Explaining Explanations in Probabilistic Logic Programming*

Germán Vidal[0000−0002−1857−6951]

VRAIN, Universitat Politècnica de València, Spain
gvidal@dsic.upv.es

Abstract. The emergence of tools based on artificial intelligence has also led to the need of producing explanations which are understandable by a human being. In most approaches, the system is considered a *black box*, making it difficult to generate appropriate explanations. In this work, though, we consider a setting where models are *transparent*: probabilistic logic programming (PLP), a paradigm that combines logic programming for knowledge representation and probability to model uncertainty. However, given a query, the usual notion of *explanation* is associated with a set of choices, one for each random variable of the model. Unfortunately, such a set does not explain *why* the query is true and, in fact, it may contain choices that are actually irrelevant for the considered query. To improve this situation, we present in this paper an approach to explaining explanations which is based on defining a new query-driven inference mechanism for PLP where proofs are labeled with *choice expressions*, a compact and easy to manipulate representation for sets of choices. The combination of proof trees and choice expressions allows us to produce comprehensible query justifications with a causal structure.

(*) This version of the contribution has been accepted for publication at APLAS 2024, after peer review, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record of this contribution is published in Programming Languages and Systems (Proceedings of APLAS 2024), Springer, 2024, and is available online at https://doi.org/10.1007/978-981-97-8943-6_7. *Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use* https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

1 Introduction

Explainable AI (XAI) [5] is an active area of research that includes many different approaches. Explainability is especially important in the context of decision

* This work has been partially supported by grant PID2019-104735RB-C41 funded by MICIU/AEI/ 10.13039/501100011033, by the *Generalitat Valenciana* under grant CIPROM/2022/6 (FassLow), and by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215.

support systems, where the user often demands to know the reasons for a decision. Furthermore, the last regulation on data protection in the European Union [12] has introduced a "right to explanation" for algorithmic decisions.

The last decades have witnessed the emergence of a good number of proposals to combine logic programming and probability, e.g., Logic Programs with Annotated Disjunctions (LPADs) [40], CP-logic [39], ProbLog [27], Probabilistic Horn Abduction [24], Independent Choice Logic [25], PRISM [37], Stochastic Logic Programs [20], and Bayesian Logic Programs [16], to name a few (see, e.g., the survey [34] and references therein). Most of these approaches are based on the so-called *distribution* semantics introduced by Sato [36]. In this work, we mainly follow the LPAD [40] approach to probabilistic logic programming (PLP), which has an expressive power similar to, for example, Bayesian networks [32]. For instance, the following probabilistic clause:

$$heads(X){:}0.5;\ tails(X){:}0.5 \leftarrow toss(X), \neg biased(X).$$

specifies that every time a coin X which is not biased is tossed, it lands on heads with probability 0.5 and on tails with probability 0.5. Note that only one choice can be true for a given X (thus ";" should not be interpreted as logical disjunction). One can say that each instance of the clause above represents a *random variable* with as many values as head disjuncts (two, in this case).

Given a program, a *selection* is basically a choice of values for *all* the random variables represented in a probabilistic logic program. Every selection induces a possible *world*, a normal logic program which is obtained by choosing the head determined by the selection in each grounding of each probabilistic clause (and removing its probability). For example, given an instance of the clause above for $X = coin1$, a selection that chooses $heads(coin1)$ will include the normal clause

$$heads(coin1) \leftarrow toss(coin1), \neg biased(coin1).$$

In this context, an *explanation* often refers to a selection or, equivalently, to the world induced from it. For instance, the MPE task [38], which stands for *Most Probable Explanation*, basically consists in finding the world with the highest probability given a query (typically denoting a set of observed *evidences*).

A world can be seen indeed as an interpretable model in which a given query holds. However, this kind of explanations also presents several drawbacks. First, a world might include clauses which are irrelevant for the query, thus adding noise to the explanation. Second, the chain of inferences that proved the query is far from obvious from the explanation (i.e., from the given world). In fact, there can be several different chains of inferences that explain why the query is true, each with an associated probability. Finally, a world (a set of clauses) might be too technical an explanation for non-experts.

Alternatively, some work considers that an explanation for a query is given by a set of choices: those that are *necessary* to prove the query (see, e.g., [30]). Although in this case there is no irrelevant information, it still does not have a causal structure. Furthermore, a set of choices does not provide an intuitive

explanation about why the query holds. In order to *explain* explanations, we propose in this work a combination of proof trees—that show the chain of inferences used to prove a query—and a new representation for choices that gives rise to a more compact notation. For this purpose, we make the following contributions:

- First, we introduce an algebra of *choice expressions*, a new notation for representing sets of choices that can be easily manipulated using standard rules like distributivity, double negation elimination, De Morgan's laws, etc.
- Then, we present SLPDNF-resolution, a query-driven top-down inference mechanism that extends SLDNF-resolution to deal with LPADs. We prove its soundness and completeness regarding the computation of explanations.
- Finally, we show how the proofs of SLPDNF-resolution can be used to produce comprehensible representations of the explanations of a query, which might help the user to understand why this query is indeed true.

We note that a practical evaluation of the proposed techniques will require the design and implementation of a software tool for generating explanations, which is left as future work.

Proofs of technical results can be found in the appendix.

2 Some Concepts of Logic Programming and PLP

In this section, we introduce some basic notions of logic programming [1,18] and probabilistic logic programming [34].

2.1 Logic Programming

We consider a *function-free* first-order language with a fixed vocabulary of predicate symbols, constants, and variables denoted by Π, $\mathcal{C}$ and $\mathcal{V}$, respectively. An *atom* has the form $f(t_1, \ldots, t_n)$ with $f/n \in \Pi$ and $t_i \in (\mathcal{C} \cup \mathcal{V})$ for $i = 1, \ldots, n$. A *literal* l is an atom a or its negation $\neg a$. A *query* Q is a conjunction of literals,[1] where the empty query is denoted by $\Box$. We use capital letters to denote (possibly atomic) queries. A *clause* has the form $h \leftarrow B$, where h (the *head*) is a positive literal (an atom) and B (the *body*) is a query; when the body is empty, the clause is called a *fact* and denoted just by h; otherwise, it is called a *rule*. A (normal) logic *program* P is a finite set of clauses.

We let $\mathsf{var}(s)$ denote the set of variables in the syntactic object s, where s can be a literal, a query or a clause. A syntactic object s is *ground* if $\mathsf{var}(s) = \emptyset$. Substitutions and their operations are defined as usual, where $\mathcal{D}om(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ is called the *domain* of a substitution σ. We let id denote the empty substitution. The application of a substitution θ to a syntactic object s is usually denoted by juxtaposition, i.e., we write $s\theta$ rather than $\theta(s)$. A syntactic object s_1 is *more general* than a syntactic object s_2, denoted $s_1 \leqslant s_2$, if there exists a substitution θ such that $s_1\theta = s_2$. A *variable renaming* is a substitution

[1] As is common in logic programming, we write a query $l_1 \wedge l_2 \wedge \ldots \wedge l_n$ as $l_1, l_2, \ldots, l_n$.

that is a bijection on $\mathcal{V}$. A substitution θ is a *unifier* of two syntactic objects s_1 and s_2 iff $s_1\theta = s_2\theta$; furthermore, θ is the *most general unifier* of s_1 and s_2, denoted by $\mathsf{mgu}(s_1, s_2)$ if, for every other unifier σ of s_1 and s_2, we have that $\theta \leqslant \sigma$,[2] i.e., there exists a substitution γ such that $\theta\gamma = \sigma$ when the domains are restricted to the variables of $\mathsf{var}(s_1) \cup \mathsf{var}(s_2)$.

In this work, we consider *negation as failure* [11] and SLDNF-resolution [3]. We say that a query $Q = l_1, \ldots, l_n$ *resolves* to Q' via σ with respect to literal l_i and clause c, in symbols $Q \leadsto_\sigma Q'$, if either i) $h \leftarrow B$ is a renamed apart variant of c, $\sigma = \mathsf{mgu}(l_i, h)$, and $Q' = (l_1, \ldots, l_{i-1}, B, l_{i+1}, \ldots, l_n)\sigma$, or ii) l_i is a negative literal, $\sigma = id$, and $Q' = l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n$. A (finite or infinite) sequence of resolution steps of the form $Q_0 \leadsto_{\sigma_1} Q_1 \leadsto_{\sigma_2} \ldots$ is called a *pseudoderivation*. As we will see below, an SLDNF-derivation is a pseudoderivation where the deletion of negative (ground) literals is justified by a finitely failed SLDNF-tree.

An SLDNF-tree Γ is given by a triple $(\mathcal{T}, T, \mathsf{subs})$, where $\mathcal{T}$ is a set of trees, $T \in \mathcal{T}$ is called the main tree, and subs is a function assigning to some nodes of trees in $\mathcal{T}$ a (subsidiary) tree from $\mathcal{T}$. Intuitively speaking, an SLDNF-tree is a directed graph with two types of edges, the usual ones (associated to resolution steps) and the ones connecting a node with the root of a subsidiary tree. A node can be marked with *failed*, *sucess*, and *floundered*. A tree is *successful* if it contains at least a leaf marked as success, and *finitely failed* if it is finite and all leaves are marked as *failed*.

Given a query Q_0, an SLDNF-tree for Q_0 starts with a single node labeled with Q_0. The tree can then be *extended* by selecting a query $Q = l_1, \ldots, l_n$ which is not yet marked (as failed, success or floundered) and a literal l_i and then proceeding as follows:

- If l_i is an atom, we add a child labeled with Q' for each resolution step $Q \leadsto_\sigma Q'$. The query is marked as *failed* if no such resolution steps exist.
- If l_i is a negative literal, $\neg a$, we have the following possibilities:
  - if a is nonground, the query Q is marked as *floundered*;
  - if $\mathsf{subs}(Q)$ is undefined, a new tree T' with a single node labeled with a is added to $\mathcal{T}$ and $\mathsf{subs}(Q)$ is set to the root of T';
  - if $\mathsf{subs}(Q)$ is defined and the corresponding tree is successful, then Q is marked as *failed*.
  - if $\mathsf{subs}(Q)$ is defined and the corresponding tree is finitely failed, then we have $Q \leadsto_{id} Q'$, where Q' is obtained from Q by removing literal l_i.

Empty leaves are marked as success. The extension of an SLDNF-tree continues until all leaves of the trees in $\mathcal{T}$ are marked.[3] An SLDNF-tree for a query Q is an SLDNF-tree in which the root of the main tree is labeled with Q. An SLDNF-tree is called *successful* (resp. finitely failed) if the main tree is successful (resp. finitely failed). An SLDNF-derivation for a query Q is a branch in the main tree of an SLDNF-tree Γ for Q, together with the set of all trees in Γ whose roots can

[2] Here, we assume that $\mathcal{D}om(\theta) \subseteq \mathsf{var}(s_1) \cup \mathsf{var}(s_2)$ if $\mathsf{mgu}(s_1, s_2) = \theta$.

[3] In [3] only the limit of the sequence of trees is called an SLDNF-tree, while the previous ones are called pre-SLDNF-trees. We ignore this distinction here for simplicity.

be reached from the nodes of this branch. Given a successful SLDNF-derivation, $Q_0 \rightsquigarrow_{\sigma_1} Q_1 \rightsquigarrow_{\sigma_2} \ldots \rightsquigarrow_{\sigma_n} Q_n$, the composition $\sigma_1\sigma_2\ldots\sigma_n$ (restricted to the variables of Q_0) is called a *computed answer substitution* of Q_0.

A normal logic program is *range-restricted* if all the variables occurring in the head of a clause also occur in the positive literals of its body. For range-restricted programs, every successful SLDNF-derivation completely grounds the initial query [21].

2.2 Logic Programs with Annotated Disjunctions

We assume that $\Pi = \Pi_p \uplus \Pi_d$, the set of predicate symbols, is partitioned into a set Π_p of *probabilistic predicates* and a set Π_d of *derived predicates*, which are disjoint. An atom $f(t_1, \ldots, t_n)$ is called a *probabilistic atom* if $f \in \Pi_p$ and a *derived atom* if $f \in \Pi_d$. An LPAD $\mathcal{P} = \mathcal{P}_p \uplus \mathcal{P}_d$—or just *program* when no confusion can arise—consists of a set of probabilistic clauses $\mathcal{P}_p$ and a set of normal clauses $\mathcal{P}_d$ defining derived predicates. A *probabilistic clause* has the form $h_1 : p_1; \ldots; h_n : p_n \leftarrow B$, where $h_1, \ldots, h_n$ are probabilistic atoms, $p_1, \ldots, p_n$ are real numbers in the interval $[0, 1]$ (their respective probabilities) such that $\sum_{i=1}^{n} p_i \leqslant 1$, and B is a query. When $\sum_{i=1}^{n} p_i < 1$, we implicitly assume that a special atom none is added to the head of the clause, where $\mathsf{none}/0$ is a fresh predicate which does not occur in the original program, with associated probability $1 - \sum_{i=1}^{n} p_i$. Thus, in the following, we assume w.l.o.g. that $\sum_{i=1}^{n} p_i = 1$ for all clauses.

Example 1. Consider the following clause[4]

$$covid(X) : 0.4; flu(X) : 0.3 \leftarrow contact(X, Y), covid(Y).$$

It states that, if X is a contact of Y and Y has covid, then either X has covid too (probability 0.4) or X has the flu (probability 0.3). Moreover, X has neither covid nor the flu (i.e., none holds) with probability 0.3 $(1 - 0.4 - 0.3)$.

Now we consider the semantics of programs. Given a probabilistic clause $c = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B)$, each ground instance $c\theta$ represents a choice between n (normal) clauses: $(h_1 \leftarrow B)\theta$, $\ldots$ $(h_n \leftarrow B)\theta$. A particular choice is denoted by a triple (c, θ, i), $i \in \{1, \ldots, n\}$, which is called an *atomic choice*, and has as associated probability $\pi(c, i)$, i.e., p_i in the clause above. We say that a set κ of atomic choices is *consistent*, in symbols, $consistent(\kappa)$, if it does not contain two atomic choices for the same grounding of a probabilistic clause, i.e., it cannot contain (c, θ, i) and (c, θ, j) with $i \neq j$. A set of consistent atomic choices is called a *composite choice*. It is called a *selection* when the composite choice includes an atomic choice for each grounding of each probabilistic clause of the program. We let $\mathcal{S}_\mathcal{P}$ denote the set of all possible selections for a given program (which is finite since we consider function-free programs). Each selection $s \in \mathcal{S}_\mathcal{P}$ identifies

[4] Here and in the remaining examples we do not show the occurrences of none.

a *world* ω_s which contains a (ground) normal clause $(h_i \leftarrow B)\theta$ for each atomic choice $(c, \theta, i) \in s$, together with the clauses for derived predicates:

$$\omega_s = \{(h_i \leftarrow B)\theta \mid c = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B) \in \mathcal{P}_p \wedge (c, \theta, i) \in s\} \cup \mathcal{P}_d$$

We assume in this work that programs are *sound*, i.e., each world has a unique two-valued well-founded model [14] which coincides with its stable model [15], and SLDNF-resolution is sound and complete. We write $\omega_s \models Q$ to denote that the (ground) query Q is true in the unique model of the program. Soundness can be ensured, e.g., by requiring logic programs to be stratified [17], acyclic [2] or modularly acyclic [35]. These characterizations can be extended to LPADs in a natural way, e.g., an LPAD is stratified if each possible world is stratified.

Given a selection s, the probability of world ω_s is then defined as follows: $P(\omega_s) = P(s) = \prod_{(c,\theta,i)\in s} \pi(c, i)$. Given a program $\mathcal{P}$, we let $\mathcal{W}_\mathcal{P}$ denote the (finite) set of possible worlds, i.e., $\mathcal{W}_\mathcal{P} = \{\omega_s \mid s \in \mathcal{S}_\mathcal{P}\}$. Here, $P(\omega)$ defines a probability distribution over $\mathcal{W}_\mathcal{P}$. By definition, the sum of the probabilities of all possible worlds is equal to 1. The probability of a (ground) query Q in a program $\mathcal{P}$, called the *success probability* of Q in $\mathcal{P}$, in symbols $P(Q)$, is obtained by marginalization from the joint distribution $P(Q, \omega)$ as follows:

$$P(Q) = \sum_{\omega \in \mathcal{W}_\mathcal{P}} P(Q, \omega) = \sum_{\omega \in \mathcal{W}_\mathcal{P}} P(Q|\omega) \cdot P(\omega)$$

where $P(Q|\omega) = 1$ if $\omega \models Q$ and $P(Q|\omega) = 0$ otherwise. Intuitively speaking, the success probability of a query is the sum of the probabilities of all the worlds where this query is provable (equivalently, has a successful SLDNF-derivation).

3 Query-Driven Inference in PLP

In this section, we present our approach to query-driven inference for probabilistic logic programs. In order to ease the understanding, let us first consider the case of programs *without negation*. In this case, it suffices to redefine queries to also include an associated composite choice κ. Intuitively speaking, κ denotes a restriction on the worlds where the computation performed so far can be proved.

An initial query has thus the form $\langle Q, \emptyset\rangle$, where Q is a standard query and $\emptyset$ is an empty composite choice. Resolution steps with probabilistic clauses should update the current composite choice accordingly. For this operation to be well-defined, computations must be performed w.r.t. the grounding $\mathcal{G}(\mathcal{P})$ of program $\mathcal{P}$ (which is finite since the considered language is function-free).[5] Given a query $\langle Q, \kappa\rangle$, resolution is then defined as follows:[6]

– If the selected atom is a derived atom and $Q \rightsquigarrow_\sigma Q'$, then $\langle Q, \kappa\rangle \rightsquigarrow_\sigma \langle Q', \kappa\rangle$.

[5] In practice, given a query, it suffices to compute the *relevant* groundings of $\mathcal{P}_p$ for this query (see [13, Section 5.1] for a discussion on this topic).

[6] For simplicity, we use the same arrow $\rightsquigarrow$ for both standard resolution and its extended version for probabilistic logic programs.

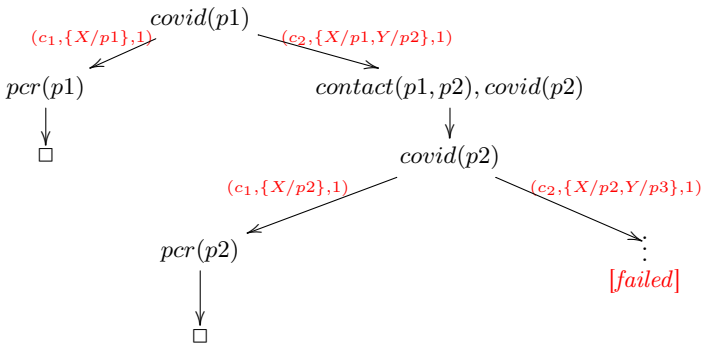

Fig. 1. Tree for the query $covid(p1)$

– If the selected atom is a probabilistic atom, a, then we have a resolution step $\langle Q, \kappa\rangle \rightsquigarrow_\sigma \langle Q', \kappa \cup \{(c, \theta, i)\}\rangle$ for each (ground) clause $c\theta = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B)\theta \in \mathcal{G}(\mathcal{P})$ such that $Q \rightsquigarrow_\sigma Q'$ is a resolution step with respect to atom a and clause $h_i\theta \leftarrow B\theta$ and, moreover, $\kappa \cup \{(c, \theta, i)\}$ is consistent.

Example 2. Consider the following program $\mathcal{P}$:

(c_1) $covid(X) : 0.9 \leftarrow pcr(X).$
(c_2) $covid(X) : 0.4; flu(X) : 0.3 \leftarrow contact(X, Y), covid(Y).$
$pcr(p1).\ \ pcr(p2).\ \ contact(p1, p2).\ \ person(p1).\ \ person(p2).\ \ person(p3).$

Here, the grounding $\mathcal{G}(\mathcal{P})$ will contain an instance of c_1 and c_2 for each person $p1$, $p2$, and $p3$. Figure 1 shows the resolution tree for the query $covid(p1)$. For simplicity, we only considered two groundings for c_2: $\{X/p1, Y/p2\}$ and $\{X/p2, Y/p3\}$; moreover, we always select the leftmost literal in a query. In the tree, for clarity, we show ordinary queries as nodes and label the edges with the atomic choices computed in the step (if any).

Here, we have two successful derivations for $covid(p1)$ that compute the composite choices $\{(c_1, \{X/p1\}, 1)\}$ and $\{(c_2, \{X/p1, Y/p2\}, 1), (c_1, \{X/p2\}, 1)\}$, i.e., the union of the atomic choices labeling the steps of each root-to-leaf successful derivation. Their probabilities are $\pi(c_1, 1) = 0.9$ and $\pi(c_2, 1) \times \pi(c_1, 1) = 0.4 * 0.9 = 0.36$, respectively. The computation of the marginal probability of a query is not generally the sum of the probabilities of its proofs since the associated worlds may overlap (as in this case). Computing the probability of a query is an orthogonal issue which is out of the scope of this paper (in this case, the marginal probability is 0.936); see Section 4 for further details on this topic.

3.1 Introducing Negation

Now, we consider the general case. In the following, we say that a selection s *extends* a composite choice κ if $\kappa \subseteq s$. Moreover, a composite choice κ identifies the set of worlds ω_κ that can be obtained by extending κ to a selection in all

possible ways. Formally, $\omega_\kappa = \{\omega_s \mid s \in \mathcal{S}_\mathcal{P} \ \wedge \kappa \subseteq s\}$. Given a set of composite choices K, we let $\omega_K = \cup_{\kappa \in K}\ \omega_\kappa$.

In principle, we could adapt Riguzzi's strategy in [30] for ICL (*Independent Choice Logic* [25]) to the case of LPAD. Basically, a resolution step for a query where a negated (ground) literal $\neg a$ is selected could proceed as follows:

- First, as in SLDNF, a tree for query a is built. Assume that the successful branches of this tree compute the composite choices $\kappa_1, \ldots, \kappa_n$, $n > 0$.
- Then, we know that a succeeds—equivalently, $\neg a$ fails—in all the worlds that extend the composite choices $\kappa_1, \ldots, \kappa_n$. Hence, we calculate a set of composite choices K that are *complementary* to those in $\kappa_1, \ldots, \kappa_n$.
- If K is not empty, the query where $\neg a$ was selected will have as many children as composite choices in K. For each child, the negated literal is removed from the query and the corresponding composite choice is added to the current one (assuming their union is consistent).

In order to formalize these ideas, we first recall the notion of *complement* [26]: If K is a set of composite choices, then a *complement* of K is a set K' of composite choices such that for all world $\omega \in \mathcal{W}_\mathcal{P}$, we have $\omega \in \omega_K$ iff $\omega \notin \omega_{K'}$. The notion of *dual* [26,30] is introduced to have an operational definition:

Definition 1 (dual). *If K is a set of composite choices, then composite choice κ' is a* dual *of K if for all $\kappa \in K$ there exist atomic choices $(c, \theta, i) \in \kappa$ and $(c, \theta, j) \in \kappa'$ such that $i \neq j$. A dual is* minimal *if no proper subset is also a dual. Let* $\mathsf{duals}(K)$ *be the set of minimal duals of K.*

The set of duals is indeed a complement of a set of composite choices (cf. Lemma 4.8 in [26]). The computation of $\mathsf{duals}(K)$ can be carried out using the notion of *hitting set* [28]. Let C be a collection of sets. Then, set H is a *hitting set* for C if $H \subseteq \bigcup_{S \in C} S$ and $H \cap S \neq \emptyset$ for each $S \in C$. In particular, we only consider hitting sets where exactly one element of each set $S \in C$ is selected Formally, $\mathsf{hits}(\{S_1, \ldots, S_n\}) = \{\{s_1, \ldots, s_n\} \mid s_1 \in S_1, \ldots, s_n \in S_n\}$.

In the following, given an atomic choice α, we let $\overline{\alpha}$ denote the relative complement (the standard notion from set theory) of $\{\alpha\}$ w.r.t. the domain of possible atomic choices for the same (ground) clause, i.e.,

$$\overline{(c, \theta, i)} = \{(c, \theta, j) \mid c = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B),\ i \neq j,\ j \in \{1, \ldots, n\}\}$$

We use $\alpha, \alpha', \ldots$ to denote atomic choices and $\beta, \beta', \ldots$ for either standard atomic choices or their complements. Furthermore, we let $\overline{K} = \{\overline{\kappa_1}, \ldots, \overline{\kappa_n}\}$ if $K = \{\kappa_1, \ldots, \kappa_n\}$, and $\overline{\kappa} = \overline{\alpha_1} \cup \ldots \cup \overline{\alpha_m}$ if $\kappa = \{\alpha_1, \ldots, \alpha_m\}$. The duals of a set of composite choices K can then be obtained from the hitting sets of $\overline{K}$:

Definition 2 (duals). *Let K be a set of composite choices. Then,* $\mathsf{duals}(K) = \mathsf{mins}(\mathsf{hits}(\overline{K}))$*, where function* mins *removes inconsistent and redundant composite choices, i.e.,* $\mathsf{mins}(K) = \{\kappa \in K \mid \mathit{consistent}(\kappa)\ \mathit{and}\ \kappa' \not\subset \kappa\ \mathit{for\ all}\ \kappa' \in K\}$.

It is easy to see that the above definition is more declarative but equivalent to similar functions in [26,30].

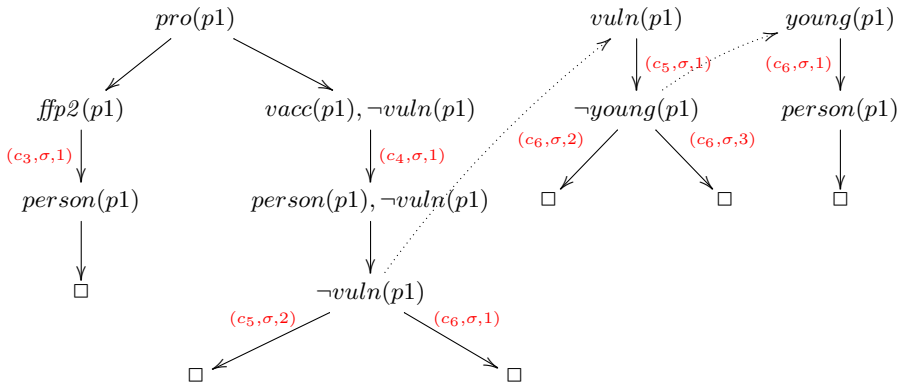

Fig. 2. Trees for the query $protected(p1)$, where predicates *protected*, *vaccinated*, and *vulnerable* are abbreviated as *pro*, *vacc*, and *vuln*, respectively.

Example 3. Consider the following LPAD program $\mathcal{P}$:

(c_1) $covid(X):0.9 \leftarrow pcr(X).$
(c_2) $covid(X):0.4; flu(X):0.3 \leftarrow contact(X,Y), covid(Y), \neg protected(X).$
(c_3) $ffp2(X):0.3; surgical:0.4; cloth:0.1 \leftarrow person(X).$
(c_4) $vaccinated(X):0.8 \leftarrow person(X).$
(c_5) $vulnerable(X):0.6 \leftarrow \neg young(X).$
(c_6) $young(X):0.2; adult(X):0.5 \leftarrow person(X).$

$protected(X) \leftarrow ffp2(X).$
$protected(X) \leftarrow vaccinated(X), \neg vulnerable(X).$
$pcr(p1).\ \ pcr(p2).\ \ contact(p1,p2).\ \ person(p1).\ \ person(p2).\ \ person(p3).$

The grounding $\mathcal{G}(\mathcal{P})$ will contain an instance of clauses $c_1, \ldots, c_6$ for each person $p1$, $p2$, and $p3$. As in the previous example, we show ordinary queries as nodes and label the edges with the new atomic choices of the step (if any). Figure 2 shows the trees for the query $protected(p1)$. Here, $young(p1)$ has only one successful derivation with composite choice $\{(c_6, \sigma, 1)\}$, where $\sigma = \{X/p1\}$. Let $K_1 = \{\{(c_6, \sigma, 1)\}\}$. In order to resolve $\neg young(p1)$, we compute the duals of K_1:

$$\begin{aligned}\mathsf{duals}(K_1) &= \mathsf{mins}(\mathsf{hits}(\overline{K_1})) = \mathsf{mins}(\mathsf{hits}(\{\overline{\{(c_6,\sigma,1)\}}\})) \\ &= \mathsf{mins}(\mathsf{hits}(\{\{(c_6,\sigma,2),(c_6,\sigma,3)\}\})) = \{\{(c_6,\sigma,2)\},\{(c_6,\sigma,3)\}\}\end{aligned}$$

Hence, $\neg young(p1)$ has two children with atomic choices $(c_6, \sigma, 2)$ and $(c_6, \sigma, 3)$. Consider now $vulnerable(p1)$. The tree includes two successful branches that compute composite choices $\{(c_5, \sigma, 1), (c_6, \sigma, 2)\}$ and $\{(c_5, \sigma, 1), (c_6, \sigma, 3)\}$. Let K_2 be a set with these two composite choices. In order to resolve $\neg vulnerable(p1)$

in the tree for $protected(p1)$, we first need the duals of K_2:

$$\begin{aligned}
&\mathsf{duals}(K_2) = \mathsf{mins}(\mathsf{hits}(\overline{K_2}))\\
&= \mathsf{mins}(\mathsf{hits}(\{\overline{\{(c_5,\sigma,1),(c_6,\sigma,2)\}}, \overline{\{(c_5,\sigma,1),(c_6,\sigma,3)\}}\}))\\
&= \mathsf{mins}(\mathsf{hits}(\{\{(c_5,\sigma,2),(c_6,\sigma,1),(c_6,\sigma,3)\},\{(c_5,\sigma,2),(c_6,\sigma,1),(c_6,\sigma,2)\}\}))\\
&= \mathsf{mins}(\{\{(c_5,\sigma,2)\},\{(c_5,\sigma,2),(c_6,\sigma,1)\},\{(c_5,\sigma,2),(c_6,\sigma,2)\},\{(c_6,\sigma,1)\},\\
&\qquad \{(c_6,\sigma,1),(c_6,\sigma,2)\},\{(c_6,\sigma,3),(c_5,\sigma,2)\},\{(c_6,\sigma,3),(c_6,\sigma,1)\},\\
&\qquad \{(c_6,\sigma,3),(c_6,\sigma,2)\}\}) = \{\{(c_5,\sigma,2)\},\{(c_6,\sigma,1)\}\}
\end{aligned}$$

Note that function mins removes redundant composite choices (e.g., all strict supersets of $\{(c_5,\sigma,2)\}$ and $\{(c_6,\sigma,1)\}$) as well as inconsistent composite choices like $\{(c_6,\sigma,3),(c_6,\sigma,2)\}$. Hence, $\neg vulnerable(p1)$ has two children with associated atomic choices $(c_5,\sigma,2)$ and $(c_6,\sigma,1)$.

We do not show the details here but, given the computed composite choices $\{(c_3,\sigma,1)\}$, $\{(c_4,\sigma,1),(c_5,\sigma,2)\}$, and $\{(c_4,\sigma,1),(c_6,\sigma,1)\}$ for $protected(p1)$, a call of the form $\neg protected(p1)$ would have twelve children.

3.2 An Algebra of Choice Expressions

The main drawback of the previous approach is that it usually produces a large number of proofs (i.e., successful branches), most of them identical except for the computed composite choice. For instance, as mentioned in Example 3, a call to $\neg protected(p1)$ will have twelve children. Likewise, a query of the form $\neg protected(p1), \neg protected(p2)$ will end up with a total of 144 children, all of them with a copy of the same query.

Our focus in this work is explainability. Hence, we aim at producing proofs that are as simple and easy to understand and manipulate as possible. An obvious first step into this direction could consist in associating *a set of composite choices* to each query rather than a single composite choice. In this way, the resolution of a query with a negative literal would produce a single child with the composition of the current set of composite choices and those returned by function duals.

For example, the query $\neg vulnerable(p1)$ in the first tree of Figure 2 would now have the form $\langle \neg vulnerable(p1),\ \{\{(c_4,\sigma,1)\}\}\rangle$. Given the duals of K_2 computed above, $\mathsf{duals}(K_2) = \{\{(c_5,\sigma,2)\},\{(c_6,\sigma,1)\}\}$, the child of $\neg vulnerable(p1)$ would have the following associated set of composite choices:

$$\{\{(c_4,\sigma,1)\}\} \otimes \left\{ \begin{array}{l} \{(c_5,\sigma,2)\},\\ \{(c_6,\sigma,1)\} \end{array} \right\} = \left\{ \begin{array}{l} \{(c_4,\sigma,1),(c_5,\sigma,2)\},\\ \{(c_4,\sigma,1),(c_6,\sigma,1)\} \end{array} \right\}$$

where the operation "$\otimes$" is defined as follows:

$$K_1 \otimes K_2 = \mathsf{mins}(\{\kappa_1 \cup \kappa_2 \mid \kappa_1 \in K_1, \kappa_2 \in K_2\}) \tag{1}$$

However, this approach would only reduce the number of identical nodes in the tree, but the computed composite choices would be the same as before.

As an alternative, we introduce an algebra of *choice expressions*, a representation for sets of composite choices which enjoy several good properties: they are more compact and can be easily manipulated using well-known logical rules (distributive laws, double negation elimination, De Morgan's laws, etc).

Definition 3. *A choice expression is defined inductively as follows:*

- $\bot$ *and* $\top$ *are choice expressions;*
- *an atomic choice* α *is a choice expression;*
- *if* C, C' *are choice expressions then* $\neg\mathsf{C}$, $\mathsf{C} \wedge \mathsf{C}'$, *and* $\mathsf{C} \vee \mathsf{C}'$ *are choice expressions too, where* $\neg$ *has higher precedence than* $\wedge$, *and* $\wedge$ *higher than* $\vee$.

Given a program $\mathcal{P}$, *we let* $\mathbb{C}_\mathcal{P}$ *denote the associated domain of choice expressions that can be built using the atomic choices of* $\mathcal{P}$. *We will omit the subscript* $\mathcal{P}$ *in* $\mathbb{C}_\mathcal{P}$ *when the program is clear from the context or irrelevant.*

A choice expression essentially represents a set of composite choices. E.g., the expression $(\alpha_1 \wedge \alpha_2) \vee \alpha_3 \vee (\alpha_4 \wedge \alpha_5)$ represents the set $\{\{\alpha_1, \alpha_2\}, \{\alpha_3\}, \{\alpha_4, \alpha_5\}\}$. In particular, negation allows us to represent sets of composite choices in a more compact way. For example, $\neg(c_3, \sigma, 1) \wedge \neg(c_6, \sigma, 1)$ represents a set with 6 composite choices, i.e., all combinations of pairs from $\{(c_3, \sigma, 2), (c_3, \sigma, 3), (c_3, \sigma, 4)\}$ and $\{(c_6, \sigma, 2), (c_6, \sigma, 3)\}$. The following function γ formalizes this equivalence:

Definition 4. *Given a choice expression* $\mathsf{C} \in \mathbb{C}$, *we let* $\gamma(\mathsf{C})$ *denote the associated set of composite choices, where function* γ *is defined inductively as follows:*

- $\gamma(\bot) = \{\}$, *i.e.,* $\bot$ *denotes an inconsistent set of atomic choices.*
- $\gamma(\top) = \{\{\}\}$, *i.e.,* $\top$ *represents a composite choice,* $\{\}$, *that can be extended in order to produce all possible selections.*
- $\gamma(\alpha) = \{\{\alpha\}\}$.
- $\gamma(\neg\mathsf{C}) = \mathsf{duals}(\gamma(\mathsf{C}))$, *i.e.,* $\neg\mathsf{C}$ *represents a complement of* C.
- $\gamma(\mathsf{C}_1 \wedge \mathsf{C}_2) = \mathsf{mins}(\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2))$, *where "*$\otimes$*" is defined in (1) above.*
- $\gamma(\mathsf{C}_1 \vee \mathsf{C}_2) = \mathsf{mins}(\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_2))$.

In practice, a set of composite choices K is just a device to represent a set of worlds ω_K. Therefore, we will not distinguish two choice expressions, C_1 and C_2, as long as the worlds identified by $\gamma(\mathsf{C}_1)$ and $\gamma(\mathsf{C}_2)$ are the same. For example, the choice expressions α_1 and $(\alpha_1 \wedge \alpha_2) \vee (\alpha_1 \wedge \neg\alpha_2)$ are equivalent, since both represent the same set of selections (those including atomic choice α_1).

Formally, we introduce the following equivalence relation on choice expressions: $\mathsf{C}_1 \sim \mathsf{C}_2$ if $\omega_{\gamma(\mathsf{C}_1)} = \omega_{\gamma(\mathsf{C}_2)}$. Roughly speaking, C_1 and C_2 are equivalent if the sets of composite choices in $\gamma(\mathsf{C}_1)$ and $\gamma(\mathsf{C}_2)$ can be extended to produce the same set of selections.

Let $\widetilde{\mathbb{C}}$ denote the quotient set of $\mathbb{C}$ by "$\sim$". Moreover, we let $\mathsf{C} \in \widetilde{\mathbb{C}}$ denote the equivalence class $[\mathsf{C}]$ when no confusion can arise. Then, $\langle \widetilde{\mathbb{C}}, \wedge, \vee, \neg, \top, \bot \rangle$ is a Boolean algebra and the following axioms hold:

Associativity $\mathsf{C}_1 \vee (\mathsf{C}_2 \vee \mathsf{C}_3) = (\mathsf{C}_1 \vee \mathsf{C}_2) \vee \mathsf{C}_3$ and $\mathsf{C}_1 \wedge (\mathsf{C}_2 \wedge \mathsf{C}_3) = (\mathsf{C}_1 \wedge \mathsf{C}_2) \wedge \mathsf{C}_3$.
Commutativity $\mathsf{C}_1 \vee \mathsf{C}_2 = \mathsf{C}_2 \vee \mathsf{C}_1$ and $\mathsf{C}_1 \wedge \mathsf{C}_2 = \mathsf{C}_2 \wedge \mathsf{C}_1$.
Absorption $\mathsf{C}_1 \vee (\mathsf{C}_1 \wedge \mathsf{C}_2) = \mathsf{C}_1$ and $\mathsf{C}_1 \wedge (\mathsf{C}_1 \vee \mathsf{C}_2) = \mathsf{C}_1$.
Identity $\mathsf{C} \vee \bot = \mathsf{C}$ and $\mathsf{C} \wedge \top = \mathsf{C}$.
Distributivity $\mathsf{C}_1 \vee (\mathsf{C}_2 \wedge \mathsf{C}_3) = (\mathsf{C}_1 \vee \mathsf{C}_2) \wedge (\mathsf{C}_1 \vee \mathsf{C}_3)$ and $\mathsf{C}_1 \wedge (\mathsf{C}_2 \vee \mathsf{C}_3) = (\mathsf{C}_1 \wedge \mathsf{C}_2) \vee (\mathsf{C}_1 \wedge \mathsf{C}_3)$.

Complements $\mathsf{C} \vee \neg\mathsf{C} = \top$ and $\mathsf{C} \wedge \neg\mathsf{C} = \bot$.

Furthermore, double negation elimination and De Morgan's laws also hold:

Double negation elimination $\neg\neg\mathsf{C} = \mathsf{C}$.
De Morgan $\neg(\mathsf{C}_1 \vee \mathsf{C}_2) = \neg\mathsf{C}_1 \wedge \neg\mathsf{C}_2$ and $\neg(\mathsf{C}_1 \wedge \mathsf{C}_2) = \neg\mathsf{C}_1 \vee \neg\mathsf{C}_2$.

In the following, function "mins" is redefined in terms of the following rewrite rules which are applied modulo associativity and commutativity of $\wedge$ and $\vee$:

$$\begin{array}{llrr} \alpha_1 \wedge \alpha_2 \to \bot & \text{if } \alpha_1 \in \overline{\alpha_2} & \mathsf{C} \wedge \neg\mathsf{C} \to \bot & \mathsf{C} \vee \neg\mathsf{C} \to \top \\ \alpha_1 \wedge \neg\alpha_2 \to \alpha_1 & \text{if } \alpha_1 \in \overline{\alpha_2} & \mathsf{C} \wedge \top \to \mathsf{C} & \mathsf{C} \wedge \bot \to \bot \\ & & \mathsf{C} \vee \top \to \top & \mathsf{C} \vee \bot \to \mathsf{C} \\ & & \mathsf{C}_1 \vee (\mathsf{C}_1 \wedge \mathsf{C}_2) \to \mathsf{C}_1 & \mathsf{C} \wedge \mathsf{C} \to \mathsf{C} \end{array}$$

The first rule introduces an inconsistency when a conjunction includes two different atomic choices for the same clause $c\theta$. The second rule simplifies a conjunction since $\neg\alpha_2$ denotes any atomic choice in $\overline{\alpha_2}$ but $\alpha_1 \in \overline{\alpha_2}$ is more specific. The remaining rules are just oriented axioms or an obvious simplification.

It is often useful to compute the DNF (Disjunctive Normal Form) of a choice expression in order to have a canonical representation:

Definition 5. *Let* C *be a choice expression. Then,* $\mathsf{dnf}(\mathsf{C})$ *is defined as follows:* $\mathsf{dnf}(\mathsf{C}) = \mathsf{mins}(\mathsf{C}')$*, where* $\mathsf{C} \to^* \mathsf{C}' \not\to$ *and the relation* $\to$ *is defined by the following canonical term rewrite system:*

$$\begin{array}{lll} \neg\neg\mathsf{C} \to \mathsf{C} & \neg(\mathsf{C}_1 \vee \mathsf{C}_2) \to \neg\mathsf{C}_1 \wedge \neg\mathsf{C}_2 & \mathsf{C}_1 \wedge (\mathsf{C}_2 \vee \mathsf{C}_3) \to (\mathsf{C}_1 \wedge \mathsf{C}_2) \vee (\mathsf{C}_1 \wedge \mathsf{C}_3) \\ & \neg(\mathsf{C}_1 \wedge \mathsf{C}_2) \to \neg\mathsf{C}_1 \vee \neg\mathsf{C}_2 & (\mathsf{C}_1 \vee \mathsf{C}_2) \wedge \mathsf{C}_3 \to (\mathsf{C}_1 \wedge \mathsf{C}_3) \vee (\mathsf{C}_2 \wedge \mathsf{C}_3) \end{array}$$

In the following, if a tree for atom a has n successful derivations computing choice expressions $\mathsf{C}_1, \ldots, \mathsf{C}_n$, we let $\mathsf{dnf}(\neg(\mathsf{C}_1 \vee \ldots \vee \mathsf{C}_n))$ denote its duals.

3.3 SLPDNF-Resolution

Finally, we can formalize the construction of SLPDNF-trees.[7] In principle, they have the same structure of SLDNF-trees. The main difference is that nodes are now labeled with pairs $\langle Q, \mathsf{C}\rangle$ and that the edges are labeled with both an mgu (as before) and a choice expression (when a probabilistic atom is selected). Given a query Q_0, the construction of an SLPDNF-tree for Q_0 starts with a single node labeled with $\langle Q_0, \top\rangle$. An SLPDNF-tree can then be *extended* by selecting a query $\langle Q, \mathsf{C}\rangle$ with $Q = l_1, \ldots, l_n$ which is not yet marked (as failed, success or floundered) and a literal l_i of Q and then proceeding as follows:

– If l_i is a derived atom and $Q \rightsquigarrow_\sigma Q'$, then $\langle Q, \mathsf{C}\rangle \rightsquigarrow_\sigma \langle Q', \mathsf{C}\rangle$.

[7] SLPDNF stands for Selection rule driven Linear resolution for Probabilistic Definite clauses augmented by the Negation as Failure rule.

- If l_i is a probabilistic atom, then we have a resolution step $\langle Q, \mathsf{C}\rangle \rightsquigarrow_{\sigma,(c,\theta,i)} \langle Q', \mathsf{C}'\rangle$ for each clause $c\theta = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B)\theta \in \mathcal{G}(\mathcal{P})$ such that $Q \rightsquigarrow_\sigma Q'$ is a resolution step with respect to atom l_i and clause $h_i\theta \leftarrow B\theta$ and, moreover, $\mathsf{C}' = \mathsf{dnf}(\mathsf{C} \wedge (c, \theta, i)) \neq \bot$.
- If l_i is a negative literal $\neg a$ we have the following possibilities:
  - if a is nonground, the query $\langle Q, \mathsf{C}\rangle$ is marked as *floundered*;
  - if $\mathsf{subs}(Q)$ is undefined, a new tree T' with a single node labeled with $\langle a, \top\rangle$ is added to $\mathcal{T}$ and $\mathsf{subs}(Q)$ is set to the root of T';
  - if $\mathsf{subs}(Q)$ is defined, the corresponding tree cannot be further extended, and it has n leaves marked as success with associated choice expressions $\mathsf{C}_1, \ldots, \mathsf{C}_n$, $n \geq 0$, then we have $\langle Q, \mathsf{C}\rangle \rightsquigarrow_{id,\mathsf{C}_a} \langle Q', \mathsf{C}'\rangle$, where Q' is obtained from Q by removing literal l_i, $\mathsf{C}_a = \mathsf{dnf}(\neg(\mathsf{C}_1 \vee \ldots \vee \mathsf{C}_n))$, and $\mathsf{C}' = \mathsf{dnf}(\mathsf{C} \wedge \mathsf{C}_a) \neq \bot$. If $\mathsf{C}' = \bot$ or $n = 0$, the node is marked as failed.

Leaves with empty queries are marked as success. An SLPDNF-tree for a query Q is an SLPDNF-tree in which the root of the main tree is labeled with $\langle Q, \top\rangle$. An SLPDNF-tree is called *successful* (resp. finitely failed) if the main tree is successful (resp. finitely failed). An SLPDNF-derivation for a query Q is a branch in the main tree of an SLPDNF-tree $\mathit{\Gamma}$ for Q, together with the set of all trees in $\mathit{\Gamma}$ whose roots can be reached from the nodes of this branch.

Given a successful SLPDNF-derivation for Q—also called a *proof*—of the form $\langle Q, \top\rangle = \langle Q_0, \mathsf{C}_0\rangle \rightsquigarrow_{\sigma_1} \langle Q_1, \mathsf{C}_1\rangle \rightsquigarrow_{\sigma_2} \ldots \rightsquigarrow_{\sigma_n} \langle Q_n, \mathsf{C}_n\rangle = \langle \square, \mathsf{C}\rangle$, the composition $\sigma_1\sigma_2\ldots\sigma_n$ (restricted to $\mathsf{var}(Q)$) is called a *computed answer substitution* of Q and C represents the worlds where this derivation can be proved.

Example 4. Consider again the LPAD from Example 3 and the same grounding for $p1$, $p2$, and $p3$. As in Example 2, we only consider two groundings for c_2 for simplicity: $\{X/p1, Y/p2\}$ and $\{X/p2, Y/p3\}$. Figure 3 shows the SLPDNF-tree for the query $covid(p1)$. Here, we have two proofs for $covid(p1)$. The choice expression C_1 of the first proof is just $(c_1, \sigma, 1)$, where $\sigma = \{X/p1\}$. The choice expression C_2 of the second proof is given by

$$\begin{aligned}&(c_2, \{X/p1, Y/p2\}, 1) \wedge (c_1, \{X/p2\}, 1) \wedge \neg(c_3, \sigma, 1) \wedge \neg(c_4, \sigma, 1)\\ \vee\ &(c_2, \{X/p1, Y/p2\}, 1) \wedge (c_1, \{X/p2\}, 1) \wedge \neg(c_3, \sigma, 1) \wedge (c_5, \sigma, 1) \wedge \neg(c_6, \sigma, 1)\end{aligned}$$

Here, C_2 represents a total of 9 composite choices which are obtained by replacing negated atomic choices by their corresponding atomic choices (i.e., $\gamma(\mathsf{C}_2)$).

Traditionally, an explanation for a ground query Q is defined as a selection s such that Q is true in the world ω_s associated to this selection. As in [26], we consider in this work a more relaxed notion and say that a composite choice κ is an explanation for ground query Q if Q is true in *all* worlds associated to every selection that extends κ. Formally, κ is an *explanation* for Q if $\omega_s \models Q$ for all selection $s \supseteq \kappa$. We also say that a set of composite choices K is *covering* w.r.t. (ground) query Q if for all $\omega \in \mathcal{W}_\mathcal{P}$ such that $\omega \models Q$ we have $\omega \in \omega_K$ [26].

Finding the most likely explanation of a query attracted considerable interest in the probabilistic logic programming field (where it is also called *Viterbi proof*

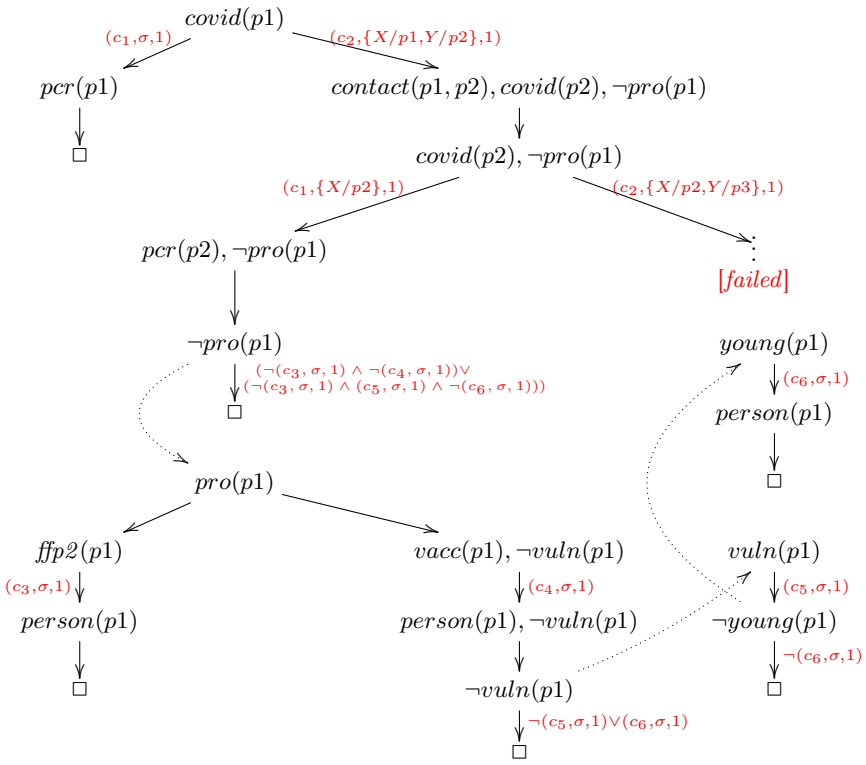

Fig. 3. SLPDNF-tree for query $covid(p1)$, where $\sigma = \{X/p1\}$ and predicates *protected*, *vaccinated*, and *vulnerable* are abbreviated as *pro*, *vacc*, and *vuln*, respectively.

[23]). Note that, although it may seem counterintuitive, the selection with the highest probability cannot always be obtained by extending the most likely proof of a query (see [38, Example 6]). Let $\mathsf{expl}_{\mathcal{P}}(Q)$ denote the set of composite choices represented by the choice expressions in the successful leaves of the SLPDNF-tree for Q w.r.t. $\mathcal{G}(\mathcal{P})$, i.e., $\mathsf{expl}_{\mathcal{P}}(Q) = \bigcup_{\mathsf{C} \in L} \gamma(\mathsf{C})$, where L is the set of leaves marked as success in the main SLPDNF-tree. The following result assumes that the query Q is ground, but could be extended to non-ground queries by considering each proof separately and backpropagating the computed answer substitution (which grounds the query since the program is range-restricted [21]).

Theorem 1. *Let $\mathcal{P}$ be a sound program and Q a ground query. Then, $\omega_s \models Q$ iff there exists a composite choice $\kappa \in \mathsf{expl}_{\mathcal{P}}(Q)$ such that $\kappa \subseteq s$.*

Therefore, $\mathsf{expl}(Q)$ is indeed a finite set of explanations which is covering for Q.

4 Proofs as Explanations

In this section, we focus on the representation of the explanations of a query. In principle, we propose to show the proofs of a query (its successful SLPDNF-derivations) as its explanations, from highest to lowest probability. Previous approaches only considered the probability of a standard derivation D computing a composite choice κ so that $P(D) = \prod_{(c,\theta,i)\in\kappa} \pi(c,i)$. Unfortunately, this is not applicable to SLPDNF-derivations computing a choice expression (equivalently, computing a set of composite choices). Therefore, we define the probability of an SLPDNF-derivation as follows:

Definition 6. *Let $\mathcal{P}$ be a program. Given a successful SLPDNF-derivation D for a query Q computing the choice expression C, its associated probability is $P(D) = \sum_{\omega\in\omega_{\gamma(\mathsf{C})}} P(\omega)$, i.e., the sum of the probabilities of all the worlds where the successful derivation can be proved.*

Computing $P(D)$ resembles the problem of computing the probability of a query: summing up the probabilities of the composite choices in $\gamma(\mathsf{C})$ for a computed choice expression C would not be correct since their associated worlds may overlap (i.e., there might be $\kappa, \kappa' \in \gamma(\mathsf{C})$ with $\kappa \neq \kappa'$ such that $\omega_\kappa \cap \omega_{\kappa'} \neq \emptyset$).

There is ample literature on computing the probability of a query, e.g., by combining inference and a conversion to some kind of Boolean formula [13]. We consider this problem an orthogonal topic which is outside of the scope of this paper. Nevertheless, we present a transformational approach that converts the problem of computing the probability of an SLPDNF-derivation into the problem of computing the probability of a query in an LPAD program.

Definition 7. *The first transformation takes an LPAD and returns a new LPAD:*

$$\mathsf{trp}(\mathcal{P}) = \{ch_1 : p_1; \ldots; ch_n : p_n \mid c\theta = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B)\theta \in \mathcal{G}(\mathcal{P}) \\ \text{and } ch_i = ch(c, \overline{\mathsf{var}}(c)\theta, i),\ i \in \{1, \ldots, n\}\}$$

where $\overline{\mathsf{var}}(c)$ returns a list *with the clause variables. Our second transformation takes a choice expression and returns a (ground) query as follows:*

$$\mathsf{trc}(\top) = \mathit{true} \qquad \mathsf{trc}(\bot) = \mathit{false}$$
$$\mathsf{trc}((c,\theta,i)) = ch(c, \overline{\mathsf{var}}(c)\theta, i) \qquad \mathsf{trc}(\neg\mathsf{C}) = \neg\mathsf{trc}(\mathsf{C})$$
$$\mathsf{trc}(\mathsf{C}_1 \wedge \mathsf{C}_2) = \mathsf{trc}(\mathsf{C}_1), \mathsf{trc}(\mathsf{C}_2) \qquad \mathsf{trc}(\mathsf{C}_1 \vee \mathsf{C}_2) = \mathsf{trc}(\mathsf{C}_1); \mathsf{trc}(\mathsf{C}_2)$$

Now, given an LPAD $\mathcal{P}$, the probability of an SLPDNF-derivation D computing choice expression C can be obtained from the probability of query $\mathsf{trc}(\mathsf{C})$ in LPAD $\mathsf{trp}(\mathcal{P})$. The correctness of the transformation is an easy consequence of the fact that the probability distribution of $\mathcal{P}$ and $\mathsf{trp}(\mathcal{P})$ is the same and that the query $\mathsf{trc}(\mathsf{C})$ computes an equivalent choice expression in $\mathsf{trp}(\mathcal{P})$; namely, an atomic choice (c,θ,i) has now the form $(c\theta,\{\},i)$ but the structure of the computed choice expression is the same.

Example 5. Consider again LPAD $\mathcal{P}$ from Example 3 and its grounding for $p1$, $p2$, and $p3$. $\mathsf{trp}(\mathcal{P})$ is as follows (for clarity, only the clauses of interest are shown):

$$ch(c_1,[p1],1):0.9. \quad ch(c_1,[p2],1):0.9.$$
$$ch(c_2,[p1,p2],1):0.4; ch(c_2,[p1,p2],2):0.3.$$
$$ch(c_3,[p1],1):0.3; ch(c_3,[p1],2):0.4; ch(c_3,[p1],3):0.1.$$
$$ch(c_4,[p1],1):0.8. \quad ch(c_5,[p1],1):0.6. \quad ch(c_6,[p1],1):0.2; ch(c_6,[p1],2):0.5.$$

The query $covid(p1)$ computes two choice expressions: $\mathsf{C}_1 = (c_1, \sigma, 1)$, where $\sigma = \{X/p1\}$, and $\mathsf{C}_2 =$

$$\begin{array}{l}(c_2,\{X/p1,Y/p2\},1) \wedge (c_1,\{X/p2\},1) \wedge \neg(c_3,\sigma,1) \wedge \neg(c_4,\sigma,1) \\ \vee\ (c_2,\{X/p1,Y/p2\},1) \wedge (c_1,\{X/p2\},1) \wedge \neg(c_3,\sigma,1) \wedge (c_5,\sigma,1) \wedge \neg(c_6,\sigma,1)\end{array} \quad (2)$$

Here, we have $\mathsf{trc}(\mathsf{C}_1) = ch(c_1,[p1],1)$ and $\mathsf{trc}(\mathsf{C}_2) =$

$$\begin{array}{l}ch(c_2,[p1,p2],1), ch(c_1,[p2],1), \neg ch(c_3,[p1],1), \neg ch(c_4,[p1],1) \\ ; ch(c_2,[p1,p2],1), ch(c_1,[p2],1), \neg ch(c_3,[p1],1), ch(c_5,[p1],1), \neg ch(c_6,[p1],1)\end{array}$$

The probability of $\mathsf{trc}(\mathsf{C}_1)$ in $\mathsf{trp}(\mathcal{P})$ using a system like PITA [33] or ProbLog [13] is 0.9, while that of $\mathsf{trc}(\mathsf{C}_2)$ is 0.147168. The probability of the query $covid(p1)$ can be obtained from the probability of the disjunction $\mathsf{trc}(\mathsf{C}_1); \mathsf{trc}(\mathsf{C}_2)$, which gives 0.9147168.

As mentioned before, we propose to show the proofs of a query as explanations, each one with its associated probability. However, instead of using SLPDNF-derivations, each proof will be represented by an AND-tree [7], whose structure is more intuitive. Roughly speaking, while an SLPDNF-derivation is a sequence of queries, in an AND-tree each node is labeled with a literal; when it is resolved with a (possibly probabilistic) clause with body literals $b_1, \ldots, b_n$, we add n nodes as children labeled with $b_1, \ldots, b_n$ (no children if $n = 0$, i.e., the clause is a fact). W.l.o.g., we assume in the following that the initial query is atomic.[8]

In order to represent the AND-trees of the proofs of a query (i.e., its successful SLPDNF-derivations), we follow these guidelines:

- First, we backpropagate the computed mgu's to all queries in the considered derivation, so that all of them become ground (a consequence of the program being range-restricted [21]). Formally, if D has the form $\langle Q_0, \mathsf{C}_0\rangle \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} \langle Q_n, \mathsf{C}_n\rangle$, we consider $\langle Q_0\sigma, \mathsf{C}_0\rangle \leadsto_{\sigma'_1} \ldots \leadsto_{\sigma'_n} \langle Q_n\sigma, \mathsf{C}_n\rangle$ instead, where $\sigma = \sigma_1\sigma_1 \ldots \sigma_n$.
- As for the choice expressions labeling the edges of the original derivation, we only show those associated to the resolution of negative literals. The case of (probabilistic) positive atoms is considered redundant since the information given by an atomic choice is somehow implicit in the tree.

[8] Nevertheless, one could consider an arbitrary query Q by adding a clause of the form $main(X_1, \ldots, X_n) \leftarrow Q$ for some fresh predicate $main/n$, where $\mathsf{var}(Q) = \{X_1, \ldots, X_n\}$, and then consider query $main(X_1, \ldots, X_n)$ instead.

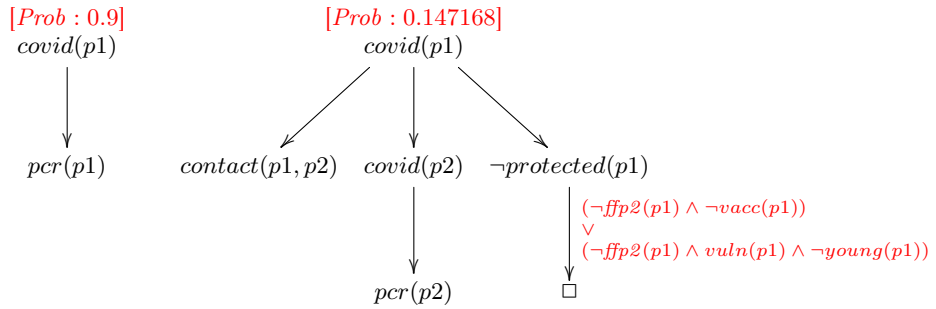

Fig. 4. Representing explanations with AND-trees

– Negative literals $\neg a$ have only one child, $\square$, and the edge is labeled with the same choice expression as in the SLPDNF-derivation, since this information cannot be extracted from the AND-tree. However, in order to improve its readability, we choose a more intuitive representation for choice expressions, which is inductively defined as follows:

$$\mathsf{chq}(\mathsf{C}) = \begin{cases} h_i\theta & \text{if } \mathsf{C} = (c, \theta, i),\ c = (h_1 : p_1; \ldots; h_n : p_n \leftarrow B) \\ \neg\mathsf{chq}(\mathsf{C}') & \text{if } \mathsf{C} = \neg\mathsf{C}' \\ \mathsf{chq}(\mathsf{C}_1) \wedge \mathsf{chq}(\mathsf{C}_2) & \text{if } \mathsf{C} = \mathsf{C}_1 \wedge \mathsf{C}_2 \\ \mathsf{chq}(\mathsf{C}_1) \vee \mathsf{chq}(\mathsf{C}_2) & \text{if } \mathsf{C} = \mathsf{C}_1 \vee \mathsf{C}_2 \end{cases}$$

E.g., given the following choice expression from Example 4:

$$\mathsf{C} = (\neg(c_3, \sigma, 1) \wedge \neg(c_4, \sigma, 1)) \vee (\neg(c_3, \sigma, 1) \wedge (c_5, \sigma, 1) \wedge \neg(c_6, \sigma, 1))$$

we have $\mathsf{chq}(\mathsf{C}) =$

$$(\neg \mathit{ffp2}(p1) \wedge \neg \mathit{vaccinated}(p1)) \vee (\neg \mathit{ffp2}(p1) \wedge \mathit{vulnerable}(p1) \wedge \neg \mathit{young}(p1))$$

Let $\mathsf{chq}(\mathsf{C})$ be the expression labeling the edge from $\neg a$. We further remove the occurrences of $\neg a$ in $\mathsf{chq}(\mathsf{C})$ (if any) since they are clearly redundant.

Consider, for instance, the two proofs of query $\mathit{covid}(p1)$ shown in Figure 3. The corresponding AND-trees are shown in Figure 4, each one with its associated probability (see Example 5 above), where the only edge issuing from a negative literal, $\neg \mathit{protected}(p1)$, is labeled with $\mathsf{chq}(\mathsf{C}) = (\neg \mathit{ffp2}(p1) \wedge \neg \mathit{vacc}(p1)) \vee (\neg \mathit{ffp2}(p1) \wedge \mathit{vuln}(p1) \wedge \neg \mathit{young}(p1))$ as computed above (here, *vacc* and *vuln* stands for *vaccinated* and *vulnerable*, as usual).

AND-trees can also be represented in textual form, as shown in Figure 5 (a), where *protected*, *vaccinated*, and *vulnerable* are abbreviated as *pro*, *vacc*, and *vuln*, respectively. Alternatively, one can easily rewrite the textual representation using natural language. For this purpose, the user should provide appropriate program annotations. For instance, given the following annotation:

```
%!read covid(A) as: "A has covid-19"
```

Explanations for query $covid(p1)$:

$[Prob : 0.9]$	$covid(p1)$	$p1$ has covid-19 because
	$pcr(p1)$	the pcr test of $p1$ was positive
$[Prob : 0.147168]$	$covid(p1)$	$p1$ has covid-19 because
	$contact(p1, p2)$	$p1$ had contact with $p2$
	$covid(p2)$	and $p2$ has covid-19 because
	$pcr(p2)$	the pcr test of $p2$ was positive
	$\neg pro(p1)$	and $p1$ was not protected because
	$\neg ffp2(p1)$	$p1$ didn't wear an ffp2 mask
	$\neg vacc(p1)$	and $p1$ was not vaccinated
	;	or because
	$\neg ffp2(p1)$	$p1$ didn't wear an ffp2 mask
	$vuln(p1)$	and $p1$ is vulnerable
	$\neg young(p1)$	and $p1$ is not young
	(a)	(b)

Fig. 5. Textual representation for explanations

we could replace $covid(p1)$ by the sentence "p1 has covid-19". Given appropriate annotations, the AND-trees in Figure 4 could also be represented as shown in Figure 5 (b).

Furthermore, one can easily design an appropriate interface where initially all elements are *folded* and one can click on each fact in order to unfold the list of reasons. In this way, the user could more easily navigate through the explanation and focus on her particular interests. Another improvement could consist in showing the concrete alternatives to negative literals. For example, one can show $\{surgical(p1), cloth(p1), none\}$ when hovering over $\neg ffp2(p1)$.

5 Related Work

We found very few works in which a query-driven inference mechanism for some form of probabilistic logic programs with negation is formalized. Among them, the closest are the works of Riguzzi [29,31,30], although the aim is different to ours (efficiently computing the marginal probability of a query rather than producing comprehensible explanations). Specifically, [29] proposes an algorithm for performing inference with LPADs where a modification of SLDNF-resolution is used for computing explanations in combination with BDDs. Later on, [31] presents an algorithm for performing inference on non-modularly acyclic LPADs. For this purpose, SLGAD (SLG for Annotated Disjunctions) is introduced, an extension of SLG-resolution for LPAD. Here, the inference mechanism uses tabling to avoid redundant computations and to avoid infinite loops. A distinctive feature of this approach is that the SLGAD-tree computes a set of composite choices which are *mutually incompatible*. This is achieved by performing a sort of *linearization* in the computation of atomic choices, so that every time a choice

is done, a new branch where this choice is not selected is also added. This is appropriate for computing the marginal probability of a query but makes the (typically huge) trees much less useful from the point of view of explainability.

The closest work is [30], which presents an extension of SLDNF-resolution for ICL (*Independent Choice Logic* [25]). There are, however, some significant differences to our work. First, the considered language is different (ICL vs LPAD). Second, [30] aims at defining a technique to compute the marginal probability of a query while our work is concerned with the generation of comprehensible explanations. Finally, the shape of the resolution trees are different since we deal with sets of composite choices (represented by choice expressions) so that queries where a negated literal is selected have (at most) one child. Indeed, the introduction of an algebra of choice expressions, together with negated atomic choices for a more compact representation, are significant differences w.r.t. [30], and they are essential for producing appropriate explanations. We also note that [30] does not require grounding the program, although in return it imposes some very strong conditions in order to guarantee that every time a literal is selected, it is ground and, morever, the computed mgu completely grounds the considered clause in the resolution step.

A different approach to computing explanations is introduced in [41]. The aim of this work is similar to ours, but there are significant differences too. On the one hand, the language considered is ProbLog without negation nor annotated disjunctions, so it is a much simpler setting (it can be seen as a particular case of the language considered in this work). On the other hand, the generated explanations are *programs* (a set of ground probabilistic clauses), which are obtained through different unfolding transformations. In fact, [41] can be seen as a complementary approach to the one presented here.

Finally, let us mention several approaches to improve the generation of explanations in some closely related but non-probabilistic fields: logic programming and *answer set programming* (ASP) [6]. First, [9] presents a tool, `xclingo`, for generating explanations from annotated ASP programs. Annotations are then used to construct derivation trees containing textual explanations. Moreover, the language allows the user to select *which* atoms or rules should be included in the explanations. On the other hand, [4] presents so-called *justifications* for ASP programs with constraints, now based on a goal-directed semantics. As in the previous work, the user can decide the level of detail required in a justification tree, as well as add annotations to produce justifications using natural language. Some of the ideas presented in Section 4 follow an approach which is similar to that of [4]. Other related approaches are the *off-line and on-line justifications* of [22], which provide a graph-based explanation of the truth value of a literal, and the *causal graph justifications* of [8], which explains why a literal is contained in an answer set (though negative literals are not represented). Obviously, our work shares the aim of these papers regarding the generation of comprehensible explanations in a logic setting. However, the considered language and the applied techniques are different. Nevertheless, we believe that our approach could be enriched with some of the ideas in these works.

6 Concluding Remarks and Future Work

In this work, we have presented a new approach for query-driven inference in a probabilistic logic language, thus defining an extension of the SLDNF-resolution principle, called SLPDNF-resolution, that keeps the structure of the original SLDNF-trees. Here, each proof of a query is now accompanied by a so-called *choice expression* that succinctly represents the possible worlds where this proof holds. We have also shown that choice expressions form a Boolean algebra, which allows us to manipulate them in a very flexible way. Furthermore, the generated proofs are especially appropriate to produce comprehensible explanations for a given query. In particular, we represent each proof in a way that its causal structure becomes evident, using either AND-trees or an equivalent textual representation using natural language.

As future work, we consider several extensions. On the one hand, we plan to deal with a broader class of programs. For this purpose, we will explore the definition of an extension of SLG-resolution [10] and/or some of the approaches for goal-directed execution of ASP programs (e.g., [19]). On the other hand, we would also like to extend the inference mechanism in order to include *evidences* (that is, ground facts whose true/false value is known). Finally, on the practical side, we plan to develop a robust tool for generating explanations that can be used with ProbLog and LPAD programs. Such a tool will allow us to evaluate in practice the usefulness of the techniques presented in this work.

Acknowledgements. I would like to thank the anonymous reviewers for their suggestions to improve this paper.

References

1. Apt, K.R.: From Logic Programming to Prolog. Prentice Hall (1997)
2. Apt, K.R., Bezem, M.: Acyclic programs. New Gener. Comput. **9**(3-4), 335–64 (1991). https://doi.org/10.1007/BF03037168
3. Apt, K.R., Doets, K.: A New Definition of SNDNF-Resolution. J. Log. Program. **18**(2), 177–190 (1994). https://doi.org/10.1016/0743-1066(94)90051-5
4. Arias, J., Carro, M., Chen, Z., Gupta, G.: Justifications for goal-directed constraint answer set programming. In: Ricca, F., Russo, A., Greco, S., Leone, N., Artikis, A., Friedrich, G., Fodor, P., Kimmig, A., Lisi, F.A., Maratea, M., Mileo, A., Riguzzi, F. (eds.) Proceedings of the 36th International Conference on Logic Programming (ICLP Technical Communications 2020). EPTCS, vol. 325, pp. 59–72 (2020). https://doi.org/10.4204/EPTCS.325.12
5. Arrieta, A.B., Rodríguez, N.D., Ser, J.D., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F.: Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. Inf. Fusion **58**, 82–115 (2020). https://doi.org/10.1016/j.inffus.2019.12.012
6. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011). https://doi.org/10.1145/2043174.2043195

7. Bruynooghe, M.: A Practical Framework for the Abstract Interpretation of Logic Programs. J. Log. Program. **10**(2), 91–124 (1991)
8. Cabalar, P., Fandinno, J., Fink, M.: Causal graph justifications of logic programs. Theory Pract. Log. Program. **14**(4-5), 603–618 (2014). https://doi.org/10.1017/S1471068414000234
9. Cabalar, P., Fandinno, J., Muñiz, B.: A system for explainable answer set programming. In: Ricca, F., Russo, A., Greco, S., Leone, N., Artikis, A., Friedrich, G., Fodor, P., Kimmig, A., Lisi, F.A., Maratea, M., Mileo, A., Riguzzi, F. (eds.) Proceedings of the 36th International Conference on Logic Programming (ICLP Technical Communications 2020). EPTCS, vol. 325, pp. 124–136 (2020). https://doi.org/10.4204/EPTCS.325.19
10. Chen, W., Warren, D.S.: Tabled evaluation with delaying for general logic programs. J. ACM **43**(1), 20–74 (1996). https://doi.org/10.1145/227595.227597
11. Clark, K.L.: Negation as Failure. In: Gallaire, H., Minker, J. (eds.) Proceedings of the Symposium on Logic and Data Bases. pp. 293–322. Advances in Data Base Theory, Plemum Press, New York (1977). https://doi.org/10.1007/978-1-4684-3384-5_11
12. EU, EEA: Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, available from https://eur-lex.europa.eu/eli/reg/2016/679/oj
13. Fierens, D., den Broeck, G.V., Renkens, J., Shterionov, D.S., Gutmann, B., Thon, I., Janssens, G., Raedt, L.D.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory Pract. Log. Program. **15**(3), 358–401 (2015). https://doi.org/10.1017/S1471068414000076
14. Gelder, A.V., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM **38**(3), 620–650 (1991). https://doi.org/10.1145/116825.116838
15. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Proceedings of the 5th International Conference on Logic Programming (ICLP'88). pp. 1070–1080. MIT Press (1988)
16. Kersting, K., Raedt, L.D.: Towards combining inductive logic programming with bayesian networks. In: Rouveirol, C., Sebag, M. (eds.) Proceedings of the 11th International Conference on Inductive Logic Programming (ILP 2001). Lecture Notes in Computer Science, vol. 2157, pp. 118–131. Springer (2001). https://doi.org/10.1007/3-540-44797-0_10
17. Lifschitz, V.: On the declarative semantics of logic programs with negation. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 177–192. Morgan Kaufmann (1988). https://doi.org/10.1016/B978-0-934613-40-8.50008-7
18. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987). https://doi.org/10.1007/978-3-642-83189-8
19. Marple, K., Bansal, A., Min, R., Gupta, G.: Goal-directed execution of answer set programs. In: Schreye, D.D., Janssens, G., King, A. (eds.) Principles and Practice of Declarative Programming (PPDP'12). pp. 35–44. ACM (2012). https://doi.org/10.1145/2370776.2370782
20. Muggleton, S.: Stochastic logic programs. In: de Raedt, L. (ed.) Advances in Inductive Logic Programming, pp. 254–264. IOS Press (1996)
21. Muggleton, S.H.: Learning stochastic logic programs. Electron. Trans. Artif. Intell. **4**(B), 141–153 (2000), http://www.ep.liu.se/ej/etai/2000/015/
22. Pontelli, E., Son, T.C., El-Khatib, O.: Justifications for logic programs under answer set semantics. Theory Pract. Log. Program. **9**(1), 1–56 (2009). https://doi.org/10.1017/S1471068408003633

23. Poole, D.: Logic programming, abduction and probability - A top-down anytime algorithm for estimating prior and posterior probabilities. New Gener. Comput. **11**(3), 377–400 (1993). https://doi.org/10.1007/BF03037184
24. Poole, D.: Probabilistic horn abduction and bayesian networks. Artif. Intell. **64**(1), 81–129 (1993). https://doi.org/10.1016/0004-3702(93)90061-F
25. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artif. Intell. **94**(1-2), 7–56 (1997). https://doi.org/10.1016/S0004-3702(97)00027-1
26. Poole, D.: Abducing through negation as failure: stable models within the independent choice logic. J. Log. Program. **44**(1-3), 5–35 (2000). https://doi.org/10.1016/S0743-1066(99)00071-0
27. Raedt, L.D., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007). pp. 2462–2467 (2007), http://ijcai.org/Proceedings/07/Papers/396.pdf
28. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987). https://doi.org/10.1016/0004-3702(87)90062-2
29. Riguzzi, F.: A top down interpreter for LPAD and cp-logic. In: Basili, R., Pazienza, M.T. (eds.) AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4733, pp. 109–120. Springer (2007). https://doi.org/10.1007/978-3-540-74782-6_11, https://doi.org/10.1007/978-3-540-74782-6_11
30. Riguzzi, F.: Extended semantics and inference for the independent choice logic. Log. J. IGPL **17**(6), 589–629 (2009). https://doi.org/10.1093/JIGPAL/JZP025
31. Riguzzi, F.: SLGAD resolution for inference on logic programs with annotated disjunctions. Fundam. Informaticae **102**(3-4), 429–466 (2010). https://doi.org/10.3233/FI-2010-313
32. Riguzzi, F.: Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning. River Publishers (2018)
33. Riguzzi, F., Swift, T.: Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics. Theory Pract. Log. Program. **13**(2), 279–302 (2013). https://doi.org/10.1017/S1471068411000664, https://doi.org/10.1017/S1471068411000664
34. Riguzzi, F., Swift, T.: A survey of probabilistic logic programming. In: Kifer, M., Liu, Y.A. (eds.) Declarative Logic Programming: Theory, Systems, and Applications, ACM Books, vol. 20, pp. 185–228. ACM / Morgan & Claypool (2018). https://doi.org/10.1145/3191315.3191319
35. Ross, K.A.: Modular acyclicity and tail recursion in logic programs. In: Rosenkrantz, D.J. (ed.) Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. pp. 92–101. ACM Press (1991). https://doi.org/10.1145/113413.113422
36. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995. pp. 715–729. MIT Press (1995)
37. Sato, T., Kameya, Y.: PRISM: A language for symbolic-statistical modeling. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes. pp. 1330–1339. Morgan Kaufmann (1997), http://ijcai.org/Proceedings/97-2/Papers/078.pdf

38. Shterionov, D.S., Renkens, J., Vlasselaer, J., Kimmig, A., Meert, W., Janssens, G.: The most probable explanation for probabilistic logic programs with annotated disjunctions. In: Davis, J., Ramon, J. (eds.) Proceedings of the 24th International Conference on Inductive Logic Programming (ILP 2014). Lecture Notes in Computer Science, vol. 9046, pp. 139–153. Springer (2014). https://doi.org/10.1007/978-3-319-23708-4_10
39. Vennekens, J., Denecker, M., Bruynooghe, M.: CP-logic: A language of causal probabilistic events and its relation to logic programming. Theory Pract. Log. Program. **9**(3), 245–308 (2009). https://doi.org/10.1017/S1471068409003767, https://doi.org/10.1017/S1471068409003767
40. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3132, pp. 431–445. Springer (2004). https://doi.org/10.1007/978-3-540-27775-0_30
41. Vidal, G.: Explanations as programs in probabilistic logic programming. In: Hanus, M., Igarashi, A. (eds.) Proceedings of the 16th International Symposium on Functional and Logic Programming (FLOPS 2022). Lecture Notes in Computer Science, vol. 13215, pp. 205–223. Springer (2022). https://doi.org/10.1007/978-3-030-99461-7_12

A Technical Proofs

First, we will prove that $\langle \widetilde{\mathbb{C}}, \wedge, \vee, \neg, \top, \bot \rangle$ is indeed a Boolean algebra. Let us recall that $\mathsf{C}_1 \sim \mathsf{C}_2$ if $\omega_{\gamma(\mathsf{C}_1)} = \omega_{\gamma(\mathsf{C}_2)}$ and $\widetilde{\mathbb{C}}$ denotes the quotient set of $\mathbb{C}$ (the domain of choice expressions for a given program) by "$\sim$", where

- $\gamma(\bot) = \{\}$, i.e., $\bot$ denotes an inconsistent set of atomic choices.
- $\gamma(\top) = \{\{\}\}$, i.e., $\top$ represents a composite choice, $\{\}$, that can be extended in order to produce all possible selections.
- $\gamma(\alpha) = \{\{\alpha\}\}$.
- $\gamma(\neg\mathsf{C}) = \mathsf{duals}(\gamma(\mathsf{C}))$, i.e., $\neg\mathsf{C}$ represents the complement of C.
- $\gamma(\mathsf{C}_1 \wedge \mathsf{C}_2) = \mathsf{mins}(\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2))$.
- $\gamma(\mathsf{C}_1 \vee \mathsf{C}_2) = \mathsf{mins}(\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_2))$.

Here, $\mathsf{C} \in \widetilde{\mathbb{C}}$ denote the equivalence class $[\mathsf{C}]$ when no confusion can arise.

Moreover, we recall the following lemma from [26]:

Lemma 1 (Lemma 4.8 in [26]). *Let K be a set of composite choices. Then, $\mathsf{duals}(K)$ is a complement of K.*

Now, we can prove that $\langle \widetilde{\mathbb{C}}, \wedge, \vee, \neg, \top, \bot \rangle$ is a Boolean algebra.

Proposition 1. *$\langle \widetilde{\mathbb{C}}, \wedge, \vee, \neg, \top, \bot \rangle$ is a Boolean algebra.*

Proof. We prove the axioms of a Boolean algebra. Here, to prove that two choice expressions, C_1 and C_2, are equivalent, we will prove that $\gamma(\mathsf{C}_1) \approx \gamma(\mathsf{C}_2)$, where $K_1 \approx K_2$ if $K_1 = K_2$ or $\omega_{K_1} = \omega_{K_2}$. In the following, we assume $\mathsf{C}, \mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3, \ldots$ are choice expressions that denote the corresponding class. Moreover, we ignore the applications of function mins for clarity since it cannot affect the result, only to the element of the class which is shown.

(Associativity) The first axiom $\mathsf{C}_1 \vee (\mathsf{C}_2 \vee \mathsf{C}_3) = (\mathsf{C}_1 \vee \mathsf{C}_2) \vee \mathsf{C}_3$ follows from the associativity of set union since $\gamma(\mathsf{C}_1 \vee \mathsf{C}_2) = \gamma(C_1) \cup \gamma(C_2)$. Similarly, the second axiom $\mathsf{C}_1 \wedge (\mathsf{C}_2 \wedge \mathsf{C}_3) = (\mathsf{C}_1 \wedge \mathsf{C}_2) \wedge \mathsf{C}_3$ follows from the fact that "$\otimes$" is associative over sets since $\gamma(\mathsf{C}_1 \wedge \mathsf{C}_2) = \gamma(C_1) \otimes \gamma(C_2)$.

(Commutativity) Again, both axioms $\mathsf{C}_1 \vee \mathsf{C}_2 = \mathsf{C}_2 \vee \mathsf{C}_1$ and $\mathsf{C}_1 \wedge \mathsf{C}_2 = \mathsf{C}_2 \wedge \mathsf{C}_1$ follow straightforwardly from the commutativity of "$\cup$" and "$\otimes$".

(Absorption) Axiom $\mathsf{C}_1 \vee (\mathsf{C}_1 \wedge \mathsf{C}_2) = \mathsf{C}_1$ follows from the fact that the composite choices in $\gamma(\mathsf{C}_1 \wedge \mathsf{C}_2)$ are all redundant (i.e., do not affect to the considered class) w.r.t. $\gamma(\mathsf{C}_1)$ since they will be supersets of the sets in $\gamma(\mathsf{C}_1)$. As for axiom $\mathsf{C}_1 \wedge (\mathsf{C}_1 \vee \mathsf{C}_2) = \mathsf{C}_1$, it follows from the following equivalences:

$$
\begin{aligned}
&\gamma(\mathsf{C}_1 \wedge (\mathsf{C}_1 \vee \mathsf{C}_2)) \\
&= \gamma(\mathsf{C}_1) \otimes (\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_2)) \\
&= (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_1)) \cup (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2)) \\
&\approx \gamma(\mathsf{C}_1) \cup (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2)) \\
&= \gamma(\mathsf{C}_1)
\end{aligned}
$$

since all composite choices in $\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2)$ are supersets of those in $\gamma(\mathsf{C}_1)$ and, thus, do not affect to the considered class.

(Identity) Both axioms $\mathsf{C} \vee \bot = \mathsf{C}$ and $\mathsf{C} \wedge \top = \mathsf{C}$ are trivial by definition (e.g., by taking $\gamma(\bot) = \{\}$ and $\gamma(\top) = \{\{\}\}$).

(Distributivity) The first axiom, $\mathsf{C}_1 \vee (\mathsf{C}_2 \wedge \mathsf{C}_3) = (\mathsf{C}_1 \vee \mathsf{C}_2) \wedge (\mathsf{C}_1 \vee \mathsf{C}_3)$ follows from the following equivalences:

$$\begin{aligned}
&\gamma((\mathsf{C}_1 \vee \mathsf{C}_2) \wedge (\mathsf{C}_1 \vee \mathsf{C}_3)) \\
&= (\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_2)) \otimes (\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_3)) \\
&= (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_1)) \cup (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_3)) \cup (\gamma(\mathsf{C}_2) \otimes \gamma(\mathsf{C}_1)) \cup (\gamma(\mathsf{C}_2) \otimes \gamma(\mathsf{C}_3)) \\
&\approx \gamma(\mathsf{C}_1) \cup (\gamma(\mathsf{C}_2) \otimes \gamma(\mathsf{C}_3)) \\
&= \gamma(\mathsf{C}_1 \vee (\mathsf{C}_2 \wedge \mathsf{C}_3))
\end{aligned}$$

The second axiom, $\mathsf{C}_1 \wedge (\mathsf{C}_2 \vee \mathsf{C}_3) = (\mathsf{C}_1 \wedge \mathsf{C}_2) \vee (\mathsf{C}_1 \wedge \mathsf{C}_3)$ follows easily since

$$\begin{aligned}
&\gamma(\mathsf{C}_1 \wedge (\mathsf{C}_2 \vee \mathsf{C}_3)) \\
&= \gamma(\mathsf{C}_1) \otimes (\gamma(\mathsf{C}_2) \cup \gamma(\mathsf{C}_3)) \\
&= (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2)) \cup (\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_3)) \\
&= \gamma((\mathsf{C}_1 \wedge \mathsf{C}_2) \vee (\mathsf{C}_1 \wedge \mathsf{C}_3))
\end{aligned}$$

(Complements) Axiom $\mathsf{C} \vee \neg\mathsf{C} = \top$ follows from Lemma 1. Thus, $\gamma(\mathsf{C}) \cup \gamma(\neg\mathsf{C}) = \gamma(\mathsf{C}) \cup \mathsf{duals}(\gamma(\mathsf{C}))$ cover all possible selections. Axiom $\mathsf{C} \wedge \neg\mathsf{C} = \bot$ holds since every element in $\gamma(\mathsf{C}) \otimes \mathsf{duals}(\gamma(\mathsf{C}))$ is inconsistent by construction. □

The following auxiliary lemma is required for the remaining results:

Lemma 2. *Let K_1, K_2 be sets of composite choices. Then,* $\mathsf{mins}(\mathsf{hits}(K_1 \otimes K_2)) = \mathsf{mins}(\mathsf{hits}(K_1) \cup \mathsf{hits}(K_2))$.

Proof. Trivially, we have $\mathsf{hits}(K_1) \subseteq \mathsf{hits}(K_1 \otimes K_2)$ and $\mathsf{hits}(K_2) \subseteq \mathsf{hits}(K_1 \otimes K_2)$. Then, the claim follows from the fact that the hitting sets that combine elements from K_1 and K_2 are redundant and are removed by mins since they include the hitting sets in either $\mathsf{hits}(K_1)$ or $\mathsf{hits}(K_2)$. □

Double negation elimination and the usual De Morgan's laws also hold for choice expressions:

Proposition 2. *Let $\mathsf{C}, \mathsf{C}_1, \mathsf{C}_2 \in \widetilde{\mathbb{C}}$. Then,*

1. $\neg\neg\mathsf{C} = \mathsf{C}$;
2. $\neg(\mathsf{C}_1 \vee \mathsf{C}_2) = \neg\mathsf{C}_1 \wedge \neg\mathsf{C}_2$;
3. $\neg(\mathsf{C}_1 \wedge \mathsf{C}_2) = \neg\mathsf{C}_1 \vee \neg\mathsf{C}_2$.

Proof. We follow the same considerations as in the proof of Proposition 1. The proof of the double negation elimination follows straightforwardly from Lemma 1 and the fact that the complement of a complement returns the original set or another one which belongs to the same equivalence class, i.e., $\gamma(\neg\neg\mathsf{C}) = \mathsf{duals}(\gamma(\neg\mathsf{C})) = \mathsf{duals}(\mathsf{duals}(\gamma(\mathsf{C}))) = \gamma(\mathsf{C})$.

The first De Morgan's law can be proved as follows:

$$
\begin{aligned}
&\gamma(\neg(\mathsf{C}_1 \vee \mathsf{C}_2))\\
&= \mathsf{duals}(\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_2))\\
&= \mathsf{hits}(\overline{\gamma(\mathsf{C}_1) \cup \gamma(\mathsf{C}_2)})\\
&= \mathsf{hits}(\overline{\gamma(\mathsf{C}_1)} \cup \overline{\gamma(\mathsf{C}_2)})\\
&\approx \mathsf{hits}(\overline{\gamma(\mathsf{C}_1)} \otimes \overline{\gamma(\mathsf{C}_2)}) \quad \text{(by Lemma 2)}\\
&= \mathsf{hits}(\gamma(\mathsf{C}_1)) \otimes \mathsf{hits}(\gamma(\mathsf{C}_2))\\
&= \mathsf{duals}(\gamma(\mathsf{C}_1)) \otimes \mathsf{duals}(\gamma(\mathsf{C}_2))\\
&= \gamma(\neg\mathsf{C}_1) \otimes \gamma(\neg\mathsf{C}_2)\\
&= \gamma(\neg\mathsf{C}_1 \wedge \neg\mathsf{C}_2)
\end{aligned}
$$

The proof of the second De Morgan's law proceeds analogously:

$$
\begin{aligned}
&\gamma(\neg(\mathsf{C}_1 \wedge \mathsf{C}_2))\\
&= \mathsf{duals}(\gamma(\mathsf{C}_1 \wedge \mathsf{C}_2))\\
&= \mathsf{duals}(\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2))\\
&= \mathsf{hits}(\overline{\gamma(\mathsf{C}_1) \otimes \gamma(\mathsf{C}_2)})\\
&= \mathsf{hits}(\overline{\gamma(\mathsf{C}_1)} \otimes \overline{\gamma(\mathsf{C}_2)})\\
&\approx \mathsf{hits}(\overline{\gamma(\mathsf{C}_1)}) \cup \mathsf{hits}(\overline{\gamma(\mathsf{C}_2)}) \quad \text{(by Lemma 2)}\\
&= \mathsf{duals}(\gamma(\mathsf{C}_1)) \cup \mathsf{duals}(\gamma(\mathsf{C}_2))\\
&= \gamma(\neg\mathsf{C}_1)) \cup \gamma(\neg\mathsf{C}_2))\\
&= \gamma(\neg\mathsf{C}_1 \vee \neg\mathsf{C}_2)
\end{aligned}
$$

□

Finally, we can prove the soundness and completeness of SLPDNF-resolution, i.e., that $\mathsf{expl}(Q)$ indeed produces a set of covering explanations for Q.

Theorem 1. *Let $\mathcal{P}$ be a sound program and Q a ground query. Then, $\omega_s \models Q$ iff there exists a composite choice $\kappa \in \mathsf{expl}_{\mathcal{P}}(Q)$ such that $\kappa \subseteq s$.*

Proof. The proof follows a similar scheme as that of the proof of Theorem 4.7 in [30] given the fact that $\neg\mathsf{C}$ represents a complement of C by definition and that function dnf preserves the worlds represented by a choice expression (Propositions 1 and 2).

We prove the theorem by structural induction on the set of trees in Γ, the SLPDNF-tree for Q.

Let us first consider the base case, where Γ only contains the main tree. Therefore, no negative literal has been selected. Consider a successful SLPDNF-derivation $\langle Q, \top\rangle = \langle Q_0, \top\rangle \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} \langle Q_n, \mathsf{C}_n\rangle$. By definition, we have that $\mathsf{C}_n = \alpha_1 \wedge \ldots \wedge \alpha_m$, where each α_i is a (positive) atomic choice, $i = 1, \ldots, n$. Trivially, $\gamma(\mathsf{C}_n) = \{\{\alpha_1, \ldots, \alpha_m\}\} = \{\kappa\}$ and κ is consistent by construction and the fact that function dnf preserves the worlds (Propositions 1 and 2). Thus, the SLDNF-derivation $Q_0 \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} Q_n$ can be proved in every world ω_s where $\kappa \subseteq s$ and, equivalently, $\omega_s \models Q$.

The opposite direction is similar. Let ω_s be a world such that $\omega_s \models Q$. Then, there exists an SLDNF-derivation for Q in ω_s. Now, we can construct an SLPDNF-derivation that mimics the steps of the SLDNF-derivation by applying either the first or the second case in the definition of SLPDNF-tree. If the considered predicate is not probabilistic, the equivalence is trivial. Otherwise, let us consider that clause $c\theta = (h_i \leftarrow B)\theta$ from ω_s has been used in the step. Therefore, the SLPDNF-step will add the atomic choice (c, θ, i) to the current choice expression. Hence, the computed choice expression in the leaf of this derivation will have the form $\alpha_1 \wedge \ldots \wedge \alpha_m$ so that $\{\alpha_1, \ldots, \alpha_m\} \subseteq s$.

Let us now consider the inductive case (Γ includes more than one tree). By the inductive hypothesis, we assume that the claim holds for every SLPDNF-tree of Γ which is not the main tree. Then, for each step, if the selected literal is positive, the proof proceeds as in the base case. Otherwise, let $\neg a$ be the selected literal. By the inductive hypothesis, we have that, $\omega_s \models a$ iff there exists a composite choice $\kappa \in \mathsf{expl}_{\mathcal{P}}(a)$ such that $\kappa \subseteq s$. Let $\mathsf{C}_1^a, \ldots, \mathsf{C}_j^a$ be the choice expressions in the leaves of the SLPDNF-tree for a. By definition, we have that every selection that extends a composite choice in $\gamma(\neg(\mathsf{C}_1^a \vee \ldots \vee \mathsf{C}_j^a))$ is inconsistent with any selection s with $\omega_s \models a$. Therefore, no SLDNF-derivation for a can be successful in the worlds of $\omega_{\gamma(\neg(\mathsf{C}_1^a \vee \ldots \vee \mathsf{C}_j^a))}$ and the claim follows.

Consider now the opposite direction. Let ω_s be a world such that $\omega_s \models Q$ and, thus, there exists a successful SLDNF-derivation for Q in ω_s. As in the base case, we can construct an SLPDNF-derivation that mimics the steps of the SLDNF-derivation in ω_s. If the selected literal is positive, the claim follows by the same argument as the base case. Otherwise, consider a negative literal $\neg a$ which is selected in some query. By the inductive hypothesis, we have that $\mathsf{expl}(a)$ is indeed a covering set of explanations for a. Moreover, by construction, the negated choice expression, say $\neg\mathsf{C}_a$ indeed represents a complement of this set of explanations (i.e., $\gamma(\neg\mathsf{C}_a)$ is a complement of $\gamma(\mathsf{C}_a)$). Hence, this query has a child with an associated choice expression, $\mathsf{dnf}(\mathsf{C} \wedge \neg\mathsf{C}_a) \neq \bot$. Therefore by mimicking all the steps of the successful SLDNF-derivation, we end up with a leaf with a choice expression C' such that $s \supseteq \kappa$ for all $\kappa \in \gamma(\mathsf{C}')$. □