*TPLP* **23** (4): 884–899, 2023. 

doi:10.1017/S1471068423000133 First published online 17 July 2023

# *"What if?" in Probabilistic Logic Programming*∗†

RAFAEL KIESEL
*TU Wien, Vienna, Austria*
(*e-mail:* rafael.kiesel@tuwien.ac.at)

KILIAN RÜCKSCHLOß and FELIX WEITKÄMPER
*Ludwig-Maximilians-Universität München, Munich, Germany*
(*e-mails:* kilian.rueckschloss@lmu.de, felix.weitkaemper@lmu.de)

*submitted 25 May 2023; accepted 12 June 2023*

**Abstract**

A ProbLog program is a logic program with facts that only hold with a specified probability. In this contribution, we extend this ProbLog language by the ability to answer "What if" queries. Intuitively, a ProbLog program defines a distribution by solving a system of equations in terms of mutually independent predefined Boolean random variables. In the theory of causality, Judea Pearl proposes a counterfactual reasoning for such systems of equations. Based on Pearl's calculus, we provide a procedure for processing these counterfactual queries on ProbLog programs, together with a proof of correctness and a full implementation. Using the latter, we provide insights into the influence of different parameters on the scalability of inference. Finally, we also show that our approach is consistent with CP-logic, that is with the causal semantics for logic programs with annotated with disjunctions.

*KEYWORDS*: counterfactual reasoning, probabilistic logic programming, ProbLog, LPAD, causality, FCM-semantics, CP-logic

## 1 Introduction

Humans show the remarkable skill to reason in terms of counterfactuals. This means we reason about how events would unfold under different circumstances without actually experiencing all these different realities. For instance, we make judgments like: "If I had taken a bus earlier, I would have arrived on time." without actually experiencing the alternative reality in which we took the bus earlier. As this capability lies at the basis of making sense of the past, planning courses of actions, making emotional and social judgments as well as adapting our behavior, one also wants an artificial intelligence to reason counterfactually (Hoeck 2015).

Here, we focus on the counterfactual reasoning with the semantics provided by Pearl (2000). Our aim is to establish this kind of reasoning in the ProbLog language of De Raedt

∗ All authors contributed equally to this work.
† 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. Additionally, it was supported by FWF project W1255-N23.

*et al.* (2007). To illustrate this issue, we introduce a version of the sprinkler example from Pearl (2000), Section 1.4.

It is spring or summer, written *szn_spr_sum*, with a probability of $\pi_1 := 0.5$. Consider a road, which passes along a field with a sprinkler on it. In spring or summer, the sprinkler is on, written *sprinkler*, with probability $\pi_2 := 0.7$. Moreover, it rains, denoted by *rain*, with probability $\pi_3 := 0.1$ in spring or summer and with probability $\pi_4 := 0.6$ in fall or winter. If it rains or the sprinkler is on, the pavement of the road gets wet, denoted by *wet*. When the pavement is wet, the road is slippery, denoted by *slippery*. Under the usual reading of ProbLog programs, one would model the situation above with the following program **P**:

```
0.5::u1. 0.7::u2. 0.1::u3. 0.6::u4.
szn_spr_sum :- u1. sprinkler :- szn_spr_sum, u2.
rain :- szn_spr_sum, u3. rain :- \+szn_spr_sum, u4.
wet :- rain. wet :- sprinkler. slippery :- wet.
```

To construct a semantics for the program **P,** we generate mutually independent Boolean random variables $u1$-$u4$ with $\pi(ui) = \pi_i$ for all $1 \leq i \leq 4$. The meaning of the program **P** is then given by the following system of equations:

$$\begin{aligned} &szn\_spr\_sum := u1 \qquad rain := (szn\_spr\_sum \wedge u3) \vee (\neg szn\_spr\_sum \wedge u4) \\ &sprinkler := szn\_spr\_sum \wedge u2 \quad wet := (rain \vee sprinkler) \quad slippery := wet \end{aligned} \tag{1}$$

Finally, assume we observe that the sprinkler is on and that the road is slippery. What is the probability of the road being slippery if the sprinkler were switched off?

Since we observe that the sprinkler is on, we conclude that it is spring or summer. However, if the sprinkler is off, the only possibility for the road to be slippery is given by *rain*. Hence, we obtain a probability of 0.1 for the road to be slippery if the sprinkler were off.

In this work, we automate this kind of reasoning. However, to the best of our knowledge, current probabilistic logic programming systems cannot evaluate counterfactual queries.While we may ask what the probability of *slippery* is if we switch the sprinkler off and observe some evidence, we obtain a zero probability for *sprinkler* after switching the sprinkler off, which renders the corresponding conditional probability meaningless. To circumvent this problem, we adapt the twin network method of Balke and Pearl (1994) from causal models to probabilistic logic programming, with a proof of correctness. Notably, this reduces counterfactual reasoning to marginal inference over a modified program. Hence, we can immediately make use of the established efficient inference engines to accomplish our goal.

We also check that our approach is consistent with the counterfactual reasoning for logic programs with annotated disjunctions or LPAD-programs (Vennekens *et al.* 2004), which was presented by Vennekens *et al.* (2010). In this way, we fill the gap of showing that the causal reasoning for LPAD-programs of Vennekens *et al.* (2009) is indeed consistent with Pearl's theory of causality and we establish the expressive equivalence of ProbLog and LPAD regarding counterfactual reasoning.

Apart from our theoretical contributions, we provide a full implementation by making use of the aspmc library (Eiter *et al.* 2021). Additionally, we investigate the scalability of the two main approaches used for efficient inference, with respect to program size and structural complexity, as well as the influence of evidence and interventions on performance.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

## 2 Preliminaries

Here, we recall the theory of counterfactual reasoning from Pearl (2000) before we introduce the ProbLog language of De Raedt *et al.* (2007) in which we would like to process counterfactual queries.

### *2.1 Pearl's formal theory of counterfactual reasoning*

The starting point of a formal theory of counterfactual reasoning is the introduction of a model that is capable of answering the intended queries. To this aim, we recall the definition of a functional causal model from Pearl (2000), Sections 1.4.1 and 7 respectively:

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

Fortunately, causal models do not only support queries about conditional and unconditional probabilities but also queries about the effect of external interventions. Assume we are given a subset of variables $\mathbf{X} := \{X_1, ..., X_k\} \subseteq \mathbf{V}$ together with a vector of possible values $\mathbf{x} := (x_1, ..., x_k)$ for the variables in $\mathbf{X}$. In order to model the effect of setting the variables in $\mathbf{X}$ to the values specified by $\mathbf{x}$, we simply replace the equations for $X_i$ in $\mathcal{M}$ by $X_i := x_i$ for all $1 \leq i \leq k$.

To guarantee that the causal models $\mathcal{M}$ and $\mathcal{M}^{\text{do}(\mathbf{X}:=\mathbf{x})}$ yield well-defined distributions $\pi_{\mathcal{M}}(\_)$ and $\pi_{\mathcal{M}}(\_|\text{do}(\mathbf{X} := \mathbf{x}))$, we explicitly assert that the systems of equations $\mathcal{M}$ and $\mathcal{M}^{\text{do}(\mathbf{X}:=\mathbf{x})}$ have a unique solution for every tuple $\mathbf{e}$ of possible values for the error terms $\text{error}(X)$, $X \in \mathbf{V}$ and for every intervention $\mathbf{X} := \mathbf{x}$.

*Example 1*
The system of equations (1) from Section 1 forms a (functional) causal model on the set of variables $\mathbf{V} := \{szn\_spr\_sum, rain, sprinkler, wet, slippery\}$ if we define $\text{error}(szn\_spr\_sum) := u1$, $\text{error}(sprinkler) := u2$ and $\text{error}(rain) := (u3, u4)$. To predict the effect of switching the sprinkler on, we simply replace the equation for *sprinkler* by $sprinkler := True$.

Finally, let $\mathbf{E}, \mathbf{X} \subseteq \mathbf{V}$ be two subset of our set of variables $\mathbf{V}$. Now suppose we observe the evidence that $\mathbf{E} = \mathbf{e}$ and ask ourselves what would have been happened if we had set $\mathbf{X} := \mathbf{x}$. Note that in general $\mathbf{X} = \mathbf{x}$ and $\mathbf{E} = \mathbf{e}$ contradict each other. In this case, we talk about a counterfactual query.

*Example 2*
Reconsider the query $\pi(slippery|slippery, sprinkler, \text{do}(\neg sprinkler))$ in the introduction, that is in the causal model (1) we observe the sprinkler to be on and the road to be slippery while asking for the probability of the road to be slippery if the sprinkler were off. This is a counterfactual query as our evidence $\{sprinkler, slippery\}$ contradicts our intervention $\text{do}(\neg sprinkler)$.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

To answer this query based on a causal model $\mathcal{M}$ on $\mathbf{V}$, we proceed in three steps: In the abduction step, we adjust the distribution of our error terms by replacing the distribution $\pi_{\mathcal{M}}(\text{error}(V))$ with the conditional distribution $\pi_{\mathcal{M}}(\text{error}(V)|\mathbf{E}=\mathbf{e})$ for all variables $V \in \mathbf{V}$. Next, in the action step we intervene in the resulting model according to $\mathbf{X} := \mathbf{x}$. Finally, we are able to compute the desired probabilities $\pi_{\mathcal{M}}(\_|\mathbf{E}=\mathbf{e}, \text{do}(\mathbf{X} := \mathbf{x}))$ from the modified model in the prediction step (Pearl 2000, Section 1.4.4). For an illustration of the treatment of counterfactuals, we refer to the introduction.

To avoid storing the joint distribution $\pi_{\mathcal{M}}(\text{error}(V)|\mathbf{E}=\mathbf{e})$ for $V \in \mathbf{V}$, Balke and Pearl (1994) developed the twin network method. They first copy the set of variables $\mathbf{V}$ to a set $\mathbf{V}^*$. Further, they build a new causal model $\mathcal{M}^K$ on the variables $\mathbf{V} \cup \mathbf{V}^*$ by setting

$$V := \begin{cases} f_X(\text{pa}(X), \text{error}(X)), & \text{if } V = X \in \mathbf{V} \\ f_X(\text{pa}(X)^*, \text{error}(X)), & \text{if } V = X^* \in \mathbf{V}^* \end{cases}.$$

for every $V \in \mathbf{V} \cup \mathbf{V}^*$, where $\text{pa}(X)^* := \{X^* | X \in \text{pa}(X)\}$. Further, they intervene according to $\mathbf{X}^* := \mathbf{x}$ to obtain the model $\mathcal{M}^{K,\text{do}(\mathbf{X}^*:=\mathbf{x})}$. Finally, one expects that

$$\pi_{\mathcal{M}}(\_|\mathbf{E}=\mathbf{e}, \text{do}(\mathbf{X} := \mathbf{x})) = \pi_{\mathcal{M}^{K,\text{do}(\mathbf{X}^*:=\mathbf{x})}}(\_^*|\mathbf{E}=\mathbf{e}).$$

In Example 8, we demonstrate the twin network method for the ProbLog program $\mathbf{P}$ and the counterfactual query of the introduction.

### *2.2 The ProbLog language*

We proceed by recalling the ProbLog language from De Raedt *et al.* (2007). As the semantics of non-ground ProbLog programs is usually defined by grounding, we will restrict ourselves to the propositional case, that is we construct our programs from a propositional alphabet $\mathfrak{P}$:

*Definition 2* (*propositional alphabet*)
A propositional alphabet $\mathfrak{P}$ is a finite set of propositions together with a subset $\mathfrak{E}(\mathfrak{P}) \subseteq \mathfrak{P}$ of external propositions. Further, we call $\mathfrak{I}(\mathfrak{P}) = \mathfrak{P} \backslash \mathfrak{E}(\mathfrak{P})$ the set of internal propositions.

*Example 3*
To build the ProbLog program $\mathbf{P}$ in Section 1, we need the alphabet $\mathfrak{P}$ consisting of the internal propositions $\mathfrak{I}(\mathfrak{P}) := \{szn\_spr\_sum,\ sprinkler,\ rain,\ wet,\ slippery\}$ and the external propositions $\mathfrak{E}(\mathfrak{P}) := \{u1,\ u2,\ u3,\ u4\}$.

From propositional alphabets, we build literals, clauses, and random facts, where random facts are used to specify the probabilities in our model. To proceed, let us fix a propositional alphabet $\mathfrak{P}$.

*Definition 3* (*Literal, Clause, and Random Fact*)
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 $\neg p$. A clause $LC$ is an expression of the form $h \leftarrow b_1, ..., b_n$, where $\text{head}(LC) := h \in \mathfrak{I}(\mathfrak{P})$ is an internal proposition and where $\text{body}(LC) := \{b_1, ..., b_n\}$ is a finite set of literals. A random fact $RF$ is an expression of the form $\pi(RF) :: u(RF)$, where $u(RF) \in \mathfrak{E}(\mathfrak{P})$ is an external proposition and where $\pi(RF) \in [0, 1]$ is the probability of $u(RF)$.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

*Example 4*
In Example 3, we have that $szn\_spr\_sum$ is a positive literal, whereas $\neg szn\_spr\_sum$ is a negative literal. Further, $rain \leftarrow \neg szn\_spr\_sum, u4$ is a clause and $0.6 :: u4$ is a random fact.

Next, we give the definition of logic programs and ProbLog programs:

*Definition 4* (*Logic Program and ProbLog Program*)
A logic program is a finite set of clauses. Further, a ProbLog program $\mathbf{P}$ is given by a logic program $\mathrm{LP}(\mathbf{P})$ and a set $\mathrm{Facts}(\mathbf{P})$, which consists of a unique random fact for every external proposition. We call $\mathrm{LP}(\mathbf{P})$ the underlying logic program of $\mathbf{P}$.

To reflect the closed world assumption, we omit random facts of the form $0 :: u$ in the set $\mathrm{Facts}(\mathbf{P})$.

*Example 5*
The program $\mathbf{P}$ from the introduction is a ProbLog program. We obtain the corresponding underlying logic program $\mathrm{LP}(\mathbf{P})$ by erasing all random facts of the form $\_ :: ui$ from $\mathbf{P}$.

For a set of propositions, $\mathfrak{Q} \subseteq \mathfrak{P}$ a $\mathfrak{Q}$-structure is a function $\mathcal{M} : \mathfrak{Q} \to \{True, False\},\ p \mapsto p^{\mathcal{M}}$. Whether a formula $\phi$ is satisfied by a $\mathfrak{Q}$-structure $\mathcal{M}$, written $\mathcal{M} \models \phi$, is defined as usual in propositional logic. As the semantics of a logic program $\mathbf{P}$ with stratified negation, we take the assignment $\mathcal{E} \mapsto \mathcal{M}(\mathcal{E}, \mathbf{P})$ that relates each $\mathfrak{E}$-structure $\mathcal{E}$ with the minimal model $\mathcal{M}(\mathcal{E}, \mathbf{P})$ of the program $\mathbf{P} \cup \mathcal{E}$.

## 3 Counterfactual reasoning: Intervening and observing simultaneously

We return to the objective of this paper, establishing Pearl's treatment of counterfactual queries in ProbLog. As a first step, we introduce a new semantics for ProbLog programs in terms of causal models.

*Definition 5* (*FCM-semantics*)
For a ProbLog program **P,** the functional causal model semantics or FCM-semantics is the system of equations that is given by

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

where $u(RF)^{\mathrm{FCM}}$ are mutually independent Boolean random variables for every random fact $RF \in \mathrm{Facts}(\mathbf{P})$ that are distributed according to $\pi\left[u(RF)^{\mathrm{FCM}}\right] = \pi(RF)$. Here, an empty disjunction evaluates to $False$ and an empty conjunction evaluates to $True$.

Further, we say that $\mathbf{P}$ has unique supported models if $\mathrm{FCM}(\mathbf{P})$ is a causal model, that is if it possesses a unique solution for every $\mathfrak{E}$-structure $\mathcal{E}$ and every possible intervention $\mathbf{X} := \mathbf{x}$. In this case, the superscript FCM indicates that the expressions are interpreted according to the FCM-semantics as random variables rather than predicate symbols. It will be omitted if the context is clear. For a Problog program $\mathbf{P}$ with unique supported models, the causal model $\mathrm{FCM}(\mathbf{P})$ determines a unique joint distribution $\pi_{\mathbf{P}}^{\mathrm{FCM}}$ on $\mathfrak{P}$.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

Finally, for a $\mathfrak{P}$-formula $\phi$ we define the probability to be true by

$$\pi_{\mathbf{P}}^{\mathrm{FCM}}(\phi) := \sum_{\substack{\mathcal{M}\ \mathfrak{P}\text{-structure} \\ \mathcal{M}\models\phi}} \pi_{\mathbf{P}}^{\mathrm{FCM}}(\mathcal{M}) = \sum_{\substack{\mathcal{E}\ \mathfrak{E}\text{-structure} \\ \mathcal{M}(\mathcal{E},\mathbf{P})\models\phi}} \pi_{\mathbf{P}}^{\mathrm{FCM}}(\mathcal{E}).$$

*Example 6*

As intended in the introduction, the causal model (1) yields the FCM-semantics of the program $\mathbf{P}$. Now let us calculate the probability $\pi_{\mathbf{P}}^{\mathrm{FCM}}(sprinkler)$ that the sprinkler is on.

$$\pi_{\mathbf{P}}^{\mathrm{FCM}}(sprinkler) = \sum_{\substack{\mathcal{M}\ \mathfrak{P}\text{-structure} \\ \mathcal{M}\models sprinkler}} \pi_{\mathbf{P}}^{\mathrm{FCM}}(\mathcal{M}) = \sum_{\substack{\mathcal{E}\ \mathfrak{E}\text{-structure} \\ \mathcal{M}(\mathcal{E},\mathbf{P})\models sprinker}} \pi_{\mathbf{P}}^{\mathrm{FCM}}(\mathcal{E}) =$$

$$= \pi(u1,u2,u3,u4) + \pi(u1,u2,\neg u3,u4) + \pi(u1,u2,u3,\neg u4)$$

$$+ \pi(u1,u2,\neg u3,\neg u4) \overset{\substack{ui \text{ mutually} \\ \text{independent}}}{=}$$

$$= 0.5\cdot 0.7\cdot 0.1\cdot 0.6 + 0.5\cdot 0.7\cdot 0.9\cdot 0.6 + 0.5\cdot 0.7\cdot 0.1\cdot 0.4 + 0.5\cdot 0.7\cdot 0.9\cdot 0.4 = 0.35$$

As desired, we obtain that the FCM-semantics consistently generalizes the distribution semantics of Poole (1993) and Sato (1995).

*Theorem 1* (*Rückschloß and Weitkämper 2022*)
Let $\mathbf{P}$ be a ProbLog program with unique supported models. The FCM-semantics defines a joint distribution $\pi_{\mathbf{P}}^{\mathrm{FCM}}$ on $\mathfrak{P}$, which coincides with the distribution semantics $\pi_{\mathbf{P}}^{dist}$.□

As intended, our new semantics transfers the query types of functional causal models to the framework of ProbLog. Let $\mathbf{P}$ be a ProbLog program with unique supported models. First, we discuss the treatment of external interventions.

Let $\phi$ be a $\mathfrak{P}$-formula and let $\mathbf{X} \subseteq \mathfrak{I}(\mathfrak{P})$ be a subset of internal propositions together with a truth value assignment $\mathbf{x}$. Assume we would like to calculate the probability $\pi_{\mathbf{P}}^{FCM}(\phi|\mathrm{do}(\mathbf{X} := \mathbf{x}))$ of $\phi$ being true after setting the random variables in $\mathbf{X}^{\mathrm{FCM}}$ to the truth values specified by $\mathbf{x}$. In this case, the Definition 1 and Definition 5 yield the following algorithm:

*Procedure 1* (*Treatment of External Interventions*)
We build a modified program $\mathbf{P}^{\mathrm{do}(\mathbf{X}:=\mathbf{x})}$ by erasing for every proposition $h \in \mathbf{X}$ each clause $LC \in \mathrm{LP}(\mathbf{P})$ with $\mathrm{head}(LC) = h$ and adding the fact $h \leftarrow$ to $\mathrm{LP}(\mathbf{P})$ if $h^{\mathbf{x}} = True$.

Finally, we query the program $\mathbf{P}^{\mathrm{do}(\mathbf{X}:=\mathbf{x})}$ for the probability of $\phi$ to obtain the desired probability $\pi_{\mathbf{P}}^{\mathrm{FCM}}(\phi|\mathrm{do}(\mathbf{X} := \mathbf{x}))$.

From the construction of the program $\mathbf{P}^{\mathrm{do}(\mathbf{X}:=\mathbf{x})}$ in Procedure 1, we derive the following classification of programs with unique supported models.

*Proposition 2* (*Characterization of Programs with Unique Supported Models*)
A ProbLog program $\mathbf{P}$ has unique supported models if and only if for every $\mathfrak{E}$-structure $\mathcal{E}$ and for every truth value assignment $\mathbf{x}$ on a subset of internal propositions $\mathbf{X} \subseteq \mathfrak{I}(\mathfrak{P})$ there exists a unique model $\mathcal{M}\left(\mathcal{E}, \mathrm{LP}\left(\mathbf{P}^{\mathrm{do}(\mathbf{X}:=\mathbf{x})}\right)\right)$ of the logic program

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

LP $\left(\mathbf{P}^{\mathrm{do}(\mathbf{X}:=\mathbf{x})}\right) \cup \mathcal{E}$. In particular, the program $\mathbf{P}$ has unique supported model if its underlying logic program LP($\mathbf{P}$) is acyclic. □

*Example 7*
As the underlying logic program of the ProbLog program $\mathbf{P}$ in the introduction is acyclic, we obtain from Proposition 2 that it is a ProbLog program with unique supported models that is its FCM-semantics is well-defined.

However, we do not only want to either observe or intervene. We also want to observe and intervene simultaneously.

Let $\mathbf{E} \subseteq \mathfrak{I}(\mathfrak{P})$ be another subset of internal propositions together with a truth value assignment $\mathbf{e}$. Now suppose we observe the evidence $\mathbf{E}^{\mathrm{FCM}} = \mathbf{e}$ and we ask ourselves what is the probability $\pi_{\mathbf{P}}^{\mathrm{FCM}}(\phi|\mathbf{E} = \mathbf{e}, \mathrm{do}(\mathbf{X} := \mathbf{x}))$ of the formula $\phi$ to hold if we had set $\mathbf{X}^{\mathrm{FCM}} := \mathbf{x}$. Note that again we explicitly allow $\mathbf{e}$ and $\mathbf{x}$ to contradict each other. The twin network method of Balke and Pearl (1994) yields the following procedure to answer those queries in ProbLog:

*Procedure 2* (*Treatment of Counterfactuals*)
First, we define two propositional alphabets $\mathfrak{P}^e$ to handle the evidence and $\mathfrak{P}^i$ to handle the interventions. In particular, we set $\mathfrak{E}(\mathfrak{P}^e) = \mathfrak{E}(\mathfrak{P}^i) = \mathfrak{E}(\mathfrak{P})$ and $\mathfrak{I}(\mathfrak{P}^{e/i}) := \{p^{e/i} : p \in \mathfrak{I}(\mathfrak{P})\}$ with $\mathfrak{I}(\mathfrak{P}^e) \cap \mathfrak{I}(\mathfrak{P}^i) = \emptyset$. In this way, we obtain maps $\_^{e/i} : \mathfrak{P} \to \mathfrak{P}^{e/i},\ p \mapsto \begin{cases} p^{e/i}, & p \in \mathfrak{I}(\mathfrak{P}) \\ p, & \text{else} \end{cases}$ that easily generalize to literals, clauses, programs, etc.

Further, we define the counterfactual semantics of $\mathbf{P}$ by $\mathbf{P}^K := \mathbf{P}^e \cup \mathbf{P}^i$. Next, we intervene in $\mathbf{P}^K$ according to $\mathrm{do}(\mathbf{X}^i := \mathbf{x})$ and obtain the program $\mathbf{P}^{K,\mathrm{do}(\mathbf{X}^i:=\mathbf{x})}$ of Procedure 1. Finally, we obtain the desired probability $\pi_{\mathbf{P}}^{FCM}(\phi|\mathbf{E} = \mathbf{e}, \mathrm{do}(\mathbf{X} := \mathbf{x}))$ by querying the program $\mathbf{P}^{K,\mathrm{do}(\mathbf{X}^i:=\mathbf{x})}$ for the conditional probability $\pi(\phi^i|\mathbf{E}^e = \mathbf{e})$.

*Example 8*
Consider the program $\mathbf{P}$ of Example 5 and assume we observe that the sprinkler is on and that it is slippery. To calculate the probability $\pi(\mathit{slippery}|\mathit{sprinkler}, \mathit{slippery}, \mathrm{do}(\neg \mathit{sprinkler}))$ that it is slippery if the sprinkler was off, we need to process the query $\pi(\mathit{slippery}^i|\mathit{slippery}^e, \mathit{sprinkler}^e)$ on the following program $\mathbf{P}^{K,\mathrm{do}(\neg \mathit{sprinkler}^i)}$.

```
0.5::u1. 0.7::u2. 0.1::u3. 0.6::u4.
szn_spr_sum__e :- u1. sprinkler__e :- szn_spr_sum__e, u2.
rain__e :- szn_spr_sum__e, u3. rain__e :- \+szn_spr_sum__e, u4.
wet__e :- rain__e. wet__e :- sprinkler__e. slippery__e :- wet__e.
szn_spr_sum__i :- u1.
rain__i :- szn_spr_sum__i, u3. rain__i :- \+szn_spr_sum__i, u4.
wet__i :- rain__i. wet__i :- sprinkler__i. slippery__i :- wet__i.
```

Note that we use the string__ to refer to the superscript $e/i$.

In the Appendix, we prove the following result, stating that a ProbLog program $\mathbf{P}$ yields the same answers to counterfactual queries, denoted $\pi_{\mathbf{P}}^{\mathrm{FCM}}(\_|\_)$, as the causal model FCM($\mathbf{P}$), denoted $\pi_{\mathrm{FCM}(\mathbf{P})}(\_|\_)$.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

*Theorem 3* (*Correctness of our Treatment of Counterfactuals*)
Our treatment of counterfactual queries in Procedure 2 is correct that is in the situation of Procedure 2 we obtain that $\pi_{\mathbf{P}}^{FCM}(\phi|\mathbf{E} = \mathbf{e}, \text{do}(\mathbf{X} := \mathbf{x})) = \pi_{\text{FCM}(\mathbf{P})}(\phi|\mathbf{E} = \mathbf{e}, \text{do}(\mathbf{X} := \mathbf{x}))$.

## 4 Relation to CP-logic

Vennekens *et al.* (2009) establish CP-logic as a causal semantics for the LPAD-programs of Vennekens *et al.* (2004). Further, recall Riguzzi (2020), Section 2.4 to see that each LPAD-program $\mathbf{P}$ can be translated to a ProbLog program Prob($\mathbf{P}$) such that the distribution semantics is preserved. Analogously, we can read each ProbLog program $\mathbf{P}$ as an LPAD-Program LPAD($\mathbf{P}$) with the same distribution semantics as $\mathbf{P}$.

As CP-logic yields a causal semantics, it allows us to answer queries about the effect of external interventions. More generally, Vennekens *et al.* (2010) even introduce a counterfactual reasoning on the basis of CP-logic. However, to our knowledge this treatment of counterfactuals is neither implemented nor shown to be consistent with the formal theory of causality in Pearl (2000).

Further, it is a priori unclear whether the expressive equivalence of LPAD and ProbLog programs persists for counterfactual queries. In the Appendix, we compare the treatment of counterfactuals under CP-logic and under the FCM-semantics. This yields the following results.

*Theorem 4* (*Consistency with CP-Logic – Part 1*)
Let $\mathbf{P}$ be a propositional LPAD-program such that every selection yields a logic program with unique supported models. Further, let $\mathbf{X}$ and $\mathbf{E}$ be subsets of propositions with truth value assignments, given by the vectors $\mathbf{x}$ and $\mathbf{e}$, respectively. Finally, we fix a formula $\phi$ and denote by $\pi^{CP/FCM}_{\text{Prob}(\mathbf{P})/\mathbf{P}}(\phi|\mathbf{E} = e, \text{do}(\mathbf{X} := \mathbf{x}))$ the probability that $\phi$ is true, given that we observe $\mathbf{E} = \mathbf{e}$ while we had set $\mathbf{X} := \mathbf{x}$ under CP-logic and the FCM-semantics respectively. In this case, we obtain $\pi^{CP}_{\mathbf{P}}(\phi|\mathbf{E} = e, \text{do}(\mathbf{X} := \mathbf{x})) = \pi^{FCM}_{\text{Prob}(\mathbf{P})}(\phi|\mathbf{E} = e, \text{do}(\mathbf{X} := \mathbf{x}))$.

*Theorem 5* (*Consistency with CP-Logic – Part 2*)
If we reconsider the situation of Theorem 4 and assume that $\mathbf{P}$ is a ProbLog program with unique supported models, we obtain $\pi^{CP}_{\text{LPAD}(\mathbf{P})}(\phi|\mathbf{E} = e, \text{do}(\mathbf{X} := \mathbf{x})) = \pi^{FCM}_{\mathbf{P}}(\phi|\mathbf{E} = e, \text{do}(\mathbf{X} := \mathbf{x}))$.

*Remark 1*
We can also apply Procedure 2 to programs with stratified negation. In this case, the proofs of Theorems 4 and 5 do not need to be modified in order to yield the same statement. However, recalling Definition 1, we see that there is no theory of counterfactual reasoning for those programs. Hence, to us it is not clear how to interpret the results of Procedure 2 for programs that do not possess unique supported models.

In Theorems 4 and 5, we show that under the translations Prob(_) and LPAD(_) CP-logic for LPAD-programs is equivalent to our FCM-semantics, which itself by Theorem 3 is consistent with the formal theory of Pearl's causality. In this way, we fill the gap by showing that the causal reasoning provided for CP-logic is actually correct. Further, Theorems 4 and 5 show that the translations Prob(_) and LPAD(_) of Riguzzi (2020),

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

Section 2.4 do not only respect the distribution semantics but also are equivalent for more general causal queries.

## 5 Practical evaluation

We have seen that we can solve counterfactual queries by performing marginal inference over a rewritten probabilistic logic program with evidence. Most of the existing solvers for marginal inference, including ProbLog (Fierens *et al.* 2015), aspmc (Eiter *et al.* 2021), and PITA (Riguzzi and Swift 2011), can handle probabilistic queries with evidence in one way or another. Therefore, our theoretical results also immediately enable the use of these tools for efficient evaluation in practice.

*Knowledge Compilation for Evaluation* The currently most successful strategies for marginal inference make use of Knowledge Compilation (KC). They compile the logical theory underlying a probabilistic logic program into a so-called tractable circuit representation, such as binary decision diagrams (BDD), sentential decision diagrams (SDD) (Darwiche 2011) or smooth deterministic decomposable negation normal forms (sd-DNNF). While the resulting circuits may be much larger (up to exponentially in the worst case) than the original program, they come with the benefit that marginal inference for the original program is possible in polynomial time in their size (Darwiche and Marquis 2002).

When using KC, we can perform compilation either bottom-up or top-down. In bottom-up KC, we compile SDDs representing the truth of internal atoms in terms of only the truth of the external atoms. After combining the SDDs for the queries with the SDDs for the evidence, we can perform marginal inference on the results (Fierens *et al.* 2015).

For top-down KC, we introduce auxiliary variables for internal atoms, translate the program into a CNF, and compile an sd-DNNF for the whole theory. Again, we can perform marginal inference on the result (Eiter *et al.* 2021).

*Implementation* As the basis of our implementation, we make use of the solver library aspmc. It supports parsing, conversion to CNF and top-down KC including a KC-version of SHARPSAT[1] based on the work of Korhonen and Järvisalo (2021). Additionally, we added (i) the program transformation that introduces the duplicate atoms for the evidence part and the query part, and (ii) allowed for counterfactual queries based on it.

Furthermore, to obtain empirical results for bottom-up KC, we use PySDD,[2] which is a python wrapper around the SDD library of Choi and Darwiche (2013). This is also the library that ProbLog uses for bottom-up KC to SDDs.

## 6 Empirical evaluation

Here, we consider the scaling of evaluating counterfactual queries by using our translation to marginal inference. This can depend on (i) the number of atoms and rules in the

[1] github.com/raki123/sharpsat-td.
[2] github.com/wannesm/PySDD.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

program, (ii) the complexity of the program structure, and (iii) the number and type of interventions and evidence.

We investigate the influence of these parameters on both the bottom-up and top-down KC. Although top-down KC as in aspmc can be faster (Eiter *et al.* 2021) on usual marginal queries, results for bottom-up KC are relevant nevertheless since it is heavily used in ProbLog and PITA.

Furthermore, it is a priori not clear that the performance of these approaches on usual instances of marginal inference translates to the marginal queries obtained by our translation. Namely, they exhibit a lot of symmetries as we essentially duplicate the program as a first step of the translation. Thus, the scaling of both approaches and a comparison thereof is of interest.

### *6.1 Questions and hypotheses*

The first question we consider addresses the scalability of the bottom-up and top-down approaches in terms of the size of the program and the complexity of the program structure.

**Q1. Size and Structure**: What size and complexity of counterfactual query instances can be solved with bottom-up or top-down compilation?

Here, we expect similar scaling as for marginal inference, since evaluating one query is equivalent to performing marginal inference once. While we duplicate the atoms that occur in the instance, thus increasing the hardness, we can also make use of the evidence, which can decrease the hardness, since we can discard models that do not satisfy the evidence.

Since top-down compilation outperformed bottom-up compilation on marginal inference instances in related work (Eiter *et al.* 2021), we expect that the top-down approach scales better than the bottom-up approach.

Second, we are interested in the influence that the number of intervention and evidence atoms has, in addition to whether it is a positive or negative intervention/evidence atom.

**Q2. Evidence and Interventions**: How does the number and type of evidence and intervention atoms influence the performance?

We expect that evidence and interventions can lead to simplifications for the program. However, it is not clear whether this is the case in general, whether it only depends on the number of evidence/intervention atoms, and whether there is a difference between negative and positive evidence/intervention atoms.

### *6.2 Setup*

We describe how we aim to answer the questions posed in the previous subsection.

*Benchmark Instances* As instances, we consider acyclic-directed graphs $G$ with distinguished start and goal nodes $s$ and $g$. Here, we use the following probabilistic logic program to model the probability of reaching a vertex in $G$:

```
r(s). 0.1::trap(Y) :- p(X,Y). r(Y) :- p(X,Y).
1/d(X)::p(X, s_1(X)); ... ; 1/d(X)::p(X,s_d(X)):- r(X), \+ trap(X).
```

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

Here, `d(X)` refers to the number of outgoing arcs of $X$ in $G$, and `s_1(X)`, ..., `s_d(X)` refer to its direct descendants. We obtain the final program by replacing the variables $X, Y$ with constants corresponding to the vertices of $G$.

This program models that we reach (denoted by `r(.)`) the starting vertex $s$ and, at each vertex $v$ that we reach, decide uniformly at random which outgoing arc we include in our path (denoted by `p(.,.)`). If we include the arc $(v, w)$, then we reach the vertex $w$. However, we only include an outgoing arc, if we do not get trapped (denoted by `trap(.)`) at $v$.

This allows us to pose counterfactual queries regarding the probability of reaching the goal vertex $g$ by computing

$$\pi_{\mathbf{P}}^{FCM}(r(g)|(\neg)r(v_1), ..., (\neg)r(v_n), \mathrm{do}((\neg)r(v'_1)), ..., \mathrm{do}((\neg)r(v'_m)))$$

for some positive or negative evidence of reaching $v_1, \ldots, v_n$ and some positive or negative interventions on reaching $v'_1, \ldots, v'_m$.

In order to obtain instances of varying sizes and difficulties, we generated acyclic digraphs with a controlled size and treewidth. Broadly speaking, treewidth has been identified as an important parameter related to the hardness of marginal inference (Eiter *et al.* 2021; Korhonen and Järvisalo 2021) since it bounds the structural hardness of programs, by giving a limit on the dependencies between atoms.

Using two parameters $n, k \in \mathbb{N}$, we generated programs of size linear in $n$ and $k$ and treewidth $\min(k, n)$ as follows. We first generated a random tree of size $n$ using network. As a tree, it has treewidth 1. To obtain treewidth $\min(k, n)$, we added $k$ vertices with incoming arcs from each of the $n$ original vertices in the tree.[3] Finally, we added one vertex as the goal vertex, with incoming arcs from each of the $k$ vertices. At the start, we use the root of the tree.

*Benchmark Platform* All our solvers ran on a cluster consisting of 12 nodes. Each node of the cluster is equipped with two Intel Xeon E5-2650 CPUs, where each of these 12 physical cores runs at 2.2 GHz clock speed and has access to 256 GB shared RAM. Results are gathered on Ubuntu 16.04.1 LTS powered on Kernel 4.4.0-139 with hyperthreading disabled using version 3.7.6 of Python3.

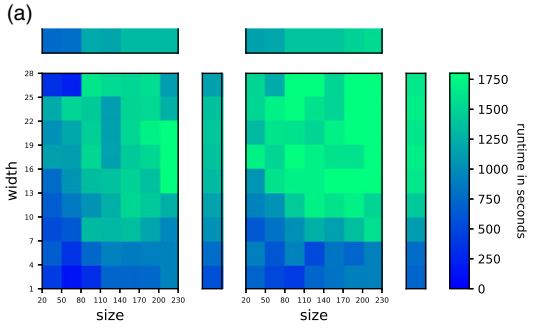
*Compared Configurations* We compare the two different configurations of our solver WhatIf (version 1.0.0, published at github.com/raki123/counterfactuals), namely bottom-up compilation with PySDD and top-down compilation with sharpSAT. Only the compilation and the following evaluation step differ between the two configurations, the rest stays unchanged.

*Comparisons* For both questions, we ran both configurations of our solver using a memory limit of 8GB and a time limit of 1800 s. If either limit was reached, we assigned the instance a time of 1800 s.

*Q1. Size and Structure* For the comparison of scalability with respect to size and structure, we generated one instance for each combination of $n = 20, 30, \ldots, 230$ and

[3] Observe that every vertex in the graph has at least degree $\min(n, k)$, which is known to imply treewidth $\geq \min(n, k)$.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press




Two plots showing the average runtime using bottom-up (right) and top-down (left) compilation. The x-axis denotes the size $n$ and the y-axis denoted the width $k$. For each square in the main plots the color of the square denotes the average runtime of all instances in the covered range. The extra plot on the top (resp. right side) denote the average for the size range (resp. width range) over all widths (resp. sizes).

The performance of the top-down (denoted as sharpsat) and bottom-up compilation (denoted as pysdd) on counterfactual queries regarding graph traversal. The x-axis denotes the number of solved instances and the y-axis denotes the runtime in seconds.

Fig. 1. Results for Q1.

$k = 1, 2, \ldots, 25$. We then randomly chose an evidence literal from the internal literals $(\neg)\ r(v)$. If possible, we further chose another such evidence literal consistent with the previous evidence. For the interventions, we chose two internal literals $(\neg)\ r(v)$ uniformly at random.

*Q2. Evidence and Interventions* For Q2, we chose a medium size ($n = 100$) and medium structural hardness ($k = 15$) and generated different combinations of evidence and interventions randomly on the same instance. Here, for each $e, i \in \{-5, \ldots, 0, \ldots, 5\}$ we consistently chose $|e|$ evidence atoms that were positive, if $e > 0$, and negative, otherwise. Analogously, we chose $|i|$ positive/negative intervention atoms.

### *6.3 Results and discussion*

We discuss the results (also available at github.com/raki123/counterfactuals/tree/final_results) of the two experimental evaluations.

*Q1. Size & Structure* The scalability results for size and structure are shown in Figure 1.

In Figure 1b, we see the overall comparison of bottom-up and top-down compilation. Here, we see that top-down compilation using SHARPSAT solves significantly more instances than bottom-up compilation with PYSDD. This aligns with similar results for usual marginal inference (Eiter *et al.* 2021). Thus, it seems like top-down compilation scales better overall.

In Figure 1a, we see that the average runtime depends on both the size and the width for either KC approach. This is especially visible in the subplots on top (resp. right) of the main plot containing the average runtime depending on the size (resp. width). While there is still a lot of variation in the main plots between patches of similar widths and sizes, the increase in the average runtime with respect to both width and size is rather smooth.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

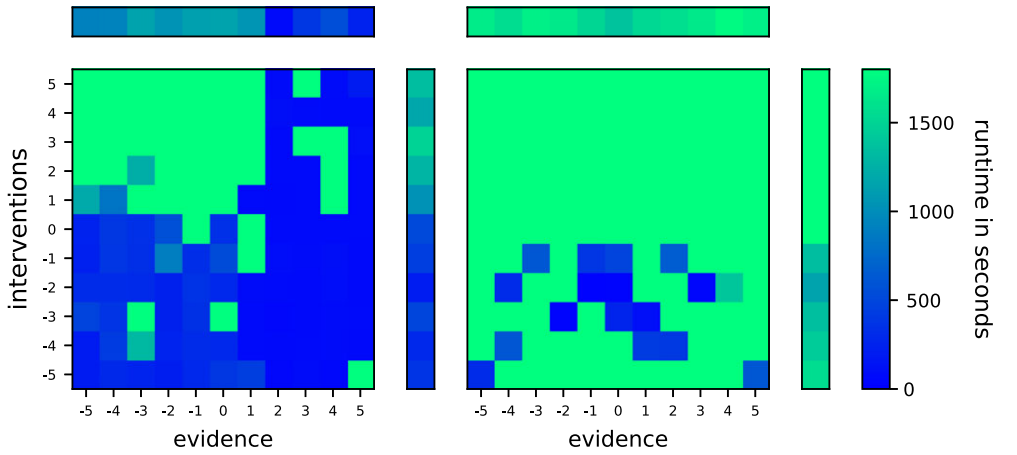



Fig. 2. Two plots showing the runtime using bottom-up (right) and top-down (left) compilation with varying evidence and intervention. The $x$-axis denotes the signed number of interventions, that is, $-n$ corresponds to $n$ negative interventions and $n$ corresponds to $n$ positive interventions. The y-axis denotes the signed number of evidence atoms using analogous logic. For each square in the main plots, the color of the square denotes the runtime of the instance with those parameters. The extra plot on the top (resp. right side) denotes the average for the number and type of evidences (resp. interventions) over all interventions (resp. evidences).

As expected, given the number of instances solved overall, top-down KC scales better to larger instances than bottom-up KC with respect to both size and structure. Interestingly however, for bottom-up KC the width seems to be of higher importance than for top-down KC. This can be observed especially in the average plots on top and to the right of the main plot again, where the change with respect to width is much more rapid for bottom-up KC than for top-down KC. For bottom-up KC, the average runtime goes from ∼500 s to ∼1800 s within the range of widths between 1 and 16, whereas for top-down KC it stays below ∼1500 s until width 28. For the change with respect to size on the other hand, both bottom-up and top-down KC change rather slowly, although the runtime for bottom-up KC is generally higher.

*Q2. Number & Type of Evidence/Intervention* The results for the effect of the number and types of evidence and intervention atoms are shown in Figure 2.

Here, for both bottom-up and top-down KC, we see that most instances are either solvable rather easily (i.e. within 500 s) or not solvable within the time limit of 1800 s. Furthermore, in both cases negative interventions, that is, interventions that make an atom false, have a tendency to decrease the runtime, whereas positive interventions, that is, interventions that make an atom true, can even increase the runtime compared to a complete lack of interventions.

However, in contrast to the results for Q1, we observe significantly different behavior for bottom-up and top-down KC. While positive evidence can vastly decrease the runtime for top-down compilation such that queries can be evaluated within 200 s, even in the presence of positive interventions, there is no observable difference between negative and positive evidence for bottom-up KC. Additionally, top-down KC seems to have a much easier time exploiting evidence and interventions to decrease the runtime.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

We suspect that the differences stem from the fact that top-down KC can make use of the restricted search space caused by evidence and negative interventions much better than bottom-up compilation. Especially for evidence, this makes sense: additional evidence atoms in bottom-up compilation lead to more SDDs that need to be compiled; however, they are only effectively used to restrict the search space when they are conjoined with the SDD for the query in the last step. On the other hand, top-down KC can simplify the given propositional theory *before* compilation, which can lead to a much smaller theory to start with and thus a much lower runtime.

The question why only negative interventions seem to lead to a decreased runtime for either strategy and why the effect of positive evidence is much stronger than that of negative evidence for top-down KC is harder to explain.

On the specific benchmark instances that we consider, negative interventions only remove rules, since all rule bodies mention $r(x)$ positively. On the other hand, positive interventions only remove the rules that entail them, but make the rules that depend on them easier to apply.

As for the stronger effect of positive evidence, it may be that there are fewer situations in which we derive an atom than there are situations in which we do not derive it. This would in turn mean that the restriction that an atom was true is stronger and can lead to more simplification. This seems reasonable on our benchmark instances, since there are many more paths through the generated networks that avoid a given vertex, than there are paths that use it.

Overall, this suggests that evidence is beneficial for the performance of top-down KC. Presumably, the performance benefit is less tied to the number and type of evidence atoms itself and more tied to the strength of the restriction caused by the evidence. For bottom-up KC, evidence seems to have more of a negative effect, if any.

While in our investigation interventions caused a positive or negative effect depending on whether they were negative or positive respectively, it is likely that in general their effect depends less on whether they are positive or negative. Instead, we assume that interventions that decrease the number of rules that can be applied are beneficial for performance, whereas those that make additional rules applicable (by removing an atom from the body) can degrade the performance.

## 7 Conclusion

The main result in this contribution is the treatment of counterfactual queries for ProbLog programs with unique supported models given by Procedure 2 together with the proof of its correctness in Theorem 3. We also provide an implementation of Procedure 2 that allows us to investigate the scalability of counterfactual reasoning in Section 6. This investigation reveals that typical approaches for marginal inference can scale to programs of moderate sizes, especially if they are not too complicated structurally. Additionally, we see that evidence typically makes inference easier but only for top-down KC, whereas interventions can make inference easier for both approaches but interestingly also lead to harder problems. Finally, Theorems 4 and 5 show that our approach to counterfactual reasoning is consistent with CP-logic for LPAD-programs. Note that this consistency result is valid for arbitrary programs with stratified negation. However, there is no theory

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press

for counterfactual reasoning in these programs. In our opinion, interpreting the results of Procedure 2 for more general programs yields an interesting direction for future work.

## Supplementary material

To view supplementary material for this article, please visit http://doi.org/10.1017/S1471068423000133.

## References


Balke, A. and Pearl, J. 1994. Probabilistic evaluation of counterfactual queries. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence (AAAI 1994)*. AAAI Press, 230–237.

Choi, A. and Darwiche, A. 2013. Dynamic minimization of sentential decision diagrams. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*. AAAI Press.

Darwiche, A. 2011. SDD: A new canonical representation of propositional knowledge bases. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. IJCAI/AAAI, 819–826.

Darwiche, A. and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research 17*, 229–264.

De Raedt, L., Kimmig, A. and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, vol. 7. AAAI Press, 2462–2467.

Eiter, T., Hecher, M. and Kiesel, R. 2021. Treewidth-aware cycle breaking for algebraic answer set counting. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021)*. IJCAI Organization, 269–279.

Fierens, D., den Broeck, G. V., Renkens, J., Shterionov, D. S., Gutmann, B., Thon, I., Janssens, G. and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming 15*, 3, 358–401.

Hoeck, N. V. 2015. Cognitive neuroscience of human counterfactual reasoning. *Frontiers in Human Neuroscience 9*. doi: 10.3389/fnhum.2015.00420

Korhonen, T. and Järvisalo, M. 2021. Integrating tree decompositions into decision heuristics of propositional model counters (short paper). In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. LIPIcs, vol. 210. Schloss Dagstuhl, 8:1–8:11.

Pearl, J. 2000. *Causality*, 2nd ed. Cambridge University Press, Cambridge, UK.

Poole, D. 1993. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64*, 81–129.

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

Riguzzi, F. and Swift, T. 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming 11*, 4–5, 433–449.

Rückschloss, K. and Weitkämper, F. 2022. Exploiting the full power of Pearl's causality in probabilistic logic programming. In *Proceedings of the International Conference on Logic Programming 2022 Workshops (ICLP 2022)*. CEUR Workshop Proceedings, vol. 3193. CEUR-WS.org.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming*. MIT Press, 715–729.

Vennekens, J., Bruynooghe, M. and Denecker, M. 2010. Embracing events in causal modelling: Interventions and counterfactuals in CP-logic. In *Logics in Artificial Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 313–325.

Vennekens, J., Denecker, M. and Bruynooghe, M. 2009. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming 9*, 3, 245–308.

Vennekens, J., Verbaeten, S. and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *Logic Programming*. Springer, 431–445.

https://doi.org/10.1017/S1471068423000133 Published online by Cambridge University Press