yields the edge $E = (G_1 \to G_2)$ with label $\lambda(E) := (RC, C, \iota)$ exists in the meta ground graph $\text{Graph}^{\mathcal{E}}(\mathbf{P})$.

b) $\Rightarrow$ a) Assume we find an edge $E := G_1 \to G_2$ with label $\lambda(E) := (RC, C, \iota)$ in the meta ground graph $\text{Graph}^{\mathcal{E}}(\mathbf{P})$. In this case we obtain for $C \in \text{causes}(RC)$ the following assertions:

i) $(\mathcal{E}^{\mathbf{I}}, \iota) \models \exists_{\text{cond}(RC) \setminus (\text{var}(\text{effect}(RC)) \cup \text{var}(C))} \text{cond}(RC)$

ii) $R^\iota = G_1$ where $C \in \{R, \neg R\}$, i.e. $C^\iota \in \{G_1, \neg G_1\}$

iii) $\text{effect}(RC)^\iota = G_2$

Hence, we find the edge $G_1 \to G_2$ in the ground graph $\text{Graph}_{\mathcal{E}}(\mathbf{P})$.

□

Corollary 1 (Lifting a Subgraph to the Meta Ground Graph) *Let $\mathcal{E} \models \boldsymbol{C}$ be an external database. Further, let $G = (V, E)$ be a subgraph of the ground graph $\text{Graph}_{\mathcal{E}}(\boldsymbol{P})$. In this case there exists a subgraph $\tilde{G}$ on the set of nodes $V \subset \mathcal{G}(\mathcal{E})$ of the meta ground graph $\text{Graph}^{\mathcal{E}}(\boldsymbol{P})$ such that the identity on V induces an isomorphism of graphs $\tilde{G} \to G$.*

Proof. This is an immediate consequence of Proposition 4. □

Further, we ground the subgraphs of the generic dependence graph to the meta ground graph as follows:

Definition 13 *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a subgraph of the n-th generic dependence graph $n\text{-Graph}(\boldsymbol{P})$. Further, let ι be a variable interpretation of the variables occurring in a random atom of $\mathbb{V}$. We say that $\mathbb{G}$ **projects** with respect to ι if for every edge $E = R_1 \to R_2$ with label $\lambda(E) := (RC, C, \theta)$ its **grounding** $E^\iota := R_1^\iota \to R_2^\iota$ exists in the meta ground graph $\text{Graph}^{\mathcal{E}}(\boldsymbol{P})$ with label $\lambda(E^\iota) := (RC, C, \iota \circ \theta)$. In this case we define $\mathbb{G}^\iota := (\mathbb{V}^\iota, \mathbb{E}^\iota)$ to be the **projection** of $\mathbb{G}$ with respect to ι.*

*Assume $\mathbb{G}$ projects. Further, let $G = (V, E)$ be a subgraph of the meta ground graph $\text{Graph}^{\mathcal{E}}(\boldsymbol{P})$. If $\mathbb{V} \xrightarrow{\iota} V$ induces an isomorphism of directed acyclic graphs, we say that ι **grounds** $\mathbb{G}$ to G. In this case we also say that $\mathbb{G}$ **grounds** or that G **lifts** to $\mathbb{G}$.*

Remark 3 *It is easy to see that Definition 13 is consistent with Definition 8, i.e. the projection $\mathbb{G} \xrightarrow{\iota} \text{Graph}_{\mathcal{E}}(\boldsymbol{P})$ is nothing else as the composition of the map $\mathbb{G} \xrightarrow{\iota} \text{Graph}^{\mathcal{E}}(\boldsymbol{P})$ with the map of Corollary 1.*

To proceed we investigate the assumption on an external database $\mathcal{E}$ for a given subgraph to ground.

Lemma 1 (Project Subgraphs) *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a subgraph of the n-th generic dependence graph $n\text{-Graph}(\boldsymbol{P})$ and let ι be a variable interpretation of $\text{var}(\mathbb{G})$. In this* *case $\mathbb{G}$ projects with respect to ι if and only if the projection condition of $\mathbb{G}$ is satisfied, i.e.* $(\mathcal{E}^{\boldsymbol{I}}, \iota) \models \text{p\_cond}(\mathbb{G})$.

Proof. $\mathbb{G}$ projects with respect to ι. $\Leftrightarrow$

$\overset{DEF\ 13}{\Leftrightarrow}$ For every edge $E = (R_1 \to R_2) \in \mathbb{E}$ with label $\lambda(E) := (RC, C, \theta)$ its grounding $E^\iota := R_1^\iota \to R_2^\iota$ with label $\lambda(E^\iota) := (RC, C, \iota \circ \theta)$ exists in the meta ground graph $\text{Graph}^{\mathcal{E}}(\mathbf{P})$.

$\overset{DEF\ 12}{\Leftrightarrow}$ For every edge $E = R_1 \to R_2 \in \mathbb{E}$ with label $\lambda(E) := (RC, C, \theta)$ we find $(\mathcal{E}, \iota \circ \theta) \models \exists_{\text{var}(\text{cond}(RC))\setminus(\text{var}(C)\cup\text{var}(\text{effect}(RC)))} \text{cond}(RC)$.

$\overset{DEF\ 8}{\Leftrightarrow}$ For every edge $E = R_1 \to R_2 \in \mathbb{E}$ we find $(\mathcal{E}^{\mathbf{I}}, \iota) \models \text{p\_cond}(E)$.

$\overset{DEF\ 8}{\Leftrightarrow}$ $(\mathcal{E}^{\mathbf{I}}, \iota) \models \text{p\_cond}(\mathbb{G})$

□

Finally, we are able to give a proof of Theorem 3:

Proof of Theorem 3. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a finite subgraph of the n-th generic dependence graph n-Graph($\mathbf{P}$) and let ι be a variable interpretation in some external database $\mathcal{E}$ on the variables occurring in a random atom of $\mathbb{V}$ such that $(\mathcal{E}, \iota) \models \text{g\_cond}(\mathbb{G})$.

By Corollary 1 it is sufficient to show that $\mathbb{G}$ grounds to the meta ground graph $\text{Graph}^{\mathcal{E}}(\mathbf{P})$. Obviously, the map $\mathbb{G} \to \mathbb{G}^\iota$ is bijective on the nodes. Hence, it is left to show that for every edge $A^\iota \to B^\iota$ in $\mathbb{G}^\iota$ there exists an edge $A \to B$ in $\mathbb{G}$ which holds by construction of $\mathbb{G}^\iota$.

Finally, note that the other direction trivially follows as monomorphisms of directed graphs are always injective. □

Proof of Theorem 4. Let G be a subgraph of the meta ground graph $\text{Graph}^{\mathcal{E}}(\mathbf{P})$ with not more that n nodes. Note that by Corollary 1 it is sufficient to find a lift of G.

Now, for every node $r(x_1, ..., x_k)$ of the graph G we replace the constants x_i of $\mathfrak{C}^*$ by pairwise distinct variables $X_{(r,x_i,i)} \in \mathcal{V}_n^{\mathbf{s}(x_i)}$ for $1 \leq i \leq k$. Note that Definition 8 guarantees that there are enough variables in $\mathcal{V}_n^{\mathbf{s}(x_i)}$. Further, for every edge $E := (R^\iota \to S^\iota)$ in G with label $\lambda(E) := (RC, C, \iota)$ obtain an edge $\tilde{E} := (R^{\theta_\iota} \to S^{\theta_\iota})$ in the n-th generic dependence graph n-Graph($\mathbf{P}$) with label $\lambda(\tilde{E}) = (RC, C, \theta_\iota)$, where θ_ι is defined as follows: Let $r(X_1, ..., X_k) \in \{R, S\}$. In this case we define $\theta_\iota(X_i) := X_{(r,x_i,i)}$ if $\iota(X_i) = x_i$. Note that θ_ι yields a well-defined substitution by Definition 3. In this way we obtain a subgraph $\mathbb{G}$ of the n-th generic dependence graph n-Graph($\mathbf{P}$) with no two nodes sharing a common variable. Finally, we set $\tilde{\iota}(X_{r,x_i}) := x_i$ to obtain $\iota = \tilde{\iota} \circ \theta_\iota$, i.e. the interpretation $\tilde{\iota}$ grounds the subgraph $\mathbb{G}$ to the graph G as desired.

□

Proof of Theorem 5. Assume the program structures $\mathbf{P}_{1/2}$ are faithfully indistinguishable. Further, let $\mathcal{E}$ be an external database and assume the Datalog query $adjacent(X_1, X_2)$ has a solution $X_1 = r(x_1, ..., x_k), X_2 = s(y_1, ..., y_l)$ with respect to the program $\mathrm{ID}(\mathbf{P}_1)$ due to clause $adjacent(R_1, R_2) \leftarrow \mathrm{g\_cond}(\mathbb{G})$. In this case we obtain directly from Theorem 3 that there exists an edge $r(x_1, ..., x_k) - s(y_1, ..., y_l)$ in the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P})$. By Proposition 2 this edge also occurs in the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P}_2)$. Further, Theorem 4 yields a lift $r(X_1', ..., X_k') - s(Y_1', ..., Y_l')$ in the second generic dependence graph 2-Graph$(\mathbf{P}_2)$. By relabelling variables we obtain the lift $E := (R{-}S)$, i.e. $(\mathcal{E}, \iota) \models \mathrm{g\_cond}(R{-}S)$ with $\iota(X_i) = x_i$ and $\iota(Y_j) = y_j$. Hence, we obtain also the solution $X_1 = r(x_1, ..., x_k), X_2 = s(y_1, ..., y_l)$ for the Datalog query $adjacent(X_1, X_2)$ with respect to $\mathrm{ID}(\mathbf{P}_2)$.

Further, assume $immorality(X_1, X_2, X_3)$ has a solution $X_1 = r(x_1, ..., x_k)$, $X_2 = s(y_1, ..., y_l)$ and $X_3 = t(z_1, ..., z_m)$ with respect to the Datalog program $\mathrm{ID}(\mathbf{P}_1)$ due to clause $immorality(R_1, R_2, R_3) \leftarrow \mathrm{g\_cond}(\mathbb{G}) \cup \{\neg\, \mathrm{p\_cond}(E_1), ..., \neg\, \mathrm{p\_cond}(E_j)\}$. In this case Theorem 3 yields a collider $r(x_1, ..., x_k) \rightarrow s(y_1, ..., y_l) \leftarrow t(z_1, ..., z_m)$. If this collider were shielded, arguing as before we can lift this shielded collider to $\mathbb{G}$ meaning by Theorem 3 that one of the projection conditions $\mathrm{p\_cond}(E_i)$ holds, which is a contradiction. Hence by Proposition 2 this unshielded collider also exists in the ground graph $\mathrm{Graph}_{\mathcal{E}}(\mathbf{P})$ and with the same argument as before we obtain that $X_1 = r(x_1, ..., x_k)$, $X_2 = s(y_1, ..., y_l)$ and $X_3 = t(z_1, ..., z_m)$ is also a solution to the query $immorality(X_1, X_2, X_3)$ with respect to the Datalog program $\mathrm{ID}(\mathbf{P}_2)$. From here the desired result follows as the program structures $\mathbf{P}_{1/2}$ share the same external databases. □