

“Would life be more interesting if I were in AI?”

Answering Counterfactuals based on Probabilistic Inductive Logic Programming

Kilian Rückschloß

Ludwig-Maximilians-Universität München
Oettingenstraße 67, 80538 München, Germany

kilian.rueckschloss@lmu.de

Felix Weitkämper

Ludwig-Maximilians-Universität München
Oettingenstraße 67, 80538 München, Germany

felix.weitkaemper@lmu.de

Probabilistic logic programs are logic programs where some facts hold with a specified probability. Here, we investigate these programs with a causal framework that allows counterfactual queries. Learning the program structure from observational data is usually done through heuristic search relying on statistical tests. However, these statistical tests lack information about the causal mechanism generating the data, which makes it unfeasible to use the resulting programs for counterfactual reasoning. To address this, we propose a language fragment that allows reconstructing a program from its induced distribution. This further enables us to learn programs supporting counterfactual queries.

1 Introduction

While only observing the world, humans are used to drawing counterfactual conclusions, i.e. they reason about how events would have unfolded under different circumstances. This leads us to judgements like: “I would have published more papers, if I were in AI.” without actually experiencing the alternative reality in which we work in AI. Note that this capability allows us to make sense of the past, to plan courses of actions, to make emotional and social judgments as well as to adapt our behaviour [2]. Hence, in artificial intelligence one also wants to infer a model of the world that supports counterfactual reasoning.

Currently, the WHATIF-solver [3] establishes a counterfactual reasoning for ProbLog programs [1], i.e. logic programs in which each clause holds with a specified probability. However, is the counterfactual reasoning provided by a ProbLog program uniquely determined by its distribution semantics [7]?

Assume for instance that a patient is treated, denoted *treatment*, with a probability of 0.5. If we treat a patient, we expect him to recover, denoted *recovery*, with a probability of 0.7, otherwise he recovers with a probability of 0.5. The resulting distribution can be encoded with the following two programs $\mathbf{P}_{1/2}$.

$$\begin{array}{llll} \mathbf{P}_1 : & 0.5 :: \textit{treatment} & 0.5 :: \textit{recovery} & 0.4 :: \textit{recovery} \leftarrow \textit{treatment} \\ \mathbf{P}_2 : & 0.5 :: \textit{treatment} & 0.5 :: \textit{recovery} \leftarrow \neg \textit{treatment} & 0.7 :: \textit{recovery} \leftarrow \textit{treatment} \end{array}$$

Assume further that the patient recovers while he has not been treated. What is the probability that he would have recovered under treatment?

In both programs $\mathbf{P}_{1/2}$, we conclude from our observations that the patient recovers because of the second clause, i.e. we conclude that the second clause holds in the world we observe. If we had additionally treated the patient, under program \mathbf{P}_1 , he would still have recovered as the second clause in \mathbf{P}_1 is still applicable under treatment. Hence, we obtain a probability of one for the patient to recover under treatment. Whereas, in program \mathbf{P}_2 , the second clause is not applicable under treatment. Hence, in this

case, if we had treated the patient here, he can only recover because of the third clause and we obtain a probability of 0.7 for the patient to recover under treatment.

As we see, in general, the classical distribution semantics [7] does not uniquely determine the outcome of a counterfactual query. Further, note that ProbLog programs are usually learned from observations sampled from a distribution of interest. Hence, even if we assume perfect learning, we can only ensure to obtain a program representing the correct distribution. In particular, a structure learning algorithm is not able to distinguish the programs $\mathbf{P}_{1/2}$, i.e. we cannot ensure that a learned program answers counterfactual queries correctly.

In this contribution, we present a fragment in which each program is uniquely determined by its class dependency graph and the corresponding distribution. We further argue that this yields a setting for the available structure learning methods which supports counterfactual reasoning.

2 Foundations

Here, we introduce ProbLog programs [1] and we recall how counterfactual queries are processed on them [3]. Finally, we quickly explain the design of the currently available structure learning algorithms for these programs.

As the semantics of non-ground ProbLog programs is usually defined by grounding, we restrict ourselves to the propositional case. Hence, we construct our programs from a **propositional alphabet** that is given by a finite set of propositions \mathfrak{P} together with a subset $\mathfrak{E}(\mathfrak{P}) \subseteq \mathfrak{P}$ of **external proposition**. In this context, we call $\mathfrak{I}(\mathfrak{P}) := \mathfrak{P} \setminus \mathfrak{E}(\mathfrak{P})$ the set of **internal propositions**.

A **literal** l is an expression p or $\neg p$ for a proposition $p \in \mathfrak{P}$. We call l a **positive literal** if it is of the form p and a **negative literal** if it is of the form $\neg p$. Further, we call l an **external** or **internal literal** if $p \in \mathfrak{E}(\mathfrak{P})$ or $p \in \mathfrak{I}(\mathfrak{P})$ respectively. A **(logical) clause** LC is an expression $h \leftarrow b_1, \dots, b_n$ where $h \in \mathfrak{I}(\mathfrak{P})$ is an internal proposition called the **head** and where $\{b_1, \dots, b_n\}$ is a finite set of literals called the **body** of LC . Finally, a **random fact** RF is an expression $\pi(RF) :: u(RF)$ where $\pi(RF) \in [0, 1]$ is the **probability** and where $u(RF) \in \mathfrak{E}(\mathfrak{P})$ is an external proposition called the **proposition** of RF .

Example 1. Consider the alphabet $\mathfrak{P} := \{treatment, recovery, u_1, u_2, u_3\}$ with external literals $\mathfrak{E}(\mathfrak{P})$ given by $\{u_1, u_2, u_3\}$. We have that *treatment* is a positive literal, whereas $\neg treatment$ is a negative literal. Further, $recovery \leftarrow treatment, u_3$ is a clause and $0.4 :: u_3$ is a random fact.

Now a **logic program** is a finite set of clauses and a **ProbLog program** \mathbf{P} is given by a logic program $LP(\mathbf{P})$ and a set $Facts(\mathbf{P})$ consisting of a unique random fact for every external proposition. In this case, we call $LP(\mathbf{P})$ the **underlying logic program** of \mathbf{P} . Finally, we define $(\mathfrak{P}$ -)formulas ϕ and $(\mathfrak{P}$ -)structures $\mathcal{M} : \mathfrak{P} \rightarrow \{True, False\}$ as usual in propositional logic. Whether a given \mathfrak{P} -structure \mathcal{M} satisfies a formula ϕ , written $\mathcal{M} \models \phi$, is also defined as usual in propositional logic.

Example 2. In the alphabet \mathfrak{P} of Example 1 we can write the following ProbLog program.

$$0.5 :: u_1 \quad 0.5 :: u_2 \quad 0.4 :: u_3 \quad treatment \leftarrow u_1 \quad recovery \leftarrow u_2 \quad recovery \leftarrow treatment, u_3$$

As the semantics of ProbLog programs we choose the FCM-semantics [6], which supports counterfactual reasoning: For a ProbLog program \mathbf{P} we define the **functional causal models semantics**

or **FCM-semantics** to be the system of Boolean equations

$$\text{FCM}(\mathbf{P}) := \left\{ p^{\text{FCM}} := \bigvee_{\substack{LC \in \text{LP}(\mathbf{P}) \\ \text{head}(LC)=p}} \left(\bigwedge_{\substack{l \in \text{body}(LC) \\ l \text{ internal literal}}} l^{\text{FCM}} \wedge \bigwedge_{\substack{u(RF) \in \text{body}(LC) \\ RF \in \text{Facts}(\mathbf{P})}} u(RF)^{\text{FCM}} \right) \right\}_{p \in \mathcal{I}(\mathfrak{B})}.$$

Here, we find that the $u(RF)^{\text{FCM}}$ are mutually independent Boolean random variables for every random fact $RF \in \text{Facts}(\mathbf{P})$, each holding hold with probability $\pi(RF)$.

Example 3. *The FCM-semantics of the program \mathbf{P} in Example 2 is given by*

$$\text{treatment}^{\text{FCM}} := u_1^{\text{FCM}} \quad \text{recovery}^{\text{FCM}} := u_2^{\text{FCM}} \vee (\text{treatment}^{\text{FCM}} \wedge u_3^{\text{FCM}}),$$

where u_1^{FCM} , u_2^{FCM} and u_3^{FCM} are mutually independent Boolean random variables holding true with a probability of 0.5, 0.5 and 0.4 respectively.

For the rest of this section, we fix a ProbLog program \mathbf{P} with an acyclic underlying logic program. Note that the FCM-semantics $\text{FCM}(\mathbf{P})$ yields a unique solution for every internal proposition $p \in \mathcal{I}(\mathfrak{B})$ in terms of the mutually independent Boolean random variables $u(RF)^{\text{FCM}}$. In this way, it defines a distribution on the \mathfrak{B} -structures $\mathcal{M} : \mathfrak{B} \rightarrow \{\text{True}, \text{False}\}$ which coincides with the distribution semantics [7] according to Rükschloß and Weitekämper [6]. Finally, we define the probability of a formula ϕ to hold as $\pi(\phi) := \sum_{\substack{\mathcal{M} \text{ } \mathfrak{B}\text{-structure} \\ \mathcal{M} \models \phi}} \pi(\mathcal{M})$.

However, the FCM-semantics does not only support queries about conditional and unconditional probabilities. It allows us to answer two more general causal query types, namely determining the effect of external interventions and counterfactuals [6]. Assume for instance we want to **intervene** and set a subset $\mathbf{X} \subseteq \mathcal{I}(\mathfrak{B})$ of internal propositions to truth values specified by an assignment \mathbf{x} . In this case, we build a modified program $\mathbf{P}^{\text{do}(\mathbf{x})}$ by erasing all clauses $LC \in \text{LP}(\mathbf{P})$ with head in \mathbf{X} and by adding a fact $p \leftarrow \text{if } p \in \mathbf{X}$ is set to true by \mathbf{x} . If we now ask for the probability $\pi(\phi | \text{do}(\mathbf{x}))$ of a formula ϕ to hold after setting \mathbf{X} to the values \mathbf{x} , we query the program $\mathbf{P}^{\text{do}(\mathbf{x})}$ for the probability of ϕ .

Example 4. *Assume we treat the patient in the program \mathbf{P} of Example 2. In this case, we obtain*

$$0.5 :: u_1 \quad 0.5 :: u_2 \quad 0.4 :: u_3 \quad \text{treatment} \leftarrow \quad \text{recovery} \leftarrow u_2 \quad \text{recovery} \leftarrow \text{treatment}, u_3$$

for the modified program $\mathbf{P}^{\text{do}(\text{treatment})}$. This means we obtain a probability of 0.7 for recovery if we are the doctor and decide to treat our patient.

Further, we do not only want to either observe or intervene, we also want to know what the probability of an event would have been if we had intervened before observing some evidence. This is especially interesting in the **counterfactual** case where our evidence contradicts the given intervention.

Example 5. *Consider the query in the introduction and observe that the evidence $\{\neg \text{treatment}, \text{recovery}\}$ contradicts the intervention $\text{do}(\text{treatment})$, i.e. this is a counterfactual query.*

Hence, fix another subset of internal propositions $\mathbf{E} \subseteq \mathcal{I}(\mathfrak{B})$ and assume we observe the evidence that the propositions in \mathbf{E} take values according to the assignment \mathbf{e} . We now ask for the probability $\pi(\phi | \mathbf{e}, \text{do}(\mathbf{x}))$ of the formula ϕ to hold if we had set the propositions in \mathbf{X} to the values specified by \mathbf{x} before observing our evidence \mathbf{e} . To answer queries like that we proceed as Kiesel et al. [3]:

First we generate two copies $\mathfrak{I}(\mathfrak{P})^{e/i}$ of the set of internal propositions – one to handle the evidence and the other to handle the interventions. Further, we set $u^{e/i} := u$ for every external proposition $u \in \mathfrak{E}(\mathfrak{P})$. Note that this yields maps $_{e/i}$ of literals, clauses, programs etc. We define the **counterfactual semantics** of \mathbf{P} to be the ProbLog program \mathbf{P}^K which consists of the logic program $\mathbf{L}(\mathbf{P})^e \cup \mathbf{L}(\mathbf{P})^i$ and the random facts $\text{Facts}(\mathbf{P})$. Now we intervene in \mathbf{P}^K and set the proposition in \mathbf{X}^i to the truth values specified by \mathbf{x} to obtain the program $\mathbf{P}^{K, \text{do}(\mathbf{x})}$. Finally, we query the program $\mathbf{P}^{K, \text{do}(\mathbf{x})}$ for the probability $\pi(\phi^i | \mathbf{E}^e = \mathbf{e})$ to obtain the desired result for $\pi(\phi | \mathbf{e}, \text{do}(\mathbf{x}))$.

Example 6. Assume we did not treat the patient in the program \mathbf{P} of Example 2 and he recovered. What is the probability $\pi(\text{recovery} | \neg \text{treatment}, \text{recovery}, \text{do}(\text{treatment}))$ that he would have recovered, if he had been treated? To answer this question we query the program $\mathbf{P}^{K, \text{do}(\text{treatment})}$

$$\begin{array}{lll} 0.5 :: u_1 & 0.5 :: u_2 & 0.4 :: u_3 \\ \text{treatment}^i \leftarrow & \text{recovery}^i \leftarrow u_2 & \text{recovery}^i \leftarrow \text{treatment}^i, u_3 \\ \text{treatment}^e \leftarrow u_1 & \text{recovery}^e \leftarrow u_2 & \text{recovery}^e \leftarrow \text{treatment}^e, u_3 \end{array}$$

for the probability $\pi(\text{recovery}^i | \text{recovery}^e, \neg \text{treatment}^e) = 1$.

This procedure automates Pearl’s counterfactual reasoning [4] and is implemented in the WHATIF-solver of Kiesel et al. [3].

In this contribution, we restrict ourselves to ProbLog programs, which can be represented with ProbLog clauses. A **ProbLog clause** RC is an expression $\pi(RC) :: \text{effect}(RC) \leftarrow \text{causes}(RC)$, where $\text{effect}(RC) \in \mathfrak{I}(\mathfrak{P})$ is an internal proposition called the **effect**, where $\text{causes}(RC)$ is a finite set of internal literals called the **causes** and where $0 \leq \pi(RC) \leq 1$ is a number called the **probability** of RC . The ProbLog clause RC is an abbreviation for the following pair of a random fact and a logical clause.

$$RF(RC) := (\pi(RC) :: u(RC)) \quad LC(RC) := (h \leftarrow b_1, \dots, b_n, u(RC)),$$

where $u(RC) \in \mathfrak{E}(\mathfrak{P})$ is a distinct external literal. From now on, by abuse of language, a **ProbLog program** \mathbf{P} is a finite set of ProbLog clauses, i.e. a ProbLog program consisting of the logic program $\mathbf{L}(\mathbf{P}) := \{LC(RC) : RC \in \mathbf{P}\}$ and of the random facts $\text{Facts}(\mathbf{P}) := \{RF(RC) : RC \in \mathbf{P}\}$.

Example 7. Observe that the program \mathbf{P}_1 from the introduction is an abbreviation for the ProbLog program in Example 2.

The **class dependency graph** $\text{Graph}(\mathbf{P})$ of a ProbLog program \mathbf{P} is the directed graph on the internal propositions $\mathfrak{I}(\mathfrak{P})$ obtained by drawing an edge $p_1 \rightarrow p_2$ if and only if there exists a ProbLog clause $RC \in \mathbf{P}$ with a cause p_1 or $\neg p_1$ and with effect p_2 . We say that the program \mathbf{P} is **acyclic** if its class dependency graph $\text{Graph}(\mathbf{P})$ is a directed acyclic graph. Moreover, the program \mathbf{P} is **positive** if the causes of every ProbLog clause $RC \in \mathbf{P}$ form a set of positive literals.

Example 8. The class dependency graph of the programs $\mathbf{P}_{1/2}$ in the introduction is given by the edge $\text{treatment} \rightarrow \text{recovery}$. Further, program \mathbf{P}_1 is positive whereas \mathbf{P}_2 is not.

Next, we quickly recall the overview over the structure learning techniques for propositional ProbLog programs from Riguzzi [5, §10]. Given suitable data, all those algorithms search the space of programs defined by a language bias and background knowledge with heuristics relying on statistical tests.

Here, the **language bias** defining the clause and therefore the program space is defined by mode declarations of the form $\text{modeb}(*, q)$ and $\text{modeh}(*, p)$ as well as declarations of the form $\text{determination}(p, q)$

for propositions p and q . A positive ProbLog clause $\pi :: \text{effect}(RC) \leftarrow \text{causes}(RC)$ lies in the language defined by our bias if we declared $\text{modeh}(*, \text{effect}(RC))$, $\text{modeb}(*, c)$ for all c in $\text{causes}(RC)$ and $\text{determination}(\text{effect}(RC), c)$ for all c in $\text{causes}(RC)$.

Moreover, we can express **background knowledge** in a logic program defining further propositions in terms of the given data. This is also the way how one can learn clauses with negation by adding a clause $\text{neg}_p \leftarrow \neg p$ to the background knowledge for every proposition p . We call the pair of a language bias and a background knowledge the **setting** for a structure learning algorithm.

Example 9. *To learn the program P_1 of the introduction we need to specify the setting \mathfrak{S}_1 which consists of the bias $\text{modeh}(*, \text{recovery})$, $\text{modeb}(*, \text{treatment})$, $\text{determination}(\text{recovery}, \text{treatment})$ and an empty background knowledge. If we want to consider the program P_2 as well, we additionally need the declarations $\text{modeb}(*, \text{neg_treatment})$ and $\text{determination}(\text{recovery}, \text{neg_treatment})$ together with the clause $\text{neg_treatment} \leftarrow \neg \text{treatment}$ in the background knowledge resulting in the setting \mathfrak{S}_2 .*

Note that via the $\text{determination}/2$ predicate the language bias essentially provides the class dependency graph, i.e. the corresponding cause-effect relationships, of our program as prior knowledge to the structure learning algorithm.

3 Results

Generally, in structure learning, from some prior knowledge encoded by a setting one wants to derive a program that describes a given set of data. In most of the cases, the data consists of observations. We additionally assume that our data consists of samples drawn from the distribution induced by a hidden ProbLog program $\tilde{\mathbf{P}}$ of interest and the prior knowledge consists of a language bias, i.e. of the class dependency graph of $\tilde{\mathbf{P}}$. Further, to decide how good a candidate program \mathbf{P} represents our dataset we process statistical tests. However, statistical tests only measure how well the induced distribution of the program \mathbf{P} fits a given set of observations. They generally reveal no information about the causal mechanism generating our data. Hence, we cannot measure whether the causal mechanism represented by a candidate ProbLog program \mathbf{P} coincides with the causal mechanism underlying our data, i.e. with the causal mechanism described by $\tilde{\mathbf{P}}$.

Example 10. *Consider the programs $P_{1/2}$ of the introduction. While they both represent different causal models yielding to different counterfactual estimations, they yield the same distribution semantics and share the same class dependency graph.*

Hence, if we take $\tilde{\mathbf{P}} := P_{1/2}$ for the hidden program, we sample from the same distribution in both cases. That means that even with the correct language bias a structure learning algorithm cannot determine which of the two programs actually generated the provided data unless it is given further knowledge.

More drastically, Example 10 illustrates that without further prior knowledge, even under the assumption of perfect learning, it is only possible to learn a program \mathbf{P} which is ensured to represent the correct distribution. In particular, we expect that a learned program \mathbf{P} does not necessarily answer counterfactual queries correctly.

In the following, we study the fragment of acyclic proper positive ProbLog programs in normal form. A ProbLog program \mathbf{P} is **proper in normal form** if every clause $RC \in \mathbf{P}$ has a probability $0 < \pi(RC) < 1$, if any two distinct clauses $RC_{1/2} \in \mathbf{P}$ have distinct causes $\text{causes}(RC_1) \neq \text{causes}(RC_2)$ or distinct effects $\text{effect}(RC_1) \neq \text{effect}(RC_2)$ and if every sink s in the class dependency graph gives rise to a random fact $\alpha :: s$. The main result of this contribution now states that all programs lying in this fragment are uniquely determined by their class dependency graph and their underlying distribution.

Theorem 1. *Every acyclic proper positive ProbLog program in normal form \mathbf{P} can be reconstructed from its class dependency graph $\text{Graph}(\mathbf{P})$ and the induced distribution π .*

Proof. We proceed by induction on the number n of nodes in the class dependency graph $\text{Graph}(\mathbf{P})$.

$n = 1$: In this case, the program \mathbf{P} consists only of one clause $\pi :: p \leftarrow$. Hence, we set $\pi := \pi(p)$ and we are done.

$n > 1$: Choose a sink $h \in \mathfrak{P}$ of $\text{Graph}(\mathbf{P})$. Further, denote by $\mathbf{P} \setminus h$ the program that results from \mathbf{P} if we erase all clauses with effect h . By maximality h does not occur in the causes of any other clause, i.e. $\mathbf{P} \setminus h$ induces the same distribution on $\mathfrak{I}(\mathfrak{P}) \setminus \{h\}$ as the program \mathbf{P} and it has the graph $\text{Graph}(\mathbf{P}) \setminus h$ as its class dependency graph. Here, $\text{Graph}(\mathbf{P}) \setminus h$ denotes the graph that results from $\text{Graph}(\mathbf{P})$ if we erase the node h together with all edges pointing into it. Now, by the induction hypothesis we can reconstruct the program $\mathbf{P} \setminus h$ from the given data.

Hence, we are left to reconstruct the clauses defining h itself. Note that the parents $b \in \text{pa}(h)$ of h in $\text{Graph}(\mathbf{P})$ are the only propositions that may occur in the body of a clause defining h . Further, note that each of these occurrences is positive. We consider the function

$$\text{Ind}_h^{\mathbf{P}} : \mathcal{P}(\text{pa}(h)) \rightarrow [0, 1] \quad T \mapsto \pi(h | \{t, \neg s : t \in T, s \in \text{pa}(h) \setminus T\}),$$

where $\mathcal{P}(\cdot)$ denotes the power set operator. Recall from Pearl [4, §3] that observing the parents of h is the same as intervening on them. Hence, we see that

$$\text{Ind}_h^{\mathbf{P}}(T) := \pi \left(\bigvee_{\substack{RC \in \mathbf{P} \\ \text{causes}(RC) \subseteq T \\ \text{effect}(RC) = h}} u(RC) \right) = \sum_{\substack{RC_1, \dots, RC_k \in \mathbf{P} \\ k \in \mathbb{N}, \text{causes}(RC) \subseteq T \\ \text{effect}(RC) = h}} (-1)^k \prod_{i=1}^k \pi(RC_i) \quad (1)$$

Now, it is easy to see that $T \subseteq \text{pa}(\mathbf{P})$ are the causes of a clause in \mathbf{P} if and only if $\text{Ind}_h^{\mathbf{P}}(S) < \text{Ind}_h^{\mathbf{P}}(T)$ for all $S \subseteq \text{pa}(h)$ with $S \subsetneq T$. Further, we obtain the parameters of \mathbf{P} by recursion: We find that $\pi(RC) = \text{Ind}_h^{\mathbf{P}}(\text{body}(RC))$ for every clause $RC \in \mathbf{P}$ with a minimal body. Further, in the recursion step Equation (1) yields a one-dimensional linear equation for the parameter of interest. \square

In a forthcoming paper, we prove that the answers to all counterfactual queries uniquely determine a proper ProbLog program in normal form. Thus if we want to answer counterfactual queries based on a learned program \mathbf{P} , we almost need to fully reconstruct the hidden program $\tilde{\mathbf{P}}$.

Remark 1. *If we estimate the functions $\text{Ind}_h^{\mathbf{P}}$ using relative frequencies, we obtain a structure learning algorithm that recovers an acyclic proper positive ProbLog program in normal form from a known causal structure and a sufficiently large set of samples. The resulting distributions (one for every counterfactual query) are guaranteed to converge in probability. Further, the complexity is exponential in the maximal number of parents of a node in the class dependency graph and linear in the size of the alphabet \mathfrak{P} .*

Finally, assume we apply a structure learning algorithm [5, §10] with a language bias encoding the class dependency graph $\text{Graph}(\tilde{\mathbf{P}})$ to obtain a program $\tilde{\mathbf{P}}$. Let us further assume that we learned perfectly, i.e. that the program \mathbf{P} encodes the same distribution as $\tilde{\mathbf{P}}$. If we now assume that both programs \mathbf{P} and $\tilde{\mathbf{P}}$ are acyclic proper positive ProbLog programs in normal form, Theorem 1 yields that \mathbf{P} and $\tilde{\mathbf{P}}$ coincide, i.e. \mathbf{P} expresses the full causal content of $\tilde{\mathbf{P}}$.

Since without background knowledge each currently available structure learning algorithm [5, §10] only searches for positive programs fitting a given dataset, the assumption to learn proper positive ProbLog programs in normal form is realized easily. Causally, the absence of background knowledge implies that we assume our data to be generated by a proper positive ProbLog program in normal form.

Corollary 1. *Assume we are given data sampled from a hidden proper positive ProbLog program in normal form \tilde{P} and assume we are aware of the class dependency graph $\text{Graph}(\tilde{P})$ of \tilde{P} . Every structure learning algorithm that is able to learn a proper positive ProbLog program in normal form with the correct class dependency graph and the correct distribution reconstructs \tilde{P} from the provided data. In particular, the result of such a structure learning algorithm supports counterfactual reasoning. \square*

4 Conclusion

In the introduction, we show that the distribution semantics does not uniquely determine counterfactual query outcomes for ProbLog programs, making it unfeasible to use the currently available structure learning algorithms for counterfactual reasoning. However, our main result reveals that proper positive ProbLog programs in normal form are actually uniquely determined by their distribution semantics and their class dependency graph. Hence, if applied without background knowledge and if we assume perfect learning, the currently available structure learning algorithms can recover these programs when provided with the correct language bias i.e. they support counterfactual reasoning in this setting.

In a forthcoming paper, we show that counterfactual reasoning uniquely determines a proper program in normal form, i.e. it is not sufficient to learn programs under a coarser notion of equivalence. Determining the equivalence classes of ProbLog programs representing the same distributions and predicting the behaviour of the available structure learning algorithm for more general fragments of ProbLog are promising directions for future work extending this contribution.

References

- [1] Luc De Raedt, Angelika Kimmig & Hannu Toivonen (2007): *ProbLog: A Probabilistic Prolog and Its Application in Link Discovery*. In: *20th International Joint Conference on Artificial Intelligence*, 7, AAAI Press, Hyderabad, India, pp. 2462–2467, doi:10.5555/1625275.1625673.
- [2] Nicole Van Hoeck (2015): *Cognitive Neuroscience of Human Counterfactual Reasoning*. *Frontiers in Human Neuroscience* 9, doi:10.3389/fnhum.2015.00420.
- [3] Rafael Kiesel, Kilian Rückschloß & Felix Weitekämper (2023): “What if?” in *Probabilistic Logic Programming*. Accepted for *TPLP Proceedings of ICLP 2023*. Available at <https://arxiv.org/abs/2305.15318>.
- [4] Judea Pearl (2000): *Causality*, 2 edition. Cambridge University Press, Cambridge, UK, doi:10.1017/CB09780511803161.
- [5] Fabrizio Riguzzi (2020): *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*. River Publishers, doi:10.1201/9781003338192.
- [6] Kilian Rückschloß & Felix Weitekämper (2022): *Exploiting the Full Power of Pearl’s Causality in Probabilistic Logic Programming*. In: *Proceedings of the International Conference on Logic Programming 2022 Workshops, CEUR Workshop Proceedings 3193*, CEUR-WS.org, Haifa, Israel. Available at <http://ceur-ws.org/Vol-3193/paper1PLP.pdf>.
- [7] Taisuke Sato (1995): *A Statistical Learning Method for Logic Programs with Distribution Semantics*. In: *Logic Programming: The 12th International Conference*, The MIT Press, Tokyo, Japan, pp. 715–729, doi:10.7551/mitpress/4298.003.0069.