Efficient Counterfactual Reasoning in ProbLog via Single World Intervention Programs

SAIMUN HABIB*
(*e-mail:* S.Habib-1@ed.ac.uk)

VAISHAK BELLE*
(*e-mail:* vbelle@ed.ac.uk)

FENGXIANG HE*
(*e-mail:* F.He@ed.ac.uk)
**University of Edinburgh, Edinburgh, United Kingdom*

submitted xx xx xxxx; revised xx xx xxxx; accepted xx xx xxxx

Abstract

Probabilistic Logic Programming (PLP) languages, like ProbLog, naturally support reasoning under uncertainty, while maintaining a declarative and interpretable framework. Meanwhile, counterfactual reasoning (i.e., answering "what if" questions) is critical for ensuring AI systems are robust and trustworthy; however, integrating this capability into PLP can be computationally prohibitive and unstable in accuracy. This paper addresses this challenge, by proposing an efficient program transformation for counterfactuals as Single World Intervention Programs (SWIPs) in ProbLog. By systematically splitting ProbLog clauses to observed and fixed components relevant to a counterfactual, we create a transformed program that (1) does not asymptotically exceed the computational complexity of existing methods, and is strictly smaller in common cases, and (2) reduces counterfactual reasoning to marginal inference over a simpler program. We formally prove the correctness of our approach, which relies on a weaker set independence assumptions and is consistent with conditional independencies, showing the resulting marginal probabilities match the counterfactual distributions of the underlying Structural Causal Model in wide domains. Our method achieves a 35% reduction in inference time versus existing methods in extensive experiments. This work makes complex counterfactual reasoning more computationally tractable and reliable, providing a crucial step towards developing more robust and explainable AI systems. The code is at `https://github.com/EVIEHub/swip`.

KEYWORDS: counterfactual reasoning, probabilistic logic programming, ProbLog, causality, single world intervention graph, structural causal models

1 Introduction

Among causal queries, counterfactual questions of the form "What if X had been different?" are crucial for explanation, diagnosis, credit assignment, and decision making [16, 9]. To situate its value, it is helpful to distinguish probabilistic, causal, and counterfactual modeling along Pearl's "ladder of causality" [15]. While probabilistic models describe associations, Structural Causal Models (SCMs) additionally specify directed functional relationships between variables with exogenous noise and how these relationships change

under interventions [15]. This hierarchy of associational, interventional, and counterfactual queries captures increasingly expressive forms of reasoning, and in particular, the capacity for counterfactual reasoning is a critical capability for artificial intelligence systems to discern effects of alternative choices by explicitly relating the actual world to hypothetical variants of a causal model's structure under intervention to explain the observation(s) produced by the underlying generative mechanisms in either world [8, 6].

Why PLP? Counterfactuals in SCMs traditionally followed a procedure of abduction, action, and prediction [15], but Hopkins and Pearl argue SCMs become unwieldy in realistic domains as they lack first-order expressivity over entity and attribute relationals, quantified rules, and temporally structured actions [10]. For example, the classic "two-riflemen" scenario [15] requires a model that distinguishes multiple agents, possible misfires, and a disjunctive causal mechanism. A propositional encoding must enumerate each marksman A and B taking aim at a target and whether they hit the target separately, whereas a relational encoding expresses the causal mechanism compactly: `hits(Target) :- fired(Soldier,Target).`

The broader lesson is that causal models benefit from symbolic structure, and symbolic systems benefit from principled causal semantics [10]. Probabilistic Logic Programming (PLP) sits naturally at this intersection. It provides explicit *symbolic mechanisms* as interpretable rules describing how entities and relations interact, combined with probabilistic uncertainty. Such rules (e.g., `causes(Smoking,Cancer)`, `infects(P1,P2)`) support recursion, quantification, and relational generalization [22, 21]. However, strictly speaking PLP languages were not designed for causal inference [15, 21, 22, 3]. These languages lack a *do*-operator, formal intervention semantics, and tools for causal identification.

A simple ProbLog program illustrates this gap:

```
0.3::lifestyle(alice)
0.3::smokes(alice).
0.6::genetic_risk(alice).
cancer(alice) :- smokes(alice), genetic_risk(alice).
```

Under distribution semantics, this program defines a joint probability over `smokes(alice)`, `genetic_risk(alice)`, and `cancer(alice)` with logical dependencies between them. Conditioning on `smokes(alice)=false` updates beliefs on Alice's likelihood of cancer but does not implement the causal intervention (`smokes(alice)`$=false$), which requires deleting the generative mechanism between smoking and cancer. This difference between conditioning and intervening marks the conceptual boundary between probabilistic and causal interpretation.

Recognizing this limitation, several PLP languages extend logic programming with causal meaning. Logic Program Annotated Disjunctions (LPADs), [23], Causal Probabilistic-logic (CP-logic) [22], and ProbLog [4] introduce probabilistic choices, and CP-logic interprets rules as probabilistic causal laws. Its intervention semantics disable or modify such laws, aligning CP-logic with SCM-style reasoning. However, computing counterfactuals still follows the abduction, action, prediction procedure and requires storing the full posterior $P(\mathbf{U} \mid \mathbf{E} = \mathbf{e})$ over exogenous causes $\mathbf{U}$ given the evidence or observations $\mathbf{e}$.

To address this, another line of work adapts the *Twin Network* construction from SCMs to ProbLog [1, 11]. Counterfactual inference is reduced to ordinary probabilistic inference by duplicating the program into factual and counterfactual copies linked by shared exogenous variables avoiding storing $P(U \mid E = e)$ and doing intervention and inference at once to compute $P(Y|do(X = x), E = e)$. While convenient, this approach suffers from (i) exponentially many cross-world independence assumptions [17, 20], and (ii) demonstrable failures of these assumptions in important causal structures. In a practical sense, it is also limited for first-order relational models as it doubles program size, increasing compilation cost.

1.1 Our Contributions

In this work, we introduce *Single-World Intervention Programs (SWIPs)*, a new method for counterfactual reasoning in ProbLog inspired by the Single-World Intervention Graph (SWIG) framework [17]. Rather than duplicating the model, SWIPs perform a semantics-preserving transformation of the program itself.

Returning to the earlier example, to compute $P($`cancer(alice)` $\mid$ (`smokes(alice)`$=0)$), SWIP removes the probabilistic fact `0.3::smokes(alice).`, inserts the deterministic fact `smokes(alice)=false`, and manipulating the rule for `cancer(alice)` by removing its dependence on the original smoking mechanism while making it reliant on the intervention. More generally, given $(X{=}x)$, the SWIP transformation deletes all clauses defining $\mathbf{X}$, inserts the deterministic fact(s) asserting $\mathbf{X} := \mathbf{x}$ in place of $\mathbf{X}$, and eliminates redundant or unreachable rules through structural simplification.

The resulting SWIP is a simplified ProbLog program whose distribution matches the counterfactual semantics of the corresponding SCM. SWIPs offer practical advantages. Across extensive synthetic experiments, SWIPs produce significantly smaller unfolded programs and reduce compilation and inference time by approximately 35% relative to Twin Networks. Since SWIPs simplify rather than duplicate rule structure, knowledge-compilation backends (d-DNNF, `SHARPSAT`) exploit the reduced treewidth directly [5]. We prove that (1) for any SCM encodable in ProbLog, SWIPs reproduce exactly the interventional and counterfactual distributions of the SCM under standard assumptions of unique supported models and faithful SCM encodings; (2) the SWIP approach is computationally less expensive than the Twin Network approach; and (3) the grounded SWIP semantics coincide with CP-logic's intervention semantics, unifying event-based and equation-based causal interpretations.

To our knowledge, SWIPs provide the first single-world, SCM-faithful counterfactual semantics inside ProbLog that avoid cross-world assumptions while retaining the expressive relational structure of logic programming.

2 Preliminaries

2.1 Causal Models and Counterfactuals

A Structural Causal Model (SCM) $\mathcal{M}$ is defined as a tuple $\langle \mathbf{U}, \mathbf{V}, \mathcal{F} \rangle$, where $\mathbf{U}$ are mutually independent exogenous variables, $\mathbf{V}$ are endogenous variables, and $\mathcal{F} = \{f_V\}_{V \in \mathbf{V}}$

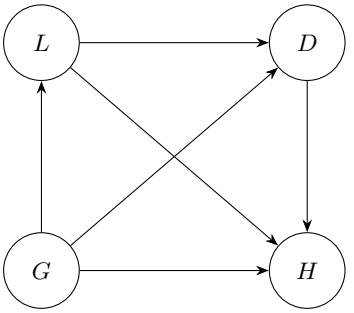


Fig. 1: A causal diagram with variables L, D, G, and H. The corresponding SCM declares H as a function of A, B and C, D as a function of G and L, and L as functions of G.

is a set of structural equations of the form $V := f_V(\text{pa}(V), U_V)$, with $\text{pa}(V) \subseteq \mathbf{V}$ and $U_V \subseteq \mathbf{U}$. Each SCM induces a directed acyclic graph (DAG) $\mathcal{G}$ encoding the causal dependencies among variables [15]. Figure 1 depicts a causal model between genetics G, lifestyle L, diet D, and health H while omitting the implicit exogenous noise variables for simplicity.

An intervention replaces the equations for a subset $\mathbf{X} \subseteq \mathbf{V}$ with constant assignments $\mathbf{X} := \mathbf{x}$, producing a modified model $\mathcal{M}_{\mathbf{x}}$. The induced post-interventional distribution is $P(\mathbf{v} \mid do(\mathbf{X} := \mathbf{x})) = \prod_{V_i \in \mathbf{V} \setminus \mathbf{X}} P(v_i \mid \text{pa}(V_i)) \cdot \mathbb{I}[x]$ in $\mathcal{M}_{\mathbf{x}}$. Counterfactual queries, $P(Y \mid do(\mathbf{X} := \mathbf{x}), \mathbf{E}{=}\mathbf{e})$, ask about outcomes under hypothetical interventions given observed evidence. Their computation typically follows the abduction, action, and prediction steps of [1]. *Abduction* requires storing the entire distribution of $P_{\mathcal{M}}(\mathbf{U}|\mathbf{E}{=}\mathbf{e})$, while *action* manipulates the model $\mathcal{M}$ into the intervened distribution $\mathcal{M}_{\mathbf{x}}$ for some assignment $\mathbf{X} := \mathbf{x}$, and *prediction* finally calculates $P_{\mathcal{M}_{\mathbf{x}}}(\mathbf{Y}|\mathbf{E} = \mathbf{e})$ [15]. [1, 11], but The twin network circumvents the abduction step and combines action and prediction by taking $\mathcal{M}$ and creating $\mathcal{M}^K$ given by the tuple $\langle \mathbf{U}, \mathbf{V}' \cup \mathbf{V}, \mathcal{F} \rangle$. It sets $\mathbf{V}' = \mathbf{V}$ and uses it to create a factual and counterfactual set of equations with shared exogeneity defined as

$$X := \begin{cases} f_X(pa(X), U_X), & X \in \mathbf{V} \\ f_X(pa(X)', U_X), & X \in \mathbf{V}' \end{cases}$$

where $pa(X)' = \{X' | X \in pa(X)\}$. Interventions are set on the variables in $\mathbf{V}'$, ie. $\mathbf{X}' := \mathbf{x}$ and we have:

$$P_{\mathcal{M}}(\cdot \,|\mathbf{E} = \mathbf{e}, do(\mathbf{X} := \mathbf{x})) = P_{\mathcal{M}^K_{\mathbf{x}}}(\cdot \,|\mathbf{E} = \mathbf{e})$$

Alternatively, the Single-World Intervention Graph (SWIG) formalism provides a convenient graphical encoding of the abduction, action, and prediction steps used to evaluate counterfactuals [17, 15]. Given a graph $\mathcal{G}$ encoding an SCM $\mathcal{M}$ and an intervention $do(\mathbf{X} := \mathbf{x})$, a SWIG is obtained by splitting each intervened node $X \in \mathbf{X}$ into two: a factual copy X that receives incoming edges from $pa(X)$ and a fixed counterfactual copy $X' := x$ with outgoing edges to its original children. Exogenous variables remain as is, yielding a single graph that represents factual and counterfactual variables within one world while avoiding ad hoc duplication of independent noise sources. Following

standard graphical separation criteria, the SWIG is a complete independence oracle and determines which counterfactual queries are identified by observed data [17].

2.2 ProbLog

ProbLog is a probabilistic logic programming language with distribution semantics [4]. A ProbLog program $\mathcal{P}$ is a pair $(\mathrm{LP}(\mathcal{P}), \mathrm{Facts}(\mathcal{P}))$. $\mathrm{Facts}(\mathcal{P})$ is a set of probabilistic facts of the form $\pi :: a$, where a is a ground atom from the set of external propositions $\mathfrak{E}(\mathfrak{B})$ and $\pi \in [0, 1]$ is its probability. These facts are assumed to be mutually independent and correspond to the exogenous variables $\mathbf{U}$ in an SCM. $\mathrm{LP}(\mathcal{P})$ is a set of logical clauses (rules) of the form $h \leftarrow b_1, \ldots, b_n$, where h is an internal proposition from $\mathfrak{I}(\mathfrak{P})$ and the body $\{b_1, \ldots, b_n\}$ is a set of literals. These rules correspond to the structural equations $\mathcal{F}$ in an SCM.

The semantics of $\mathcal{P}$ define a probability distribution $\pi_{\mathcal{P}}$ over possible worlds ω, where a world is a complete truth assignment to all ground atoms in $\mathfrak{B} = \mathfrak{J}(\mathfrak{B}) \cup \mathfrak{E}(\mathfrak{B})$. Formally, the probability of a world ω is $\pi_{\mathcal{P}}(\omega) = \prod_{a_i \in \omega} \pi_i \prod_{a_i \notin \omega} (1 - \pi_i)$ for all probabilistic facts $a_i \in \mathit{Facts}(P)$, i.e., the product of the probabilities of all true probabilistic facts and the complements of those that are false. The probability of a query ϕ is the sum of the probabilities of all worlds in which ϕ is true[4]:

$$\pi_{\mathcal{P}}(\phi) = \sum_{\omega \models \phi} \pi_{\mathcal{P}}(\omega).$$

For a ProbLog program to correctly represent an SCM, it must have unique supported models, meaning that for any truth assignment to the external propositions $\mathfrak{E}(\mathfrak{P})$, the logical rules in $\mathrm{LP}(\mathcal{P})$ must yield a single, unique truth assignment for all internal propositions $\mathfrak{I}(\mathfrak{P})$. A sufficient condition for this property is that the dependency graph of $\mathrm{LP}(\mathcal{P})$ is acyclic [12].

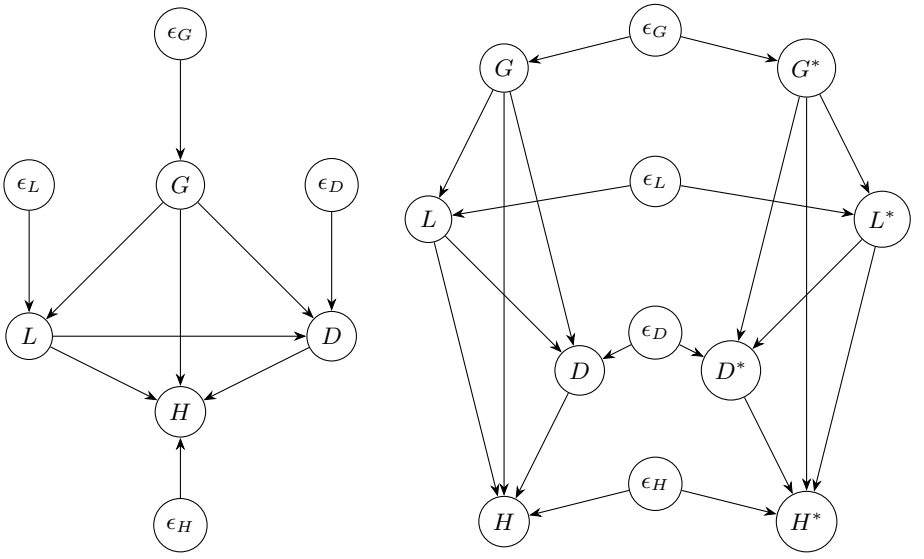
Within ProbLog, Pearl's d-separation reasoning for independence and identifiability can be implemented declaratively as a meta-interpreter that encodes both the syntactic rules of do-calculus and the associated independence checks as higher-order logic predicates [19]. This enables automated symbolic reasoning about causal identifiability directly within the ProbLog environment.

3 Challenges of the Twin Network in ProbLog

This section analyses a prevailing method for computing counterfactuals in ProbLog through a program transformation that constructs a Twin Network [12]. This approach operationalizes the SCM framework by creating a new ProbLog program, $\mathcal{T}(\mathcal{P})$, that explicitly represents both a factual and a counterfactual world. The transformation systematically duplicates all internal propositions $\mathfrak{I}(\mathfrak{P})$ and their defining rules in $\mathrm{LP}(\mathcal{P})$, while the external propositions in $\mathrm{Facts}(\mathcal{P})$ remain shared between the two worlds. This sharing of exogenous variables is the mechanism that relates the factual and counterfactual outcomes and the full procedure is detailed in Algorithm 1.

This approach, however, implicitly relies on a strong and untestable cross-world assumption of all of these exogenous variables being independent of one another and in fact, the number of assumptions required grows at a doubly exponential rate [20, 17].

The duplication of the entire program by nature introduces higher compute cost for program compilation in proportion to the the program size and length of clauses in the program. In the worst case, it is $\Theta(|\mathcal{P}| \cdot L_{\max})$



(a) Original causal model

(b) Twin network with counterfactual variables

Fig. 2: Side-by-side comparison of (a) the original structural causal model and (b) its corresponding twin network construction.

Theorem 3.1 (*Twin Network Transformation Complexity*)
Let $\mathcal{P} = (\text{LP}(\mathcal{P}), \text{Facts}(\mathcal{P}))$ be a ProbLog program with $|\mathcal{P}| = |\text{LP}(\mathcal{P})| + |\text{Facts}(\mathcal{P})|$ denoting the total number of clauses and facts. Let $L_{\max}$ be the maximum body length of any clause in $\text{LP}(\mathcal{P})$. Then the Twin Network transformation $\mathcal{T}(\mathcal{P})$ following Algorithm 2 from Kiesel et al. (2023) has complexity $\Theta(|\mathcal{P}| \cdot L_{\max})$.

An intervention $do(\mathbf{X} := \mathbf{x})$ is applied to the program by modifying the rules in the counterfactual part of the program. Subsequently, a counterfactual query $P_{\mathcal{M}^{\mathcal{K}}_{\mathbf{x}}}(Y|\mathbf{E} = \mathbf{e})$ is evaluated by computing a standard marginal probability on the transformed program $\mathcal{T}(\mathcal{P})$, as shown in Algorithm 2. The inference complexity itself is determined by an algorithm, such as knowledge compilation, whose runtime is given by a function $g(w(G))$ for a program with primal graph G of treewidth given by $w(G)$ [5]. A necessary condition for the validity of this approach for a counterfactual queries is the original program P has unique supported models, for which a sufficient condition is an acyclic underlying logic program $\text{LP}(\mathcal{P})$.

Algorithm 1 ConstructTwinNetwork($\mathcal{P}, do(\mathbf{X} := \mathbf{x})$)

Require: ProbLog program $\mathcal{P} = (\text{LP}(\mathcal{P}), \text{Facts}(\mathcal{P}))$; intervention $\mathbf{X} := \mathbf{x}$
Ensure: Twin Network program $\mathcal{T}(\mathcal{P})$
Initialize $\mathcal{T}(\mathcal{P}) \leftarrow \emptyset$
for all $p :: a \in \text{Facts}(\mathcal{P})$ **do**
  if $a \in \mathbf{X}$ **then**
    Add $1.0 :: a$ and $0.0 :: a'$ to $\mathcal{T}(\mathcal{P})$
  else
    Add $p :: a$ and $p :: a'$ to $\mathcal{T}(\mathcal{P})$
  end if
end for
for all $h \leftarrow b_1, \ldots, b_n \in \text{LP}(\mathcal{P})$ **do**
  if $h \notin \mathbf{X}$ **then**
    Add $h \leftarrow b_1, \ldots, b_n$ to $\mathcal{T}(\mathcal{P})$
    Add $h' \leftarrow b'_1, \ldots, b'_n$ to $\mathcal{T}(\mathcal{P})$
  end if
end for
return $\mathcal{T}(\mathcal{P})$

Algorithm 2 EvaluateTwinNetworkQuery($\mathcal{P}, \mathcal{T}(\mathcal{P}), \mathbf{X} := \mathbf{x}, \mathbf{E} = \mathbf{e}, \phi$)

Require: Original program $\mathcal{P}$, Twin Network $\mathcal{T}(\mathcal{P})$, intervention $\mathbf{X} := \mathbf{x}$, evidence $\mathbf{E} = \mathbf{e}$, query formula ϕ
Ensure: Counterfactual probability $\pi_{\mathcal{P}}(\phi_{\mathbf{x}} | \mathbf{E} = \mathbf{e})$
$p_1 \leftarrow \pi_{\mathcal{P}}(\mathbf{E} = \mathbf{e})$
$p_2 \leftarrow \pi_{\mathcal{T}(\mathcal{P})}(\phi' \wedge \mathbf{E} = \mathbf{e})$
return p_2 / p_1

4 Single World Intervention Programs

To address both limitations, we introduce our *Single-World Intervention Fact Transformation* (SWIFT) algorithm detailed in Algorithm 3 to produce a Single World Intervention Program (SWIP) fit for a counterfactual query given an intervention and evidence.

The SWIFT algorithm operationalizes the "graph surgery" of Single-World Intervention Graphs [17], shown in Figure 3, at the level of logical rules. Because rules interact through relational, quantified, and recursive dependencies, altering or deleting a clause may change the support, reachability, and logical structure of downstream atoms. The Twin Network approach avoids this by model duplication, at the cost of additional assumptions and computational overhead. Instead, the SWIFT algorithm, provides a rule rewriting procedure that preserves the deterministic closure implied by existing rules and unique supported model requirement by the distribution semantics to produce a program semantically equivalent to the intervened Structural Causal Model.

To distinguish interventiosn in the Twin Network setting and the SWIP setting, we use the $fix(\mathbf{X} := \mathbf{x})$ notation. This procedure first removes all clauses from $\text{LP}(\mathcal{P})$ that define the intervened propositions in $\mathbf{X}$. This step corresponds to severing the causal arrows

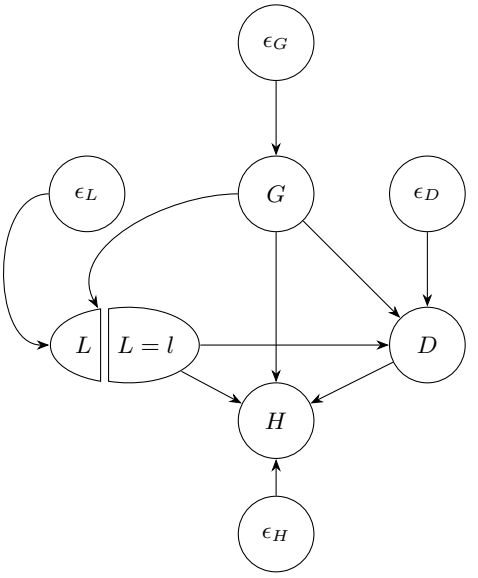


Fig. 3: SWIG where intervention $fix(L = l)$ is acted

into the intervened nodes. Second, it iterates through the remaining rules and replaces any occurrence of an intervened atom $X_i \in \mathbf{X}$ in a rule body with a new, unique atom $X_{i,fixed}$ that represents its fixed value. This corresponds to redirecting the outgoing causal arrows from the original random node to the new fixed-value node. Finally, it adds deterministic facts to assert the values of these new fixed atoms. In contrast to constructing $\mathcal{T}(\mathcal{P})$, the complexity of SWIFT is at worst proportional to the program size, ie. when an intervened variable appears as an atom in every rule, while for $\mathcal{T}(\mathcal{P})$, it is always proportional. The resulting SWIP, $\mathcal{S}(\mathcal{P})$, is a valid ProbLog program that directly represents the counterfactual world.

Algorithm 3 SWIFT($\mathcal{P}, fix(\mathbf{X} := \mathbf{x})$)

Require: ProbLog program $\mathcal{P} = (\text{LP}(\mathcal{P}), \text{Facts}(\mathcal{P}))$; intervention $\mathbf{X} := \mathbf{x}$
Ensure: Single-World Intervention Program $\mathcal{S}(\mathcal{P})$
Initialize $\mathcal{S}(\mathcal{P}) \leftarrow \text{Facts}(\mathcal{P})$
Let $\text{LP}_{-\mathbf{X}} \leftarrow C \in \text{LP}(\mathcal{P}) : head(C) \notin \mathbf{X}$
for all $C = (h \leftarrow b_1, \ldots, b_n) \in LP_{\mathbf{X}}$ **do**
    Let $B' = b'_1, \ldots, b'_n$ be a new set of body literals
    for all $b_j \in b_1, \ldots, b_n$ **do**
        Let a be the atom of the literal b_j.
        if $a \in \mathbf{X}$ **then**
            $b'_j \leftarrow$ literal corresponding to $a_{fixed}(x_a)$with the same sign as b_j.
        else
            $b'_j \leftarrow b_j$
        end if
    end for
    Add the rewritten rule $h \leftarrow B'$ to $\mathcal{S}(\mathcal{P})$
end for
for all $X_i \in \mathbf{X}$ **do**
    Add the fact $1.0 :: X_{i,fixed}(x_i)$ to $\mathcal{S}(\mathcal{P})$
end for
return $\mathcal{S}(\mathcal{P})$

Theorem 4.1 (*SWIP Transformation Complexity*)
Let $\mathcal{P} = (\text{LP}(\mathcal{P}), \text{Facts}(\mathcal{P}))$ be a ProbLog program, and let $|\mathcal{P}| = |\text{LP}(\mathcal{P})| + |\text{Facts}(\mathcal{P})|$ denote the total number of clauses and facts. Let $L_{\max}$ be the maximum body length of any clause in $\text{LP}(\mathcal{P})$. Then the SWIP $\mathcal{S}(\mathcal{P})$ following Algorithm 3 has complexity $O(|\mathcal{P}| \cdot L_{\max})$.

As a corollary, it immediately follows

Corollary 4.1.1 (*Asymptotic Advantage of SWIFT over Twin Networks*)
Let $\mathcal{P} = (\text{LP}(\mathcal{P}), \text{Facts}(\mathcal{P}))$ be a ProbLog program of size $|\mathcal{P}|$, and let $L_{\max}$ be the maximum body length of any clause in $\text{LP}(\mathcal{P})$. Let $\mathcal{S}(\mathcal{P})$ be the Single-World Intervention Program produced by the SWIFT transformation, and let $\mathcal{T}(\mathcal{P})$ be the Twin Network transformation of $\mathcal{P}$.

Then the time complexity of constructing $\mathcal{S}(\mathcal{P})$ is $O(|\mathcal{P}| \cdot L_{\max})$, while the time complexity of constructing $\mathcal{T}(\mathcal{P})$ is $\Theta(|\mathcal{P}| \cdot L_{\max})$. Moreover, $\mathcal{S}(\mathcal{P})$ is never asymptotically larger than $\mathcal{T}(\mathcal{P})$, and is strictly smaller whenever at least one intervened atom does not appear in all clause bodies.

That is to say, while asymptotically equivalent in the worst case, SWIPs avoid unconditional duplication and are strictly smaller for sparse interventions. This directly translates to query speed ups in the general case, and in the worst case, is the same cost as querying over $\mathcal{T}(\mathcal{P})$. As shown in Algorithm 4, evidence is incorporated by adding facts to $\mathcal{S}(\mathcal{P})$, and the counterfactual probability is obtained via standard marginal inference on this final program.

Algorithm 4 EvaluateSWIPQuery($\mathcal{S}(\mathcal{P}), \mathbf{E} = \mathbf{e}, \phi$)

Require: SWIP $\mathcal{S}(\mathcal{P})$; evidence $\mathbf{E} = \mathbf{e}$; counterfactual query formula ϕ
Ensure: Counterfactual probability $\pi_{\mathcal{S}(\mathcal{P})}(\phi|\mathbf{E} = \mathbf{e})$
1: $p_1 \leftarrow \pi_{\mathcal{S}(\mathcal{P})}(\mathbf{E} = \mathbf{e})$
2: $p_2 \leftarrow \pi_{\mathcal{S}(\mathcal{P})}(\phi \wedge \mathbf{E} = \mathbf{e})$
3: **return** p_2/p_1

Theorem 4.2 (*Inference Complexity Comparison*)
Let $g(\cdot)$ be the complexity of some inference algorithm. Then inference complexity for querying over $\mathcal{S}(\mathcal{P})$ vs $\mathcal{T}(\mathcal{P})$ is

$$O(g(w(\mathcal{S}(\mathcal{P})))) \leq O(g(w(\mathcal{T}(\mathcal{P}))))$$

Because our transformation is proven to yield a program whose semantics are equivalent to the counterfactual distribution of the underlying SCM, it inherits the established consistency with other causal formalisms like CP-logic and LPADs [12, 22, 23].

Theorem 4.3 (*Correctness of SWIP-Based Counterfactual Queries*)
Let P be a ProbLog program encoding a structural causal model M with unique supported models. Let $\mathcal{S}(\mathcal{P}) = \text{SWIFT}(\mathcal{P},\ fix(\mathbf{X} := \mathbf{x}))$ be the SWIG-transformed program and let $\mathcal{S}^e(\mathcal{P})$ be the program augmented with evidence $\mathbf{E} = \mathbf{e}$. Then for any query ϕ under intervention $\mathbf{x}$,

$$\pi_{\mathcal{S}^e(\mathcal{P})}(\phi) = \pi_{\mathcal{M}}(\phi_{\mathbf{x}}|\mathbf{E} = \mathbf{e}).$$

These theorems suggest with respect to program construction, the Twin Networks unconditionally duplicates the program, regardless of the intervention query while, the SWIP approach scales slower with program size and maximum clause body length and at worst case, will incur the same cost as the Twin Network approach. In realistic applications, where programs and evidence sets may be large but interventions focused on a small subset of predicates, SWIPs can take advantage of the locality and specificity of the query structure. As a result, SWIPs naturally favor compact counterfactual representations and furthermore, are a more realistic approach for capturing real world counterfactuals. Beyond the issue of cross-world independence assumptions growing at a doubly exponential rate, Richardson and Robins (2013) strongly emphasize the assumptions of the Twin Network are, by definition, mutually exclusive and thus, experimentally unverifiable [17]. SWIGs, and by extension, SWIPs, ability to avoid this is especially salient in sequentially randomized trials and longitudinal decision problems. In such settings, treatments are assigned over time based on evolving histories, and counterfactual reasoning must respect the temporal and logical structure of these assignments. Richardson and Robins show that Twin Network constructions can induce incorrect independencies in these cases, whereas SWIGs preserve the correct causal structure by explicitly representing interventions as node-splitting operations within a single world [17]. Our results show that SWIPs inherit this advantage at the level of probabilistic logic programs: in-

terventions modify only the clauses corresponding to treatment assignment mechanisms, while leaving downstream deterministic and probabilistic dependencies intact.

We can characterize these counterfactual independence conclusions in PLP contexts when not using SWIPs and show our method is a more general and robust implementation of the SCM semantics and is consistent with CP-logic over the same set of models as the Twin Network.

Theorem 4.4 (*SWIP Consistency with LPAD*)
Let $\mathcal{P}$ be a propositional LPAD-program such that every selection yields a logic program with a unique supported model. Let $\mathbf{X}, \mathbf{E} \subseteq \mathfrak{B}$ be sets of propositions with value assignments $\mathbf{x}$ and $\mathbf{e}$, respectively, and let ϕ be a $\mathfrak{P}$-formula.

Denote by $\pi^{CP}_{\mathcal{P}}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x}))$ the counterfactual probability computed by CP-logic using the fixed-operator semantics on LPADs, and by

$$\pi^{SWIP}_{\mathrm{Prob}(\mathcal{P})}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x}))$$

the counterfactual probability induced by the corresponding Single World Intervention Program (SWIP) constructed from Prob($\mathcal{P}$).

Then,

$$\pi^{CP}_{\mathcal{P}}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x})) \;=\; \pi^{SWIP}_{\mathrm{Prob}(\mathcal{P})}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x})).$$

Theorem 4.5 (*Consistency of SWIPs with CP-Logic*)
Let $\mathcal{P}$ be a ProbLog program with unique supported models, and let $\mathbf{X}, \mathbf{E} \subseteq \mathfrak{B}$ with value assignments $\mathbf{x}$ and $\mathbf{e}$. For any $\mathfrak{P}$-formula ϕ, denote by

$$\pi^{CP}_{\mathrm{LPAD}(\mathcal{P})}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x}))$$

the counterfactual probability defined via CP-logic on the LPAD-transformation of $\mathcal{P}$, and by

$$\pi^{SWIP}_{\mathcal{P}}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x}))$$

the counterfactual probability computed by the SWIP semantics directly on $\mathcal{P}$.

Then,

$$\pi^{CP}_{\mathrm{LPAD}(\mathcal{P})}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x})) \;=\; \pi^{SWIP}_{\mathcal{P}}(\phi \mid \mathbf{E} = \mathbf{e}, fix(\mathbf{X} := \mathbf{x})).$$

For example, consider a simple power failure system that involves a deterministic relationship that creates a logical constraint. Let A be the main power supply, B the independent backup power supply, C an indicator for if the system is on, and D a deterministic report filed iff C is true. As a ProbLog program, $\mathcal{P}$:

```
% Exogenous variables for the two power supplies
0.5::u_a
0.5::u_b

% Endogenous variables defined by structural rules
a :- u_a.
```

```
b :- u_b.
c :- a.
c :- b.
d :- c.
```

The atom a represents the main power being active, b represents the backup being active, c represents the system being on, and d represents the report being filed. The rule `d :- c` establishes the deterministic link. Now consider the query, "Given we know the back up generator was on, is the report being filed independent of the main generator?" or formally, $D \perp A|B = 1$. According to standard do-calculus on the Twin Network of this program, we see that D is not independent of A but of course, the status of A is no longer relevant as $B = 1$ fully informs us of D. Indeed, in the ProbLog d-separation metainterpreter by [19], the twin network program will not reveal this dependency but it is trivially identified in the SWIP.

The SWIG literature emphasizes that single-world node-splitting correctly exposes many counterfactual conditional independencies that cannot be read from a naive multi-world duplication via d-separation without further assumptions [17]. All d-separation claims in this paper are evaluated on the grounded dependency graph corresponding to the induced Structural Causal Model of the ProbLog program. Concretely, this graph is obtained by grounding the program and interpreting probabilistic facts as mutually independent exogenous variables and logical clauses as directed functional dependencies between endogenous variables. The following formal statement makes this concrete in the ProbLog setting and generalizes the power failure example.

Theorem 4.6 (*Misidentified Counterfactual Independencies in Twin Network Programs*)

Let $\mathcal{P}$ be a ProbLog program encoding an SCM $\mathcal{M} = \langle \mathbf{U}, \mathbf{V}, \mathcal{F} \rangle$. Suppose there exist distinct endogenous atoms $A, B, D \in \mathbf{V}$ such that:

(1) D is defined in LP($\mathcal{P}$) by a set of clauses whose combined effect is a deterministic function $d = g(a, b, \mathbf{u}_D)$ (possibly expressed via multiple rules), where $\mathbf{u}_D$ denotes the exogenous input(s) relevant to D;
(2) there exists a value $b^\star$ for B with the *screening property*

$$g(a, b^\star, u_D) = g(a', b^\star, u_D) \quad \forall a, a' \text{ and all } \mathbf{u}_D,$$

i.e. when $B = b^\star$ the value of D is pointwise independent of A given the same exogenous input u_D;
(3) A and B are not d-separated by the empty set in the causal graph underlying $\mathcal{P}$ and the program satisfies unique supported model conditions so the ProbLog semantics is well defined.

Then the following hold:

(a) On the SWIP for the intervention (B=$b^\star$) the node $D_{b^\star}$ is d-separated from A (possibly conditioning on nothing or on appropriate observed variables), and hence the SWIP implies the counterfactual independence $D \perp A|B = b^*$
(b) There exist programs $\mathcal{P}$ satisfying (1) - (3) for which the Twin Network construction $\mathcal{T}(\mathcal{P})$ does not d-separate the factual atom A from the counterfactual copy D^*. Consequently, ordinary *do*-calculus on $\mathcal{T}(\mathcal{P})$ will not soundly identify the independence.

In the power failure example, it is trivial to see this is a result of the direct dependency between D and C. However, the value of a counterfactual program transformation which is a complete independence oracle is highlighted by condition (1) of Theorem 4.6, where D may be expressed, unobviously, deterministically as a composition of multiple rules. The above results demonstrate the failure of the Twin Network is not a PLP issue nor can PLP alone reveal independencies in program structure when interventions are implemented across worlds [19][18]. SWIPs, by design, align functional causal semantics with the interventional theory of CP-Logic while avoiding independence pathologies.

5 Experiments

We've established that SWIPs can be used for counterfactual queries by reducing them to marginal inference and carry out this query via SharpSAT, a top down Knowledge Compilation [13]. To assess the scalability and efficiency of our SWIP counterfactual inference method, we replicate and extend the experimental setup from [11]. Our evaluation focuses on three primary questions: (i) how counterfactual program size varies with the SWIP approach and the Twin Network approach, (ii) how inference time varies with program size and structural complexity, and (iii) how inference time is affected by the number and type of evidence and intervention atoms.

Benchmark Instances Following [11], we generate acyclic directed graphs (DAGs) with a controlled size and treewidth. Each instance corresponds to a random probabilistic logic program modeling reachability in a directed graph. For a given graph $G = (V, E)$ with distinguished start and goal nodes $s, g \in V$, we encode the probability of reaching g from s using the following ProbLog schema:

```
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).
```

Here, $d(X)$ denotes the out-degree of vertex X, and $\texttt{s\_i}(X)$ denotes its i-th child node. The resulting program represents the random process of traversing the graph from s to g, avoiding nodes marked as traps. We vary two parameters controlling instance difficulty: the number of vertices n and the treewidth k. We first generate a random tree of size n using the `networkx` library (which has treewidth 1), and then add k additional nodes, each connected by incoming arcs from randomly selected original nodes. Finally, a single goal vertex g is added, receiving edges from each of the k new nodes. This procedure yields a DAG of size $n+k+1$ with treewidth $\min(n, k)$ and ensures acyclicity. For each instance, we sample up to five pieces of evidence and five interventions, allowing us to examine the interaction between query complexity and inference performance. Counterfactual queries are defined over this model by introducing positive or negative evidence on intermediate reachability predicates and positive or negative interventions on selected edges, following the schema: $\pi_{\mathcal{P}}^{\mathcal{M}}\big(r(g) \mid \neg r(v_1), \ldots, \neg r(v_n), (\neg r(v'_1)), \ldots, (\neg r(v'_m))\big)$, for some evidence nodes $v_1, \ldots, v_n$ and intervened nodes $v'_1, \ldots, v'_m$.

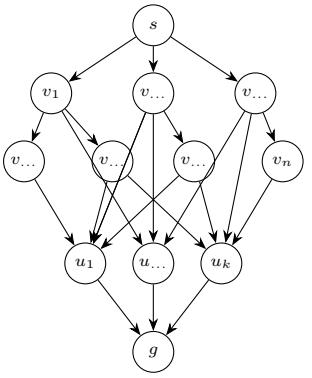


Fig. 4: Construction of benchmark DAGs. A random tree of size n rooted at s (top) is generated, a dense layer of k nodes $u_1, \dots, u_k$ is added below (each receiving edges from multiple tree nodes), and a single goal node g is appended below the dense layer receiving edges from every u_i.

All experiments were executed on the University of Edinburgh's compute cluster. Each node is equipped with two Intel Xeon Gold 5218 CPUs (16 cores per CPU, 2.30 GHz base frequency), with 256 GB of DDR4 RAM per node operating at 2666 MHz. The cluster runs Red Hat Enterprise Linux 8.6 and uses the Slurm workload manager for parallel execution. We performed all experiments using Python 3.9.21 and ProbLog 2.2, with inference powered by the `SHARPSAT` knowledge compiler for top-down inference. This compiler was shown in [11] to be fastest for counterfactual program query compilation. Each query was given a time limit of 1800 seconds; queries reaching the limit were assigned this maximum runtime for consistency with [11].

5.1 Results and Discussion

Figure 5 shows the unfolded treewidths of the compiled ProbLog programs as a function of the synthetic graph size and original treewidth. As predicted, the SWIG-based transformation produces programs with substantially smaller unfolded treewidths than the Twin Network baseline. This difference arises because our approach performs a localized "graph surgery," modifying only the clauses corresponding to intervened variables, rather than duplicating the entire logical program. By maintaining a single-world representation, the resulting dependency graph remains tighter and less entangled, leading to more compact knowledge-compilation structures. These smaller unfolded programs directly reduce the complexity of subsequent inference procedures, since treewidth is the dominant factor determining compilation cost in ProbLog-based inference [12, 5, 13].

This structural advantage translates directly into computational gains, as illustrated in Figure 6. Across all benchmark instances, the SWIG-based programs consistently compile and evaluate faster than their Twin Network counterparts. On average, our method requires approximately 65% of the runtime of the baseline for equivalent evidence and intervention configurations. The performance gap remains stable across varying program sizes and query complexities, confirming that reducing unfolded treewidth yields measurable improvements in both compilation and inference phases. Together, these results empirically validate that the SWIG-based transformation preserves the expressive and

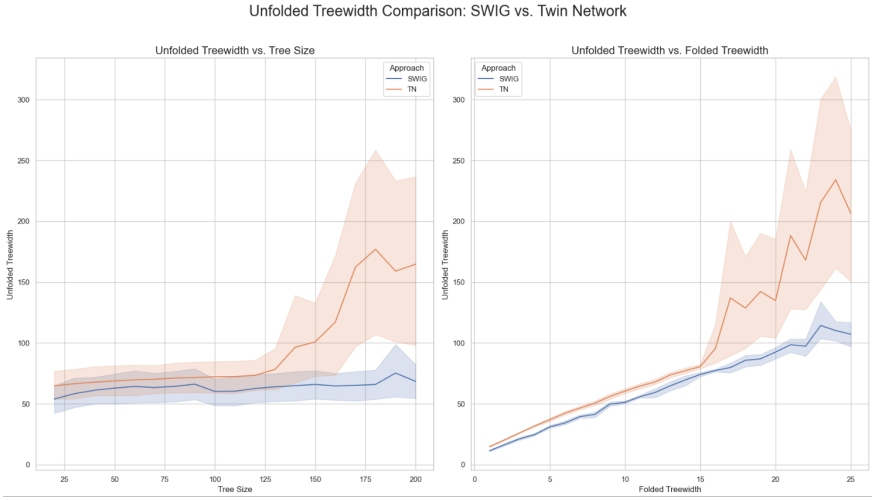


Fig. 5: Unfolded treewidth as a function of synthetic graph size and original treewidth. The SWIG-based transformation yields narrower unfolded dependency structures than the Twin Network approach.

causal semantics of counterfactual reasoning while significantly improving computational efficiency.

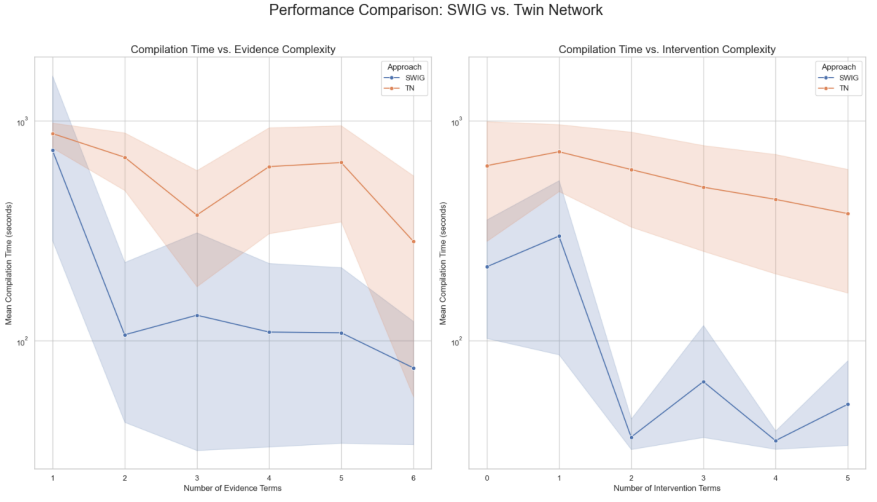


Fig. 6: Mean compilation and inference times for varying numbers of evidence and intervention atoms. The SWIG-based transformation achieves lower inference and compilation times, corresponding to its smaller unfolded program structures.

6 Conclusion

We have presented a SWIG-based approach to counterfactual reasoning in ProbLog. By "fixing" intervened nodes and propagating this intervention to descendant rules, we are

able to reduce counterfactual queries to standard marginal inference queries. Our procedure of a transformation under intervention and subsequent evidence incorporation, is proven to yield the counterfactual distribution $P(\cdot \mid do(X = x), \mathbf{E} = \mathbf{e})$ under assumptions of consistency and modularity. It also provides significant computational speedups in inference compared to existing approaches for counterfactual reasoning in ProbLog. In future work, it would be interesting to apply this approach to real-world settings and establishing counterfactual reasoning capabilities to DeepProbLog, an extension of ProbLog that combines it with neural network predicates to combine highlevel logical reason with low level subsymbolic perception [14]. Furthermore, exploring σ-calculus, a more general form of *do*-calculus [2, 7], to define counterfactuals for programs without unique supported models offers promising research directions.

Acknowledgments

Funding for this research was provided by NERC through an E4 DTP studentship (NE/S007407/1).

References

[1] Alexander Balke and Judea Pearl. "Probabilistic Evaluation of Counterfactual Queries". In: *Probabilistic and Causal Inference: The Works of Judea Pearl*. 1st ed. Vol. 36. New York, NY, USA: Association for Computing Machinery, Mar. 2022, pp. 237–254. ISBN: 978-1-4503-9586-1. URL: https://doi.org/10.1145/3501714.3501733 (visited on 04/16/2025).

[2] Juan Correa and Elias Bareinboim. "A Calculus for Stochastic Interventions:Causal Effect Identification and Surrogate Experiments". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.06 (Apr. 2020), pp. 10093–10100. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i06.6567. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6567 (visited on 02/04/2026).

[3] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. "ProbLog: a probabilistic prolog and its application in link discovery". In: *Proceedings of the 20th international joint conference on Artifical intelligence*. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jan. 2007, pp. 2468–2473. (Visited on 04/16/2025).

[4] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery". In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jan. 6, 2007, pp. 2468–2473.

[5] Thomas Eiter, Markus Hecher, and Rafael Kiesel. "Treewidth-Aware Cycle Breaking for Algebraic Answer Set Counting". en. In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning* 18.1 (Sept. 2021). Conference Name: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, pp. 269–279. ISSN: 2334-1033. DOI: 10.24963/kr.2021/26. URL: https://proceedings.kr.org/2021/26/ (visited on 08/21/2025).

[6] Kai Epstude and Neal J. Roese. "The Functional Theory of Counterfactual Thinking". In: *Personality and social psychology review : an official journal of the Society for Personality and Social Psychology, Inc* 12.2 (May 2008), pp. 168–192. ISSN: 1088-8683. DOI: 10.1177/1088868308316091. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC2408534/ (visited on 11/07/2025).

[7] Patrick Forré and Joris M. Mooij. *Constraint-based Causal Discovery for Non-Linear Structural Causal Models with Cycles and Latent Confounders*. arXiv:1807.03024 [stat]. July 2018. DOI: 10.48550/arXiv.1807.03024. URL: http://arxiv.org/abs/1807.03024 (visited on 02/04/2026).

[8] Tobias Gerstenberg. "Counterfactual simulation in causal cognition". In: *Trends in Cognitive Sciences* 28.10 (Oct. 2024), pp. 924–936. ISSN: 1364-6613. DOI: 10.1016/j.tics.2024.04.012. URL: https://www.sciencedirect.com/science/article/pii/S1364661324001074 (visited on 11/07/2025).

[9] Joseph Y. Halpern. *Actual Causality*. en. Cambridge, MA, USA: MIT Press, Feb. 2019. ISBN: 978-0-262-53713-1. URL: https://mitpress.mit.edu/9780262537131/actual-causality/ (visited on 11/21/2025).

[10] Mark Hopkins and Judea Pearl. "Causality and Counterfactuals in the Situation Calculus". In: *Journal of Logic and Computation* 17.5 (Oct. 2007), pp. 939–953. ISSN: 0955-792X. DOI: 10.1093/logcom/exm048. URL: https://doi.org/10.1093/logcom/exm048 (visited on 11/21/2025).

[11] Rafael Kiesel, Kilian Rückschloß, and Felix Weitkämper. *"What if?" in Probabilistic Logic Programming*. arXiv:2305.15318. May 2023. DOI: 10.48550/arXiv.2305.15318. URL: http://arxiv.org/abs/2305.15318 (visited on 03/21/2025).

[12] Rafael Kiesel, Kilian Rückschloß, and Felix Weitkämper. *"What If?" In Probabilistic Logic Programming*. May 24, 2023. DOI: 10.48550/arXiv.2305.15318. arXiv: 2305.15318. URL: http://arxiv.org/abs/2305.15318 (visited on 03/21/2025). Pre-published.

[13] Tuukka Korhonen and Matti Järvisalo. "Integrating Tree Decompositions into Decision Heuristics of Propositional Model Counters (Short Paper)". en. In: *LIPIcs, Volume 210, CP 2021* 210 (2021). Ed. by Laurent D. Michel. Artwork Size: 11 pages, 918439 bytes ISBN: 9783959772112 Medium: application/pdf Publisher: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 8:1–8:11. ISSN: 1868-8969. DOI: 10.4230/LIPICS.CP.2021.8. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2021.8 (visited on 08/21/2025).

[14] Robin Manhaeve et al. "Neural probabilistic logic programming in DeepProbLog". In: *Artificial Intelligence* 298 (Sept. 2021), p. 103504. ISSN: 0004-3702. DOI: 10.1016/j.artint.2021.103504. URL: https://www.sciencedirect.com/science/article/pii/S0004370221000552 (visited on 08/20/2025).

[15] Judea Pearl. *Causality*. 2nd ed. Cambridge: Cambridge University Press, 2009. ISBN: 978-0-521-89560-6. DOI: 10.1017/CBO9780511803161. URL: https://www.cambridge.org/core/books/causality/B0046844FAE10CBF274D4ACBDAEB5F5B (visited on 08/21/2025).

[16] Judea Pearl and Dana Mackenzie. *The Book of Why: The New Science of Cause and Effect*. 1st. USA: Basic Books, Inc., Apr. 2018. ISBN: 978-0-465-09760-9.

[17] T. Richardson. "Single World Intervention Graphs (SWIGs) : A Unification of the Counterfactual and Graphical Approaches to Causality". In:

2013. URL: https://www.semanticscholar.org/paper/Single-World-Intervention-Graphs-(-SWIGs-)-%3A-A-of-Richardson/fb11d632e389e8deca243491f7a65eb238556097 (visited on 08/21/2025).

[18] Kilian Rückschloß and Felix Weitkämper. "On the Independencies Hidden in the Structure of a Probabilistic Logic Program". In: *Electronic Proceedings in Theoretical Computer Science* 385 (Sept. 2023), 169–182. ISSN: 2075-2180. DOI: 10.4204/eptcs.385.17. URL: http://dx.doi.org/10.4204/EPTCS.385.17.

[19] Kilian Rückschloß and Felix Weitkämper. "On the Subtlety of Causal Reasoning in Probabilistic Logic Programming: A Bug Report about the Causal Interpretation of Annotated Disjunctions". en. In: ().

[20] Ilya Shpitser, Thomas S. Richardson, and James M. Robins. *Multivariate Counterfactual Systems And Causal Graphical Models*. Aug. 28, 2021. DOI: 10.48550/arXiv.2008.06017. arXiv: 2008.06017 [stat]. URL: http://arxiv.org/abs/2008.06017 (visited on 04/15/2025). Pre-published.

[21] Joost Vennekens, Maurice Bruynooghe, and Marc Denecker. "Embracing Events in Causal Modelling: Interventions and Counterfactuals in CP-Logic". en. In: *Logics in Artificial Intelligence*. Ed. by Tomi Janhunen and Ilkka Niemelä. Berlin, Heidelberg: Springer, 2010, pp. 313–325. ISBN: 978-3-642-15675-5. DOI: 10.1007/978-3-642-15675-5_27.

[22] Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. "Cp-logic: A language of causal probabilistic events and its relation to logic programming". In: *Theory Pract. Log. Program.* 9.3 (May 2009), pp. 245–308. ISSN: 1471-0684. DOI: 10.1017/S1471068409003767. URL: https://doi.org/10.1017/S1471068409003767 (visited on 08/06/2025).

[23] Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe. "Logic Programs with Annotated Disjunctions". en. In: *Logic Programming*. Ed. by Bart Demoen and Vladimir Lifschitz. Berlin, Heidelberg: Springer, 2004, pp. 431–445. ISBN: 978-3-540-27775-0. DOI: 10.1007/978-3-540-27775-0_30.

Appendix A Notation and Definitions

Table A 1: Summary of key notation used throughout the paper.

Symbol	Definition
$\mathbf{V}$	Set of endogenous variables
$\mathbf{U}$	Set of exogenous variables
$\mathcal{F}$	Set of functional equations
$pa(X)$	Set of endogenous parents of x
$\mathcal{M}$	Structural causal model (SCM): tuple of $\langle \mathbf{U}, \mathbf{V}, \mathcal{F} \rangle$
$X := f_X(\text{pa}(X), \epsilon_X)$	Structural equation for variable X w
$\text{do}(\mathbf{X} := \mathbf{x})$	Pearl's do-operator: modifies SCM equations of involving $\mathbf{X}$
$\text{fix}(\mathbf{X} := \mathbf{x})$	Fix-operator: constructs a Single-World Intervention Graph
$\mathbf{V}^*$	Counterfactual copies of variables in the Twin Network
$\mathcal{M}^K$	Twin Network SCM with shared exogenous variables
$\mathcal{P}$	ProbLog program with logic and random facts
$\text{LP}(\mathcal{P})$	Underlying logical clauses in a ProbLog program
$\mathfrak{B}$	Propositional alphabet for logic atoms
$\mathcal{T}(\mathcal{P})$	Transformed Twin Network ProbLog program
$\mathcal{S}(\mathcal{P})$	Transformed Single World Intervention Program

Appendix B Proofs of Theorems 3.1, 4.1, 4.2

Proof of Theorem 3.1
Consider that for each fact $r :: A \in \text{Facts}(\mathcal{P})$, we either intervene on r if A is in the intervention set $\mathbf{X}$ or we add the observed and counterfactual facts to our program $\mathcal{T}(\mathcal{P})$. This has a complexity of $\Theta(|\text{Facts}(\mathcal{P})|)$.

Similarly, for each clause $C = h \leftarrow b_1, \ldots, b_n \in \text{LP}(P)$ with $h \notin \mathbf{X}$:, we add the clause and the counterfactual copy to $\mathcal{T}(\mathcal{P})$. Contructing the body of the counterfactual clause takes $\Theta(n)$ per clause. Thus the total rule transformation complexity is:

$$\sum_{C \in \text{LP}(P)} \Theta(|\text{body}(C)|) = \Theta(|\text{LP}(P)| \cdot L_{\max})$$

Summing these steps together we get a complexity of $\Theta(|\mathcal{P}| \cdot L_{\max})$. □

Proof of Theorem 4.1
Analogous to Theorem 3.1, except we do not need to rewrite every clause, just the clauses with variables in the intervention set. Thus, we have $O(|\mathcal{P}| \cdot L_{\max})$ □

Proof of Theorem 4.2
Let A be the set of internal atoms. Then $\mathcal{T}(\mathcal{P})$ contains approximately $2|A|$ atoms and rules. However, as the primal graph of $\mathcal{T}(\mathcal{P})$, denoted $G_{\mathcal{T}(\mathcal{P})}$, is constructed over the set of endogenous atoms $A \cup A^*$ and no rule in $\mathcal{T}(\mathcal{P})$ contains both an atom from A and an atom from A^* in its body, the resulting primal graph consists of two disconnected

components. The treewidth of this composite graph, $w(\mathcal{T}(\mathcal{P}))$, is therefore equal to the treewidth of the primal graph of the original program, $w(\mathcal{P})$. Since inference (e.g., via knowledge compilation) has complexity as a function of treewidth, we have

$$O(g(w(\mathcal{T}(\mathcal{P})))) = O(g(w(\mathcal{P})))$$

Let $G_{\mathcal{S}(\mathcal{P})}$ be the primal graph of the transformed program $\mathcal{S}(\mathcal{P})$. Note that removing rules with heads in $\mathbf{X}$ deletes edges in G_P and that rewriting variables with fixed values does not add new dependencies. Therefore:

$$w(\mathcal{S}(\mathcal{P})) \leq w(P)$$

Hence, inference complexity is:

$$O(g(w(\mathcal{S}(\mathcal{P})))) \leq O(g(w(\mathcal{T}(\mathcal{P}))))$$

□

Appendix C Proofs of Theorem 4.4 and Theorem 4.5

To demonstrate the consistency of the SWIP treatment of counterfactuals with CP-logic we start by recalling the theory of CP-Logic from Vennekens et al. (2009) and the LPAD-programs of Vennekens et al. (2004) with their standard semantics. An LPAD program, $\mathbf{P}$, is a finite set of rules of the following form:

$$RC := h_1 : \pi_1; \ldots; h_l : \pi_l \leftarrow b_1, \ldots, b_n$$

where h_i and b_i are atoms and literals in some set of propositions $\mathfrak{B}$. We have that the $\pi_i \in [0,1]$ are associated probabilities for h_i such that $\sum_i \pi_i \leq 1$. We define $\text{head}(RC) := (h_1, \ldots, h_l)$ to be a tuple of propositions which are the head of RC, where $h \in (h_1, \ldots, h_l)$ if $h = h_i$ for a $1 \leq i \leq l$ and furthermore, $l(RC) := l$ and $h_i(RC) := h_i$ for $1 \leq i \leq l$. Similarly, the body of RC is the finite set of literals $\text{body}(RC) := \{b_1, \ldots, b_n\}$

A selection, σ, of $\mathbf{P}$ is a function $\sigma : \mathcal{P} \to \mathbb{N} \cup \{\bot\}$, where $\bot \notin \mathbb{N}$, that assigns to each LPAD-clause $RC \in \mathbf{P}$ a natural number in $[1, l]$ or $\sigma(RC) := \bot$. To each selection σ, we associate a probability

$$\pi(\sigma) := \prod_{\substack{RC \in \mathcal{P} \\ \sigma(RC) \in \mathbb{N}}} \pi_{\sigma(RC)}(RC) \prod_{\substack{RC \in \mathcal{P} \\ \sigma(RC) = \bot}} \left(1 - \sum_{i=1}^{l(RC)} \pi_i(RC)\right)$$

and each selection gives a logic program

$$\mathcal{P}^\sigma := \{h_{\sigma(RC)} \leftarrow \text{body}(RC) : RC \in \mathcal{P}, \sigma(RC) \neq \bot\}.$$

We define π^{dist}, the distribution semantics of $\mathcal{P}$, as

$$\pi^{dist}_{\mathcal{P}}(\phi) := \sum_{\substack{\sigma \text{ selection} \\ \mathcal{P}^\sigma \models \phi}} \pi(\sigma).$$

where ϕ is some $\mathfrak{P}$-formula.

Rigguzi (2020) §2.4 establishes we can can translate an LPAD program $\mathcal{P}$ in $\mathfrak{B}$ to a ProbLog program, $\text{Prob}(\mathcal{P})$ where the logic of the program $LP(\text{Prob}(\mathcal{P})$ is given by

choosing distinct propositions $h_i^{RC}, u_i(RC) \notin \mathfrak{B}$ for all $RC \in \mathcal{P}$ and natural number in $[1, l]$ and letting

$$h_i^{RC} \leftarrow \text{body}(RC) \cup \{\neg h_j^{RC} \mid 1 \leq j < i\} \cup \{u_i(RC)\},$$

$$h_i \leftarrow h_i^{RC}$$

Meanwhile, the random facts, Facts(Prob($\mathcal{P}$)) are defined as

$$\text{Facts}(\text{Prob}(\mathcal{P})) := \left\{ \frac{\pi_i(RC)}{1 - \prod_{1 \leq j < i} \pi_j(RC)} :: u_i(RC) \;\middle|\; RC \in \mathcal{P}, 1 \leq i \leq l \right\}.$$

This then ensures the following:

Theorem Appendix C.1 (*Riguzzi (2020), §2.4*)
Let $\mathcal{P}$ be a LPAD-program. Then, for every selection σ of $\mathcal{P}$ a set of possible worlds $\mathcal{E}(\sigma)$, which consists of all possible worlds $\mathcal{E}$ such that $\neg u_i(RC)$ holds unless $\sigma(RC) \neq \bot$ or $i > \sigma(RC)$ and such that $u_{\sigma(RC)}(RC)$ holds for every $RC \in \mathcal{P}$ with $\sigma(RC) \neq \bot$. We conclude that $\mathcal{P}^\sigma$ yields the same answer to every $\mathfrak{B}$-formula as the logic programs LP(Prob($\mathcal{P}$)) $\cup\, \xi$ for every $\xi \in \mathcal{E}(\sigma)$ and that $\pi(\mathcal{E}(\sigma)) = \pi(\sigma)$. Further, the distribution semantics $\pi_{\mathcal{P}}^{dist}$ of $\mathcal{P}$ and the distribution semantics $\pi_{\text{Prob}(\mathcal{P})}^{dist}$ of Prob($\mathcal{P}$) yield the same joint distribution on $\mathcal{P}$.

Conversely, we can turn each ProbLog program to an equivalent LPAD-Program. Again, Riguzzi (2020), §2.4 establishes for a ProbLog program $\mathcal{P}$ the LPAD-transformation LPAD($\mathcal{P}$) is the LPAD-program that consists of one clause of the form $u(RF) : \pi(RF) \leftarrow$ for every random fact $\pi(RF) :: u(RF)$ of $\mathcal{P}$ and a clause of the form $\text{head}(LC) : 1 \leftarrow \text{body}(LC)$ for every logic clause $LC \in \text{LP}(\mathcal{P})$. In this case, every selection σ of LPAD($\mathcal{P}$) of probability not zero corresponds to a unique possible world $\mathcal{E}(\sigma)$, in which $u(RC)$ is true if and only if $\sigma(RC) \neq \bot$.

Again, we obtain that the LPAD-transformation respects the distribution semantics.

Theorem Appendix C.2 (*Riguzzi (2020), §2.4*)
By the transformation of $\mathcal{P}$ to the LPAD program LPAD($\mathcal{P}$), we have LP($\mathcal{P}$) $\cup\, \mathcal{E}(\sigma)$ and LPAD($\mathcal{P}$)$^\sigma$ yield the same answer to every $\mathcal{P}$-formula. We also get that $\pi(\sigma) = \pi(\mathcal{E}(\sigma))$. Hence, $\mathcal{P}$ and LPAD($\mathcal{P}$) yield the same probability for every $\mathcal{P}$-formula.

Up until now, we have not differed from Kiesel et al. (2023) proof for showing the equivalence of interventions and counterfactuals in CP-Logic with SCMs by the *do*-operator in Twin Networks. However, to demonstrate that the equivalence holds under the fix-operator, we must extend the above theorems with the lemma which follows (and is proved) closely in style to Kiesel et al. (2023) Appendix B.

Lemma Appendix C.3
Choose a proposition $X \in \mathcal{P}$ together with a truth value x.

1. In the situation of Theorem Appendix C.1, for every possible world $\mathcal{E} \in \mathcal{E}(\sigma)$ the logic programs $\mathcal{P}^\sigma_{fix(X:=x)}$ and LP(Prob($\mathcal{P}$)$_{fix(X:=x)}$) $\cup\, \mathcal{E}$ yield the same answer to every $\mathfrak{P}$-formula.
2. In the situation of Theorem Appendix C.2, for every selection σ of LPAD($\mathcal{P}$), the logic programs LPAD($\mathcal{P}$)$^\sigma_{fix(X:=x)}$ and LP(Prob($\mathcal{P}$)$_{fix(X:=x)}$) $\cup\, \mathcal{E}(\sigma)$ yield the same answer to every $\mathfrak{P}$-formula.

Proof
To prove (1), consider by Theorem B.1, for every $\mathfrak{P}$-formula ϕ, $\mathcal{P}^\sigma$ and $\mathrm{LP}(\mathrm{Prob}(\mathcal{P})) \cup \mathcal{E}$ will give the same answer to the query because in both programs are modular and consequently, their behavior is invariant to erasing clauses with X in the head or adding $X \leftarrow$. More precisely, the SWIG style $fix(X := x)$-operation in a LPAD/ProbLog syntactic setting is implemented by removing every head-option $h : \pi$ from every LPAD-clause whose head contains X, and adds some deterministic fact $X \leftarrow 0/1$.

This implements the same surgical operations as used in implementing the *do*-operator in earlier proofs. If we fix a selection σ of $\mathbf{P}$ and let $\mathcal{E} \in \mathcal{E}(\sigma)$ be an arbitrary possible world corresponding to σ, then the translation $\mathrm{Prob}(\mathbf{P})$ introduces for each annotated disjunction in $\mathbf{P}$ a sequence of auxiliary atoms and random facts whose semantic effect is to pick exactly one head option (or none) according to the annotated probabilities. A selection σ corresponds to fixing which auxiliary random facts are true in a possible world $\mathcal{E}$ of $\mathrm{Prob}(\mathbf{P})$; conversely $\mathcal{E}$ determines σ.

Performing the syntactic removals and additions that implement $\mathrm{fix}(X := x)$ on $\mathbf{P}^\sigma$ is equivalent to performing the corresponding removals and additions on the $\mathrm{Prob}(\mathbf{P})$ translation and then conjoining the choices represented by $\mathcal{E}$. Intuitively, the removals delete the same head-options in both syntaxes, and the deterministic additions become deterministic facts in the ProbLog encoding (or deterministic clauses whose head is an ordinary atom).

Because logic programs are modular, if two programs differ only in clauses for a restricted set of atoms (here, the atoms in X and the auxiliary atoms that were introduced to encode choices for those rules), then the truth of a $\mathbf{P}$-formula that mentions only other atoms remains unaffected. In particular, when we compare $\mathbf{P}^\sigma_{\mathrm{fix}(X:=x)}$ to $\mathrm{LP}(\mathrm{Prob}(\mathbf{P})_{\mathrm{fix}(X:=x)}) \cup \mathcal{E}$, all choices about auxiliary atoms (those in $\mathcal{E}$) are already fixed by $\mathcal{E}$, and the forced/blocked heads for atoms in X are syntactically identical in the two programs. It follows that the two programs produce the same derivations for all atoms and thus agree on the truth of every $\mathfrak{P}$-formula.

For (2), we analogously apply Theorem B.2 but start with a selection σ of $\mathrm{LPAD}(\mathcal{P})$. The construction of $\mathrm{LPAD}(\mathbf{P})^\sigma$ and the effect of $\mathrm{fix}(X := x)$ on it correspond, under the ProbLog translation, to $\mathrm{LP}(\mathrm{Prob}(\mathbf{P})_{\mathrm{fix}(X:=x)})$ conjoined with the possible world $\mathcal{E}(\sigma)$ that encodes the selection σ. The same modularity and syntactic-translation observations as above show the two programs are extensionally identical (on atoms of interest) and therefore produce the same answers to every $\mathfrak{P}$-formula. □ □

From here, our proofs for Theorem 4 and 5 follows Kiesel et al. (2023) again. CP-logic estalishes a causal semantics for LPAD-programs. The semantics focus on $\mathfrak{P}$-processes and tying them to the logic of LPAD. More precisely, a $\mathfrak{P}$-process $\mathcal{T}$ is a tuple $(T, \mathcal{I})$, where T is a directed tree and each edge is a labelled with a probability that describes transitioning from each node which are Humean events and. For each non-leaf node, the outgoing edges probabilities must sum to one. $\mathcal{I}$ is a map that assigns each node n in T a Herbrand Interpretation $\mathcal{I}(n)$ in $\mathfrak{B}$.

Furthermore, for each n in T, we associate the probability $\pi^T(n)$ which is given by the product of the probabilities of all edges along the walk from root $\bot$ of T to n. This

produces a distribution π^T on the Herbrand interpretations of I of $\mathfrak{B}$ by

$$\pi^{\mathcal{T}}(I) := \sum_{l \text{ leaf of } T, \mathcal{I}(l)=I} \pi^{\mathcal{T}}(l).$$

Vennekens et al. (2009) connects LPAD-programs to $\mathfrak{P}$-processes by fixing a LPAD-program $\mathcal{P}$ and defining a hypothetical derivation sequence of n in $\mathcal{T}$ as a sequence of three valued interpretations $(\nu_i)_{0 \leq i \leq n}$ where ν_0 assigns False to all atoms not in $\mathcal{I}(n)$ and for $i > 0$, there exists $RC \in \mathbf{P}$ and $j \in [1, l]$ with $\text{body}(RC)^{\nu_i} \neq False$, with $h_j^{i+1} = \text{Undefined}$, and with $\nu_i(p) = \nu_{i+1}(p)$ for all other proposition $p \in \mathcal{P}$.

Such a sequence is terminal if it cannot be extended and each terminal hypothetical derivation sequence n has the same limit ν_n, which is known as the potential in n.

For $RC \in \mathcal{P}$, we say that RC fires in a node n of T if for each $1 < i < l(RC)$ there exists a child n_i of n such that $\mathcal{I}(n_i) = \mathcal{I}(n) \cup \{h_i(RC)\}$ and such that each edge (n, n_i) is labeled with $\pi_i(RC)$. Moreover, there exists a child $n_{l(RC)+1}$ of n with $\mathcal{I}(n_{l(RC)+1}) = \mathcal{I}(n)$.

Let $\mathcal{R}_{\mathcal{E}}(n)$ denotes the set of all rules $RC \in \mathcal{P}$, for which there exists no ancestor a of n with $\mathcal{E}(a) = RC$. Then, $\mathcal{T}$ then may be an execution model of $\mathcal{P}$, $\mathcal{T} \models \mathcal{P}$, if there exists a mapping $\mathcal{E}$ from the non-leaf nodes of T to P such that:

1. $\mathcal{I}(\bot) = \emptyset$ for the root $\bot$ of T.
2. In each non-leaf node n a LPAD-clause $\mathcal{E}(n) \in \mathcal{R}_{\mathcal{E}}(n)$ fires with $\mathcal{I}(n) \models \text{body}(\mathcal{E}(n))$.
3. For each leaf l of T there exists no LPAD-clauses $RC \in \mathcal{R}_{\mathcal{E}}(l)$ with $\mathcal{I}(l) \models \text{body}(RC)$.
4. For every node n of T we find $\text{body}(\mathcal{E}(n))^{\nu_n} \neq Undefined$, where ν_n is the potential in n.

If $\mathcal{T} \models \mathcal{P}$, then the probability distribution defined by $\pi_{\mathcal{P}}^{CP} := \pi^{\mathcal{T}}$ matches the distribution semantics $\pi_{\mathcal{P}}^{dist}$ and implies the following.

Lemma Appendix C.4 (*Vennekens et al. (2009), §A.2*)
Let l be a leaf node in an execution model $\mathcal{T}$ of the LPAD-program $\mathcal{P}$. In this case, there exists a unique path p from the root $\bot$ of T to l. Define the selection $\sigma(l)$ by setting $\sigma(l)(RC) := i \in \mathbb{N}$ if and only if there exists a node n_j along p with $\mathcal{E}(n_j) = RC$ and $\mathcal{I}(n_{j+1}) := \mathcal{I}(n_j) \cup \{h_i(RC)\}$. Otherwise, we set $\sigma(l)(RC) := \bot$. In this way, we obtain that $\mathcal{P}^{\sigma(l)} \models \mathcal{I}(l)$. On the other hand, we find for each selection σ of $\mathcal{P}$ a leaf l of T with $\sigma(l) = \sigma$.

To finally demonstrate the equivalence of this treatment of counterfactuals to CP-logic, consider the presentation of interventions and counterfactuals in CP-logic from Vennekens et al. (2010).

Algorithm 5 Treatment of Counterfactuals in CP-logic

Require: $\mathbf{X} := \mathbf{x}, \mathbf{E} = \mathbf{e} \subseteq \mathcal{P}$, $\mathfrak{P}$-formula ϕ.
Ensure: Counterfactual probability $\pi^{CP}_{\mathcal{P}}(\phi \mid E = e, do(X := x))$.
1: Choose an execution model $\mathcal{T}$ of $\mathcal{P}$.
2: **for** each leaf l of $\mathcal{T}$ **do**
3: Intervene in the logic program $\mathcal{P}^{\sigma(l)}$ according to $X := x$ to obtain $\mathcal{P}^{\sigma(l),do(X:=x)}$.
4: Define

$$\pi^l(\phi) := \begin{cases} 1, & \mathcal{I}(l) \models (E = e) \ \wedge \ \mathcal{P}^{\sigma(l),do(X:=x)} \models \phi \\ 0, & \text{else.} \end{cases}$$

5: **end for**
6: Return

$$\pi^{CP}_{\mathcal{P}}(\phi \mid E = e, do(X := x)) := \sum_{l \text{ leaf of } \mathcal{T}} \pi^l(\phi) \cdot \pi^{CP}_{\mathcal{P}}(\mathcal{I}(l) \mid E = e). \tag{B1}$$

As we have established in Lemma Appendix C.3, we can analogously posit that with the rewiring of clauses under the fix-operator the following algorithm:

Algorithm 6 Fixed Operator Counterfactuals in CP-logic

Require: $\mathbf{X} := \mathbf{x}, \mathbf{E} = \mathbf{e} \subseteq \mathcal{P}$, $\mathfrak{P}$-formula ϕ.
Ensure: Counterfactual probability $\pi^{CP}_{\mathcal{P}}(\phi \mid E = e, fix(X := x))$.
1: Choose an execution model $\mathcal{T}$ of $\mathcal{P}$.
2: **for** each leaf l of $\mathcal{T}$ **do**
3: Intervene in the logic program $\mathcal{P}^{\sigma(l)}$ according to $X := x$ to obtain $\mathcal{P}^{\sigma(l),fix(X:=x)}$.
4: Define

$$\pi^l(\phi) := \begin{cases} 1, & \mathcal{I}(l) \models (E = e) \ \wedge \ \mathcal{P}^{\sigma(l),fix(X:=x)} \models \phi \\ 0, & \text{else.} \end{cases}$$

5: **end for**
6: Return

$$\pi^{CP}_{\mathcal{P}}(\phi \mid E = e, fix(X := x)) := \sum_{l \text{ leaf of } \mathcal{T}} \pi^l(\phi) \cdot \pi^{CP}_{\mathcal{P}}(\mathcal{I}(l) \mid E = e). \tag{B2}$$

With these preparations we can now turn to the proof of the desired consistency results:

Proof of Theorem 4.4
By Theorem 6, Lemma 9 and Lemma 8 the right-hand side of (B2) for $\mathcal{P}$ is the sum of the conditional probabilities $\pi(\mathcal{E}|E = e)$ of all possible worlds $\mathcal{E}$ of Prob($\mathcal{P}$) such that

$$\mathcal{M}\big(\mathcal{E}, \mathrm{LP}(\mathrm{Prob}(\mathcal{P})^{do(X:=x)})\big) \models \phi \quad \text{and} \quad \mathcal{M}(\mathcal{E}, \mathrm{LP}(\mathrm{Prob}(\mathcal{P}))) \models (E = e).$$

These are exactly the possible worlds that make the query ϕ true after intervention while the observation $E = e$ is true before intervening. Hence, we can consult the proof of Theorem 3 to see that (B1) computes the same value as Algorithm 6. □

Proof of Theorem 4.5

By Theorem 7, Lemma 8 and Lemma 9 the right-hand side of (B2) for LPAD($\mathcal{P}$) is the sum of the conditional probabilities $\pi(\mathcal{E}|E = e)$ of all possible worlds $\mathcal{E}$ of $\mathcal{P}$ such that

$$\mathcal{M}\big(\mathcal{E}, \mathrm{LP}(\mathcal{P}^{do(X:=x)})\big) \models \phi \quad \text{and} \quad \mathcal{M}(\mathcal{E}, \mathrm{LP}(\mathcal{P})) \models (E = e).$$

These are exactly the possible worlds that make the query ϕ true after intervention while the observation $E = e$ is true before intervening. Hence, we can consult the proof of Theorem 3 to see that (B1) computes the same value as Algorithm 6. □

Appendix D Proof of Theorem 5.1

Proof

We summarize notation from Kiesel et al. (2023) [12]. First, we define two propositional alphabets $\mathfrak{P}^S$ to handle the evidence from the source world and $\mathfrak{P}^I$ to handle the interventions in the counterfactual. In particular, we set

$$e(\mathfrak{P}^S) \;=\; e(\mathfrak{P}) \quad \text{and} \quad i(\mathfrak{P}^I) \;=\; i(\mathfrak{P}),$$

and we require that

$$\mathfrak{P}^S \;\cap\; \mathfrak{P}^I \;=\; \emptyset$$

In this way, we obtain maps

$$e/i : \mathfrak{P} \;\to\; \mathfrak{P}^{\,S/I} \quad \text{by} \quad p \;\mapsto\; p^{\,S/I},$$

that easily generalize to literals, clauses, and programs.

Furthermore, we define the *counterfactual semantics* of $\mathcal{P}$ by

$$\mathcal{P}^K \;=\; \mathcal{P}^S \;\cup\; \mathcal{P}^I.$$

Next, we intervene in $\mathcal{P}^K$ for the counterfactual query is for the probability of a consequent ϕ^I in the interventional world given evidence $\mathbf{E}^S = \mathbf{e}$ in the source world, after applying an intervention $fix(\mathbf{X} := \mathbf{x})$ to the interventional part of the program. Let $P^{K,fix(\mathbf{X}:=\mathbf{x})}$ denote this modified program given by Algorithm 3. Finally, we obtain the desired probability $\pi_P^{\mathrm{SCM}}(\,\phi \mid \mathbf{E} = \mathbf{e},\; fix(\mathbf{X} := \mathbf{x})\,)$ by querying the program $P^{K,fix(\mathbf{X}:=\mathbf{x})}$ for the conditional probability $\pi(\phi \mid \mathbf{E} = \mathbf{e})$.

$$
\begin{aligned}
\pi_{\mathcal{P}}^{SCM}(\phi \mid \mathbf{E}=\mathbf{e},\, fix(\mathbf{X}=\mathbf{x})) &:= \pi_{\mathcal{P}^K, fix(\mathbf{X}:=\mathbf{x})}(\phi^I \mid \mathbf{E}^S=\mathbf{e}) \\
&= \frac{\pi_{\mathcal{P}^K, fix(\mathbf{X}:=\mathbf{x})}(\phi^I \wedge \mathbf{E}^S=\mathbf{e})}{\pi_{\mathcal{P}^K, fix(\mathbf{X}:=\mathbf{x})}(\mathbf{E}^S=\mathbf{e})} \\
&= \frac{\pi_{\mathcal{P}^K, fix(\mathbf{X}:=\mathbf{x})}(\phi^I \wedge \mathbf{E}^S=\mathbf{e})}{\pi_{\mathcal{P}^K}(\mathbf{E}^S=\mathbf{e})} \\
&= \frac{1}{\pi_{\mathcal{P}^K}(\mathbf{E}^S=\mathbf{e})} \sum_{\varepsilon} \pi_{\mathcal{P}^K}(\varepsilon) \cdot [\varepsilon \models_{\mathcal{P}^K_{fix(\mathbf{X}:=\mathbf{x})}} (\phi^I \wedge \mathbf{E}^S=\mathbf{e})] \\
&= \frac{1}{\pi_{\mathcal{P}}(\mathbf{E}=\mathbf{e})} \sum_{\varepsilon} \pi_{\mathcal{P}}(\varepsilon) \cdot [\varepsilon \models_{\mathcal{P}^K} \mathbf{E}^S=\mathbf{e}] \cdot [\varepsilon \models_{\mathcal{P}^K_{fix(\mathbf{X}:=\mathbf{x})}} \phi^I] \\
&= \frac{\sum_{\varepsilon} \pi_{\mathcal{P}}(\varepsilon) \cdot [\varepsilon \models \mathbf{E}=\mathbf{e}] \cdot [\varepsilon \models_{\mathcal{M}_x} \phi]}{\sum_{\varepsilon} \pi_{\mathcal{P}}(\varepsilon) \cdot [\varepsilon \models \mathbf{E}=\mathbf{e}]} \\
&= \frac{P_{SCM}(\phi_{\mathbf{X}\leftarrow\mathbf{x}}, \mathbf{E}=\mathbf{e})}{P_{SCM}(\mathbf{E}=\mathbf{e})} \\
&= P_{SCM}(\phi \mid \mathbf{E}=\mathbf{e},\, fix(\mathbf{X}:=\mathbf{x}))
\end{aligned}
$$

□