A Coalgebraic Perspective on Probabilistic Logic Programming

Tao Gu
University College London, London, UK
tao.gu.18@ucl.ac.uk

Fabio Zanasi
University College London, London, UK
f.zanasi@ucl.ac.uk

Abstract

Probabilistic logic programming is increasingly important in artificial intelligence and related fields as a formalism to reason about uncertainty. It generalises logic programming with the possibility of annotating clauses with probabilities. This paper proposes a coalgebraic perspective on probabilistic logic programming. Programs are modelled as coalgebras for a certain functor F, and two semantics are given in terms of cofree coalgebras. First, the cofree F-coalgebra yields a semantics in terms of derivation trees. Second, by embedding F into another type G, as cofree G-coalgebra we obtain a 'possible worlds' interpretation of programs, from which one may recover the usual distribution semantics of probabilistic logic programming.

2012 ACM Subject Classification Theory of computation; Theory of computation → Logic

Keywords and phrases probabilistic logic programming, coalgebraic semantics, distribution semantics

Digital Object Identifier 10.4230/LIPIcs.CALCO.2019.5

Funding *Fabio Zanasi*: partial support from EPSRC grant n. EP/R020604/1.

Acknowledgements Fabio Zanasi acknowledges support from EPSRC grant n. EP/R020604/1. The authors thank Alessandro Facchini for useful pointers to the literature, and the anonymous reviewers for the useful comments and feedback.

1 Introduction

Probabilistic logic programming (PLP) [23, 5, 25] is a family of approaches extending the declarative paradigm of logic programming with the possibility of reasoning about uncertainty. This has been proven useful in various applications, including bioinformatics [6, 22], robotics [27] and the semantic web [29].

The most common version of PLP — on which for instance ProbLog is based [6], the probabilistic analogue of Prolog — is defined by letting clauses in programs to be annotated with mutually independent probabilities. As for the interpretation, *distribution semantics* [25] is typically used as a benchmark for the various implementations of PLP, such as pD, PRISM and ProbLog [24]. In this semantics, the probability of refuting a goal in a program is obtained as a sum of the probabilities of the *possible worlds* (sets of clauses) in which the goal is refutable. The distribution semantics is particularly interesting because it is compatible with the encoding of Bayesian networks as probabilistic logic programs [24], thus indicating that PLP can be effectively employed for Bayesian reasoning.

The main goal of this work is to present a coalgebraic perspective on PLP and its distribution semantics. We first consider the case of ground programs, that is, those without variables. Our approach is based on the observation — inspired by the coalgebraic treatment of 'pure' logic programming [16] — that ground programs are in 1-1 correspondence with coalgebras

8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019).
Editors: Markus Roggenbach and Ana Sokolova; Article No. 5; pp. 5:1–5:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for the functor $\mathcal{M}_{pr}\mathcal{P}_f$, where $\mathcal{M}_{pr}$ is the finite multiset functor on $[0,1]$ and $\mathcal{P}_f$ is the finite powerset functor. We then provide two coalgebraic semantics for ground PLP.

- The first interpretation $[\![-]\!]$ is in terms of execution trees called *stochastic derivation trees*, which represent parallel SLD-derivations of a program on a goal. Stochastic derivation trees are the elements of the cofree $\mathcal{M}_{pr}\mathcal{P}_f$-coalgebra on a given set of atoms At, meaning that any goal $A \in \mathsf{At}$ can be given a semantics in terms of the corresponding stochastic derivation tree by the universal property map $[\![-]\!]$ to the cofree coalgebra.
- The second interpretation $\langle\!\langle - \rangle\!\rangle$ recovers the usual distribution semantics of PLP. This requires some work, as expressing probability distributions on the possible worlds needs a different coalgebra type. We introduce *distribution trees*, a tree-like representation of the distribution semantics, as the elements of the cofree $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$-coalgebra on At, where $\mathcal{D}_{\leq 1}$ is the sub-probability distribution monad. In order to characterise $\langle\!\langle - \rangle\!\rangle$ as the map given by universal property of distribution trees, we need a canonical extension of PLP to the setting of $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$-coalgebras. This is achieved via a 'possible worlds' natural transformation $\mathcal{M}_{pr}\mathcal{P}_f \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$.

In the second part of the paper we recover the same framework for arbitrary probabilistic logic programs, possibly including variables. The encoding of programs as coalgebras is subtler. The space of atoms is now a presheaf indexed by a 'Lawvere theory' of terms and substitutions. The coalgebra map can be defined in different ways, depending on the substitution mechanism on which one wants to base resolution. For pure logic programs, the definition by term matching is the best studied, with [17] observing that moving from sets to posets is required in order for the corresponding coalgebra map to be well-defined as a natural transformation between presheaves. A different route is taken in [3], where the problem of naturality is neutralised via 'saturation', a categorical construction which amounts to defining resolution by unification instead of term-matching.

In developing a coalgebraic treatment of PLP with variables, we follow the saturation route, as it also allows to recover the term-matching approach, via 'desaturation' [3]. This provides a cofree coalgebra semantics $[\![-]\!]$ for arbitrary PLP programs, as a rather straightforward generalisation of the saturated semantics of pure logic programs. On the other hand, extending the ground distribution semantics $\langle\!\langle - \rangle\!\rangle$ to arbitrary PLP programs poses some challenges: we need to ensure that, in computing the distribution over possible worlds associated to each sub-goal in the computation, each clause of the program is 'counted' only once. This is solved by tweaking the coalgebra type of the distribution trees for arbitrary PLP programs, so that some nodes are labelled with clauses of the program. Thanks to this additional information, the term-matching distribution semantics of an arbitrary PLP goal is computable from its distribution tree.

In light of the coalgebraic treatment of pure logic programming [16, 17, 2, 3], the generalisation to PLP may not appear so surprising. In fact, we believe its importance is two-fold. First, whereas the derivation semantics $[\![-]\!]$ is a straight generalisation of the pure setting, the distribution semantics $\langle\!\langle - \rangle\!\rangle$ is genuinely novel, and does not have counterparts in pure logic programming. Second, a paper dedicated to establishing the foundations of coalgebraic PLP is a necessary preliminary step for a number of interesting applications:

- as mentioned, reasoning in Bayesian networks can be seen as a particular case of PLP, equipped with the distribution semantics. Our coalgebraic perspective thus readily applies to Bayesian reasoning, paving the way for combination with recent works [11, 12, 4] modelling belief revision, causal inference and other Bayesian tasks in algebraic terms.
- the combination of logic programming and probabilities comes in different flavours [24]: the more abstract viewpoint offered by coalgebra may provide a unifying perspective on

these approaches, as well as a formal connection with seemingly related languages such as weighted logic programming [8] and Bayesian logic programming [13].

- the coalgebraic treatment of pure logic programming has been used as a formal justification [19, 14] for coinductive logic programming [15, 9]. Coinduction in the context of probabilistic logic programs is, to the best of our knowledge, a completely unexplored field, for which the current paper establishes semantic foundations.

We leave the exploration of these venues as follow-up work.

2 Preliminaries

Signature, Terms, and Categories. A *signature* Σ is a set of function symbols, each equipped with a fixed finite arity. Throughout this paper we fix a signature Σ, and a countably infinite set of variables $Var = \{x_1, x_2, \dots\}$. The Σ-terms over Var are defined as usual. A *context* is a finite set of variables $\{x_1, x_2, \dots, x_n\}$. With some abuse of notation, we shall often use n to denote this context. We say a Σ-term t is *compatible* with context n if the variables appearing in t are all contained in $\{x_1, \dots, x_n\}$.

We are going to reason about Σ-terms categorically using Lawvere theories. First, we will use $\mathsf{Ob}(\mathbf{C})$ to denote the set of objects and $\mathbf{C}[C, D]$ for the set of morphisms $C \to D$ in a category $\mathbf{C}$. A $\mathbf{C}$-indexed *presheaf* is a functor $\mathsf{F} \colon \mathbf{C} \to \mathbf{Sets}$. $\mathbf{C}$-indexed presheaves and natural transformations between them form a category $\mathbf{Sets}^{\mathbf{C}}$. Recall that the (opposite) Lawvere Theory of Σ is the category $\mathbf{L}_{\Sigma}^{\mathrm{op}}$ with objects the natural numbers and morphisms $\mathbf{L}_{\Sigma}^{\mathrm{op}}[n, m]$ the n-tuples $\langle t_1, \dots, t_n \rangle$, where each t_i is a Σ-term in context m. For modelling logic programming, it is convenient to think of each $n \in \mathsf{Ob}(\mathbf{L}_{\Sigma}^{\mathrm{op}})$ as representing the context $\langle x_1, \dots, x_n \rangle$, and a morphism $\langle t_1, \dots, t_n \rangle : n \to m$ as the substitution transforming Σ-terms in context n to Σ-terms in context m by replacing each x_i with t_i. We shall also refer to $\mathbf{L}_{\Sigma}^{\mathrm{op}}$ morphisms simply as substitutions (notation $\theta, \tau, \sigma, \dots$).

Logic programming. We now recall the basics of (pure) logic programming, and refer the reader to [20] for a more systematic exposition. An *alphabet* $\mathcal{A}$ consists of a signature Σ, a set of variables Var, and a set of predicate symbols $\{P_1, P_2, \dots\}$, each with a fixed arity. Given an n-ary predicate symbol P in $\mathcal{A}$, and Σ-terms $t_1, \dots, t_n$, $P(t_1 \cdots t_n)$ is called an *atom* over $\mathcal{A}$. We use $A, B, \dots$ to denote atoms. Given an atom A in context n, and a substitution $\theta = \langle t_1, \dots, t_n \rangle : n \to m$, we write $A\theta$ for *substitution instance* of A obtained by replacing each appearance of x_i with t_i in A. For convenience, we also use $\{B_1, \dots, B_k\}\theta$ as a shorthand for $\{B_1\theta, \dots, B_k\theta\}$. Given two atoms A and B (over $\mathcal{A}$), a *unifier* of A and B is a pair $\langle \sigma, \tau \rangle$ of substitutions such that $A\sigma = B\tau$. *Term matching* is a special case of unification, where σ is the identity substitution. In this case we say that τ matches B with A if $A = B\tau$.

A (pure) logic program $\mathbb{L}$ consists of a finite set of clauses $\mathcal{C}$ in the form $H \leftarrow B_1, \dots, B_k$, where $H, B_1, \dots, B_k$ are atoms. H is called the *head* of $\mathcal{C}$, and $B_1, \dots, B_k$ form the *body* of $\mathcal{C}$. We denote H by $\mathsf{Head}(\mathcal{C})$, and $\{B_1, \dots, B_k\}$ by $\mathsf{Body}(\mathcal{C})$. A *goal* is simply an atom. Since one can regard a clause $H \leftarrow B_1, \dots, B_k$ as the logic formula $B_1 \wedge \cdots \wedge B_k \to H$, we say that a goal G is *derivable* in $\mathbb{L}$ if there exists a derivation of G with empty assumption using the clauses in $\mathbb{L}$.

The central task of logic programming is to check whether a goal G is *provable* in a program $\mathbb{L}$, in the sense that some substitution instance of G is derivable in $\mathbb{L}$. The key algorithm for this task is SLD-resolution, see e.g. [20]. We use the notation $\mathbb{L} \vdash G$ to mean that G is provable in $\mathbb{L}$.

Probabilistic logic programming. We now recall the basics of PLP; the reader may consult [7, 6] for a more comprehensive introduction. A probabilistic logic program $\mathbb{P}$ based on a logic program $\mathbb{L}$ assigns a probability label r to each clause $\mathcal{C}$ in $\mathbb{L}$, denoted as $\mathsf{Label}(\mathcal{C})$. One may also regard $\mathbb{P}$ as a set of probabilistic clauses of the form $r :: \mathcal{C}$, where $\mathcal{C}$ is a clause in $\mathbb{L}$, and each clause $\mathcal{C}$ is assigned a unique probability label r in $\mathbb{P}$. We also refer to $r :: \mathcal{C}$ simply as clauses.

▶ **Example 1.** As our leading example we introduce the following probabilistic logic program $\mathbb{P}^{al}$. It models the scenario of Mary's house alarm, which is supposed to detect burglars, but it may be accidentally triggered by an earthquake. Mary may hear the alarm if she is awake, but even if the alarm is not sounding, in case she experiences an auditory hallucination (paracusia). The language of $\mathbb{P}^{al}$ includes 0-ary predicates Alarm, Eearthquake, Burglary, and 1-ary predicates $\mathsf{Wake}(-)$, $\mathsf{Hear\_alarm}(-)$ and $\mathsf{Paracusia}(-)$, and signature $\Sigma_{\mathsf{al}} = \{\mathsf{Mary}^0\}$ consisting of a constant. We do not have variables here, so $\mathbb{P}^{al}$ is a ground program. For readability we abbreviate Mary as M in the program.

$$\begin{array}{rlll|rlll} 0.01 :: & \mathsf{Earthquake} & \leftarrow & & 0.01 :: & \mathsf{Paracusia(M)} & \leftarrow & \\ 0.2 :: & \mathsf{Burglary} & \leftarrow & & 0.6 :: & \mathsf{Wake(M)} & \leftarrow & \\ 0.5 :: & \mathsf{Alarm} & \leftarrow & \mathsf{Earthquake} & 0.8 :: & \mathsf{Hear\_alarm(M)} & \leftarrow & \mathsf{Alarm},\ \mathsf{Wake(M)} \\ 0.9 :: & \mathsf{Alarm} & \leftarrow & \mathsf{Burglary} & 0.3 :: & \mathsf{Hear\_alarm(M)} & \leftarrow & \mathsf{Paracusia(M)} \end{array}$$

As a generalisation of the pure case, in probabilistic logic programming one is interested in the *probability* of a goal G being refutable in a program $\mathbb{P}$. There are potentially multiple ways to define such probability— in this paper we focus on the *distribution semantics* [7].

The probability of refuting a goal is computed in the distribution semantics as follows. Given a probabilistic logic program $\mathbb{P} = \{p_1 :: \mathcal{C}_1, \dots, p_n :: \mathcal{C}_n\}$, let $|\mathbb{P}|$ be its underlying pure logic program, namely $|\mathbb{P}| = \{\mathcal{C}_1, \dots, \mathcal{C}_n\}$. A *sub-program* $\mathbb{L}$ of $|\mathbb{P}|$ is a logic program consisting of a subset of the clauses in $|\mathbb{P}|$. This justifies using $\mathcal{P}(|\mathbb{P}|)$ to denote the set of all sub-programs of $|\mathbb{P}|$, and using $\mathbb{L} \subseteq |\mathbb{P}|$ to denote that $\mathbb{L}$ is a sub-program of $\mathbb{P}$. The central concept of the distribution semantics is that $\mathbb{P}$ determines a distribution $\mu_{\mathbb{P}}$ over the sub-programs $\mathcal{P}(|\mathbb{P}|)$: for any $\mathbb{L} \in \mathcal{P}(|\mathbb{P}|)$, $\mu_{\mathbb{P}}(\mathbb{L}) := \prod_{\mathcal{C}_i \in \mathbb{L}} p_i \prod_{\mathcal{C}_j \in |\mathbb{P}| \setminus \mathbb{L}} (1 - p_j)$. The value $\mu_{\mathbb{P}}(\mathbb{L})$ is called the *probability* of the sub-program $\mathbb{L}$. For an arbitrary goal $G \in \mathsf{At}$, the *success probability* $\mathsf{Pr}_{\mathbb{P}}(G)$ of G w.r.t. program $\mathbb{P}$ is then defined as the sum of the probabilities of all the sub-programs of $\mathbb{P}$ in which G is refutable:

$$\mathsf{Pr}_{\mathbb{P}}(G) := \sum_{|\mathbb{P}| \supseteq \mathbb{L} \vdash G} \mu_{\mathbb{P}}(\mathbb{L}) = \sum_{|\mathbb{P}| \supseteq \mathbb{L} \vdash G} \Big(\prod_{\mathcal{C}_i \in \mathbb{L}} p_i \prod_{\mathcal{C}_j \in |\mathbb{P}| \setminus \mathbb{L}} (1 - p_j) \Big) \tag{1}$$

Intuitively one can regard every clause in $\mathbb{P}$ as an event, then every sub-program $\mathbb{L}$ can be seen as a possible world, and $\mu_{\mathbb{P}}$ is a distribution over the possible worlds.

▶ **Example 2.** For the program $\mathbb{P}^{al}$, consider the goal $\mathsf{Hear\_alarm(M)}$. By definition (1), we can compute its success probability $\mathsf{Pr}_{\mathbb{P}^{al}}(\mathsf{Hear\_alarm(M)})$, and the result is 0.091102896.

3 Ground case

In this section we introduce a coalgebraic semantics for *ground* probabilistic logic programming, i.e. for those programs where no variable appears. Our approach consists of two parts. First, in Subsection 3.1, we represent PLP logic programs as coalgebras and their executions as a final coalgebra semantics (Subsection 3.2)— this is a straight generalisation of the coalgebraic treatment of pure logic programs given in [16]. Next, in Subsection 3.3 we show

how to represent the distribution semantics in terms as a final coalgebra, via a transformation of the coalgebra type of logic programs. Appendix A shows how the probability of a goal is effectively computable from the above representation.

3.1 Coalgebraic Representation of PLP

A ground program will be represented as a coalgebra for the composite $\mathcal{M}_{pr}\mathcal{P}_f\colon \mathbf{Sets} \to \mathbf{Sets}$ of the finite probability functor $\mathcal{M}_{pr}\colon \mathbf{Sets} \to \mathbf{Sets}$ and the finite powerset functor $\mathcal{P}_f\colon \mathbf{Sets} \to \mathbf{Sets}$. The definition of $\mathcal{M}_{pr}$ deserves some further explanation. It can be seen as the finite multiset functor based on the commutative monoid $([0,1], 0, \vee)$, where $a \vee b \coloneqq 1-(1-a)(1-b)$. That is to say, on objects, $\mathcal{M}_{pr}(A)$ is the set of all *finite probability assignments* $\varphi\colon A \to [0,1]$ with a finite support $\mathsf{supp}(\varphi) \coloneqq \{a \in A \mid \varphi(a) \neq 0\}$. For φ with support $\{a_1, \dots, a_k\}$ and values $\varphi(a_i) = r_i$, it will often be convenient to use the standard notation $\varphi = \sum_{i=1}^{k} r_i a_i$ or $\varphi = r_1 a_1 + \cdots + r_k a_k$, where the purely formal "+" here should not be confused with the addition in $\mathbb{R}$. On morphisms, $\mathcal{M}_{pr}(h\colon A \to B)$ maps $\sum_{i=1}^{k} r_i a_i$ to $\sum_{i=1}^{k} r_i h(a_i)$.

Fix a ground probabilistic logic program $\mathbb{P}$ on a set of ground atoms At. The definition of $\mathbb{P}$ can be encoded as an $\mathcal{M}_{pr}\mathcal{P}_f$-coalgebra $p\colon \mathsf{At} \to \mathcal{M}_{pr}(\mathcal{P}_f(\mathsf{At}))$, as follows. Given $A \in \mathsf{At}$,

$$
\begin{aligned}
p(A)\colon \quad & \mathcal{P}_f(\mathsf{At}) && \to \quad [0,1] \\
& \{B_1, \dots, B_n\} && \mapsto \begin{cases} r & \text{if } r :: A \leftarrow B_1, \dots, B_n \text{ is a clause in } \mathbb{P} \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

Or, equivalently, $p(A) \coloneqq \sum_{(r::A \leftarrow B_1,\dots,B_n)\,\in\,\mathbb{P}} r\{B_1, \dots, B_n\}$.

▶ **Example 3.** Consider program $\mathbb{P}^{al}$ from Example 1. The set of ground atoms At_{al} is $\{\mathsf{Alarm}, \mathsf{Earthquake}, \mathsf{Burgary}, \mathsf{Wake(M)}, \mathsf{Paracusia(M)}, \mathsf{Hear\_alarm(M)}\}$. Here are some values of the corresponding coalgebra $p_{al}\colon \mathsf{At}_{al} \to \mathcal{M}_{pr}\mathcal{P}_f\mathsf{At}_{al}$:

$$p_{al}(\mathsf{Hear\_alarm(M)}) = 0.8\{\mathsf{Alarm}, \mathsf{Wake(M)}\} + 0.3\{\mathsf{Paracusia(M)}\} \quad p_{al}(\mathsf{Earthquake}) \quad = 0.01\{\}$$

▶ Remark 4. One might wonder why not simply adopt $\mathcal{P}_f(\mathcal{P}_f(-) \times [0,1])$ as the coalgebra type for PLP. Note that this encoding would not have $1-1$ correspondence with ground PLP programs: a clause $\mathcal{C} \in \mathcal{P}_f(\mathsf{At})$ may be associated with different probabilities in $[0,1]$, which violates the standard definition of PLP programs.

3.2 Derivation Semantics

In this section we are going to construct the final $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-)$-coalgebra, thus providing a semantic interpretation for probabilistic logic programs based on At.

Before the technical developments, we give an intuitive view on the semantics that the final coalgebra is going to provide. We shall represent each goal as a *stochastic derivation tree* in the final coalgebra. These trees are the probabilistic version of and-or derivation trees, which represent parallel SLD-resolutions in pure logic programming [10].

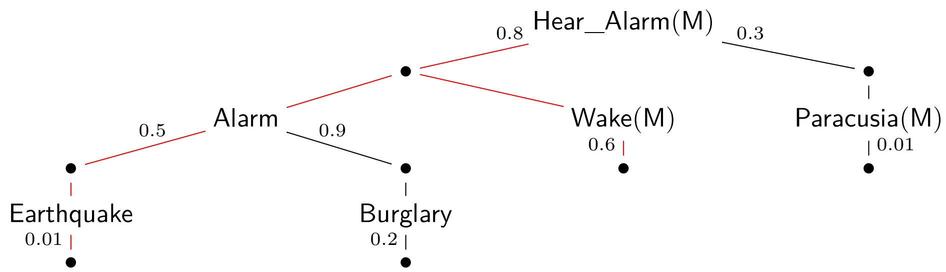
▶ **Definition 5** (Stochastic derivation trees). *Given a ground* PLP *program $\mathbb{P}$ and a ground atom A, the* stochastic derivation tree *for A in $\mathbb{P}$ is the possibly infinite tree $\mathcal{T}$ such that:*

1. *Every node is either an atom-node (labelled with an atom $A' \in \mathsf{At}$) or a clause-node (labelled with $\bullet$). They appear alternatingly in depth, in this order. The root is an atom-node labelled with A.*

2. *Each edge from an atom-node to its (clause-)children is labelled with a probability value.*
3. *Suppose s is an atom-node with label A'. Then for every clause $r :: A' \leftarrow B_1, \dots, B_k$ in $\mathbb{P}$, s has exactly one child t, the edge $s \to t$ is labelled with r, and t has exactly k children labelled with $B_1, \dots, B_k$, respectively.*

The final coalgebra semantics $[\![-]\!]_p$ for a program $\mathbb{P}$ will map a goal A to the stochastic derivation tree representing all possible SLD-resolutions of A in $\mathbb{P}$.

▶ **Example 6.** Continuing Example 1, $[\![\mathsf{Hear\_alarm(M)}]\!]_{p_{al}}$ is the stochastic derivation tree below. The subtree highlighted in red represents one of the successful refutations of Hear_alarm(M) in p_{al}: indeed, note that a single child is selected for each atom-node A (corresponding to a clause matching A), all children of any clause-node are selected (corresponding to the atoms in the body of the clause), and the subtree has clause-nodes as leaves (all atoms are proven).



(2)

Any such subtree describes a refutation, but does not yield a probability value to be associated to a goal— this is the remit of the distribution semantics, see Example 10 below.

In the remaining part of the section, we construct the cofree coalgebra for $\mathcal{M}_{pr}\mathcal{P}_f$ via a so-called terminal sequence [28], and obtain $[\![-]\!]_p$ from the resulting universal property. We report the steps of the terminal sequence as they are instrumental in showing that the elements of the cofree coalgebra can be seen as stochastic derivation trees.

▶ **Construction 7.** The terminal sequence for the functor $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-) : \mathbf{Sets} \to \mathbf{Sets}$ consists of sequences of objects $\{X_\alpha\}_{\alpha \in \mathbf{Ord}}$ and arrows $\{\delta^\alpha_\beta \colon X_\alpha \to X_\beta\}_{\beta < \alpha \in \mathbf{Ord}}$ constructed by the following inductive definitions:

$$X_\alpha := \begin{cases} \mathsf{At} & \alpha = 0 \\ \mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\xi) & \alpha = \xi + 1 \\ \lim\{\delta^\chi_\xi \mid \xi < \chi < \alpha\} & \alpha \text{ is limit} \end{cases} \qquad \delta^\alpha_\beta := \begin{cases} \pi_1 & \alpha = 1, \beta = 0 \\ \mathsf{id}_{\mathsf{At}} \times \mathcal{M}_{pr}\mathcal{P}_f(\delta^{\xi+1}_\xi) & \alpha = \beta + 1 = \xi + 2 \\ \text{the limit projections} & \alpha \text{ is limit}, \beta < \alpha \\ \text{universal map to } X_\beta & \beta \text{ is limit}, \alpha = \beta + 1 \end{cases}$$

▶ **Proposition 8.** *The terminal sequence for the functor $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-)$ converges to a limit X_γ such that $X_\gamma \cong X_{\gamma+1}$.*

Proof. We need to verify the assumptions of [28, Corollary 3.3]. It is well-known that $\mathcal{P}_f$ is ω-accessible, and $\mathcal{M}_{pr}$ has the same property, see e.g. [26, Prop. 6.1.2]. Because accessibility is defined in terms of colimit preservation, it is clearly preserved by composition, and thus $\mathcal{M}_{pr}\mathcal{P}_f$ is also accessible. It remains to check that it preserves monics. For $\mathcal{M}_{pr}$, given any monomorphism $i : C \to D$ in **Sets**, suppose $\mathcal{M}_{pr}(i)(\varphi) = \mathcal{M}_{pr}(i)(\varphi')$ for some $\varphi, \varphi' \in \mathcal{M}_{pr}(C)$. Then for any $d \in D$, $\mathcal{M}_{pr}(i)(\varphi)(d) = \mathcal{M}_{pr}(i)(\varphi')(d)$. If we focus on the image $i[C]$, then there is an inverse function $i^{-1} : i[C] \to C$, and $\mathcal{M}_{pr}(i)(\varphi) = \mathcal{M}_{pr}(i)(\varphi')$

implies that $\varphi(i^{-1}(d)) = \varphi'(i^{-1}(d))$ for any $d \in i[C]$. But this simply means that $\varphi = \varphi'$. As the same is true for $\mathcal{P}_f$ and the property is preserved by composition, we have that $\mathcal{M}_{pr}\mathcal{P}_f$ preserves monics. We can then conclude by [28, Corollary 3.3] that the terminal sequence for $At \times \mathcal{M}_{pr}\mathcal{P}_f$ converges to the cofree $\mathcal{M}_{pr}\mathcal{P}_f$-coalgebra on At. ◀

Note that $X_{\gamma+1}$ is defined as $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$, and the above isomorphism makes $X_\gamma \to \mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$ the final $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f$-coalgebra— or, in other words, *cofree $\mathcal{M}_{pr}\mathcal{P}_f$-coalgebra* on At. As for the tree representation of the elements of X_γ, recall that elements of the cofree $\mathcal{P}_f\mathcal{P}_f$-coalgebra on At can be seen as and-or trees [16]. By replacing the first $\mathcal{P}_f$ with $\mathcal{M}_{pr}$, effectively one adds probability values to the edges from and-nodes to or-nodes (which are edges from atom-nodes to or-nodes in our stochastic derivation trees), as in (2). Thus stochastic derivation trees as in Definition 5 are elements of X_γ. The action of the coalgebra map $\cong: X_\gamma \to \mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$ is best seen with an example: the tree $\mathcal{T}$ in (2) (an element of X_γ) is mapped to the pair $\langle \mathsf{Hear\_alarm(M)}, \varphi\rangle$, where φ is the function $\mathcal{P}_f(X_\gamma) \to [0,1]$ assigning 0.8 to the set consisting of the subtrees of $\mathcal{T}$ with root Alarm and with root Wake(M), 0.3 to the singleton consisting to the subtree of $\mathcal{T}$ with root Paracusia(M), and 0 to any other finite set of trees.

With all the definitions at hand, it is straightforward to check that $[\![-]\!]_p$ mapping $A \in \mathsf{At}$ to its stochastic derivation tree in p makes the following diagram commute

$$\begin{array}{ccc} \mathsf{At} & \xrightarrow{\ [\![-]\!]_p\ } & X_\gamma \\ \downarrow{\scriptstyle <id,p>} & & \downarrow{\scriptstyle \cong} \\ \mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(\mathsf{At}) & \xrightarrow{\ id \times \mathcal{M}_{pr}\mathcal{P}_f([\![-]\!]_p)\ } & \mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma) \end{array}$$

and thus by uniqueness it coincides with the $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f$-coalgebra map provided by the universal property of the final $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f$-coalgebra $X_\gamma \to \mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(X_\gamma)$.

3.3 Distribution Semantics

This section gives a coalgebraic definition of the usual *distribution semantics* of probabilistic logic programming. As in the previous section, before the technical developments we gather some preliminary intuition. Recall from Section 2 that the core of the distribution semantics is the probability distribution over the sub-programs (sets of clauses) of a given program $\mathbb{P}$. These sub-programs are also called (possible) worlds, and the distribution semantics of a goal is the sum of the probabilities of all the worlds in which it is refutable.

In order to code this information as elements of a final coalgebra, we need to present it in tree-shape. Roughly speaking, we form a distribution over the sub-programs along the execution tree. This justifies the following notion of *distribution trees.*

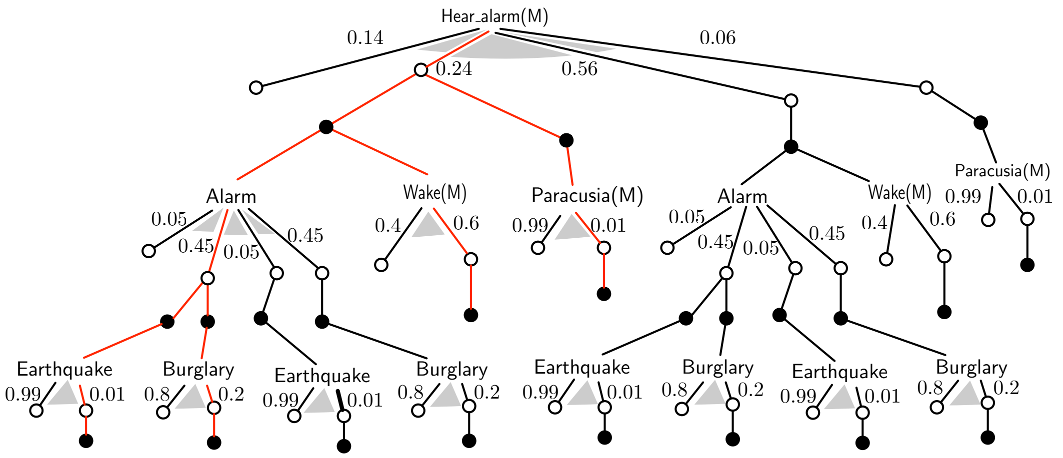
▶ **Definition 9** (Distribution trees)**.** *Given a* PLP *program $\mathbb{P}$ and an atom A, the* distribution tree *for A in $\mathbb{P}$ is the possibly infinite tree $\mathcal{T}$ satisfying the following properties:*

1. *Every node is exactly one of the three kinds: atom-node (labelled with an atom $A \in \mathsf{At}$), world-node (labelled with $\circ$), clause-node (labelled with $\bullet$). They appear alternatingly in this order in depth. The root is an atom-node labelled with A.*
2. *Every edge from an atom-node to its (world-)children is labelled with a probability value, and they sum up to one.*
3. *Suppose s is an atom-node labelled with A', and $C = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ is the set of all the clauses in $\mathbb{P}$ whose head is A'. Then s has 2^m children, each standing for a subset X of C. If a child t stands for X, then the edge $s \to t$ is labelled with probability*

$\prod_{\mathcal{C} \in X} \mathsf{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in C \setminus X}(1 - \mathsf{Label}(\mathcal{C}'))$— *recall that* $\mathsf{Label}(\mathcal{C})$ *is the probability labelling* $\mathcal{C}$*. Also, t has exactly $|X|$ children, each standing for a clause $\mathcal{C} \in X$. If a child u stands for $\mathcal{C} = r :: A' \leftarrow B_1, \ldots, B_k$, then u has k children, labelled with $B_1, \ldots, B_k$ respectively.*

Comparing distribution trees with stochastic derivation trees (Definition 5) , one can observe the addition of another class of nodes, representing possible worlds. Moreover, the possible worlds associated with an atom-node (a goal) must form a probability distribution— as opposed to stochastic derivation trees, in which probabilities labelling parallel edges do not need to share any relationship. An example of the distribution tree associated with a goal is provided in the continuation of our leading example (Examples 1 and 6).

▶ **Example 10.** In the context of Example 1, the distribution tree of $\mathsf{Hear\_alarm(M)}$ is depicted below, where we use grey shades to emphasise sets of edges expressing a probability distribution. Also, note the ∘s with no children, standing for empty worlds.



In the literature, the distribution semantics usually associates with a goal a single probability value (1), rather than a whole tree. However, given the distribution tree it is straightforward to compute such probability. The subtree highlighted in red above describes a refutation of $\mathsf{Hear\_alarm(M)}$ with probability 0.000001296 (= the product of all the probabilities in the subtree). The sum of all the probabilities associated to such "refutation" subtrees yields the usual distribution semantics (1)— the computation is shown in detail in Appendix A.

In the remainder of this section, we focus on the coalgebraic characterisation of distribution trees and the associated semantics map. Our strategy will be to introduce a novel coalgebra type $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$, such that distribution trees can be seen as elements of the cofree coalgebra. Then, we will provide a natural transformation $\mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f$, which can be used to transforms stochastic derivation trees into distribution trees. Finally, composing the universal properties of these cofree coalgebras will yield the desired distribution semantics.

We begin with the definition of $\mathcal{D}_{\leq 1}\mathcal{P}_f$. This is simply the composite $\mathcal{D}_{\leq 1}\mathcal{P}_f \colon \mathbf{Sets} \to \mathbf{Sets}$, where $\mathcal{D}_{\leq 1}$ is the *sub-probability distribution* functor. Recall that $\mathcal{D}_{\leq 1}$ maps X to the set of sub-probability distributions with finite supports on X (i.e., convex combinations of elements of X whose sum is less or equal to 1), and acts component-wise on functions.

▶ Remark 11. Note that we cannot work with full probabilities here, since a goal may not match any clause. In such a case there is no world in which the goal is refutable and its probability in the program is 0.

The next step is to recover distribution trees as the elements of the $\mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$-cofree coalgebra on At. This goes via a terminal sequence, similarly to the case of $\mathcal{M}_{pr}\mathcal{P}_f$ in the previous section. The terminal sequence for $\mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f(-) : \mathbf{Sets} \to \mathbf{Sets}$ is constructed as the one for $\mathsf{At} \times \mathcal{M}_{pr}\mathcal{P}_f(-) : \mathbf{Sets} \to \mathbf{Sets}$ (Construction 7), with $\mathcal{D}_{\leq 1}\mathcal{P}_f$ replacing $\mathcal{M}_{pr}$.

▶ **Proposition 12.** *The terminal sequence of* $\mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f(-)$ *converges at some limit ordinal* χ, *and* $(\lambda_\chi^{\chi+1})^{-1} : Y_\chi \to \mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f Y_\chi$ *is the final* $\mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$ *coalgebra.*

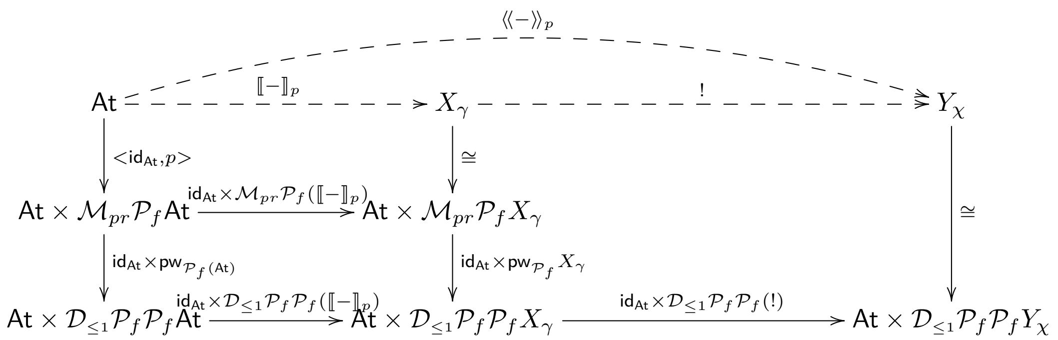
Proof. As for Proposition 8, by [28, Cor. 3.3] it suffices to show that $\mathcal{D}_{\leq 1}\mathcal{P}_f$ is accessible and preserves monos. Both are simple exercises; in particular, see [1] for accessibility of $\mathcal{D}_{\leq 1}$. ◀

The association of distribution trees with elements of Y_χ is suggested by the type $\mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$. Indeed, $\mathsf{At} \times \mathcal{D}_{\leq 1}$ is the layer of atom-nodes, labelled with elements of At and with outgoing edges forming a sub-probability distribution; the first $\mathcal{P}_f$ is the layer of world-nodes; the second $\mathcal{P}_f$ is the layer of clause-nodes. The coalgebra map $Y_\chi \to \mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f Y_\chi$ associates a goal to subtrees of its distribution trees, analogously to the coalgebra structure on stochastic derivation trees in the previous section.

The last ingredient we need is a translation of stochastic derivation trees into distribution trees. We formalise this as a natural transformation $\mathsf{pw} : \mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f$. The naturality of pw can be checked with a simple calculation.

▶ **Definition 13.** *The "**possible worlds**" natural transformation* $\mathsf{pw}\colon \mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f$ *is defined by* $\mathsf{pw}_X : \varphi \mapsto \sum_{Y \subseteq \mathsf{supp}(\varphi)} r_Y Y$, *where each* $r_Y = \prod_{y \in Y} \varphi(y) \cdot \prod_{y' \in \mathsf{supp}(\varphi) \setminus Y}(1 - \varphi(y'))$. *In particular, when* $\mathsf{supp}(\varphi)$ *is empty,* $\mathsf{pw}_X(\varphi) = 0$.

Now we have all the ingredients to characterise the distribution semantics coalgebraically, as the morphism $\langle\!\langle - \rangle\!\rangle_p \colon \mathsf{At} \to Y_\chi$ defined by the following diagram, which maps $A \in \mathsf{At}$ to its distribution tree in p.



(3)

Note the use of pw to extend probabilistic logic programs and stochastic derivation trees to the same coalgebra type as distribution trees. Then the distribution semantics $\langle\!\langle - \rangle\!\rangle_p$ is uniquely defined by the universal property of the final $\mathsf{At} \times \mathcal{D}_{\leq 1}\mathcal{P}_f\mathcal{P}_f$-coalgebra. By uniqueness, it can also be computed as the composite $! \circ [\![-]\!]_p$, that is, first one derives the semantics $[\![-]\!]_p$, then applies the translation pw to each level of the resulting stochastic derivation tree, in order to turn it into a distribution tree.

4 General Case

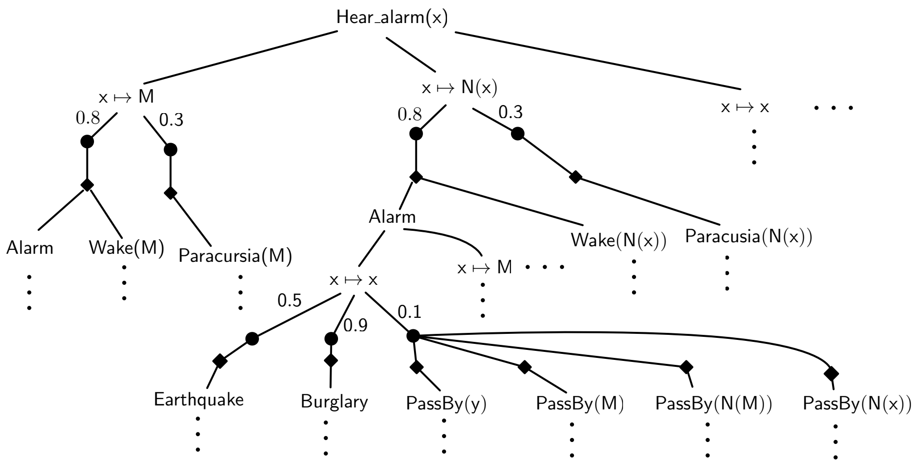
We now generalise our coalgebraic treatment to arbitrary probabilistic logic programs and goals, possibly including variables. The section has the same structure as the one devoted to the ground case. First, in Subsection 4.2, we give a coalgebraic representation for general PLP, and equip it with a final coalgebra semantics in terms of stochastic derivation trees. Next, in Subsection 4.3, we study the coalgebraic representation of the distribution semantics. We begin by introducing our leading example— an extension of Example 1.

▶ **Example 14.** We tweak the ground program of Example 1. Now it is not just Mary that may hear the alarm, but also her neighbours. There is a small probability that the alarm rings because someone passes too close to Mary's house. However, we can only estimate the possibility of paracusia and being awake for Mary, not the neighbours. The revised program, which by abuse of notation we also call $\mathbb{P}^{al}$, is based on an extension of the language in Example 1: we add a new 1-ary function symbol Neigh^1 to the signature Σ_{al}, and a new 1-ary predicate $\mathsf{PassBy}(-)$ to the alphabet. Note the appearance of a variable x.

$0.01 ::$	Earthquake	$\leftarrow$	$0.5 ::$	Alarm	$\leftarrow$ Earthquake
$0.2 ::$	Burglary	$\leftarrow$	$0.9 ::$	Alarm	$\leftarrow$ Burglary
$0.6 ::$	Wake(Mary)	$\leftarrow$	$0.1 ::$	Alarm	$\leftarrow$ PassBy(x)
$0.01 ::$	Paracusia(Mary)	$\leftarrow$	$0.3 ::$	Hear_alarm(x)	$\leftarrow$ Paracusia(x)
$0.8 ::$	Wake(Neigh(x))	$\leftarrow$ Wake(x)	$0.8 ::$	Hear_alarm(x)	$\leftarrow$ Alarm, Wake(x)

As we want to maintain our approach a direct generalisation of the coalgebraic semantics [3] of pure logic programs, the derivation semantics $[\![-]\!]$ for PLP will represent resolution by *unification*. This means that, at each step of the computation, given a goal A, one seeks substitutions θ, τ such that $A\theta = H\tau$ for some head H of a clause in the program. As a roadmap, we anticipate the way this computation is represented in terms of stochastic derivation trees (Definition 20 below), with a continuation of our leading example.

▶ **Example 15.** In the context of Example 14, the tree for $[\![\mathsf{Hear\_alarm(x)}]\!]_{\mathbb{P}^{al}}$ is (partially) depicted below. Compared to the ground case (Example 6), now substitutions applied on the goal side appear explicitly as labels. We abbreviate Neigh as N and Mary as M.



(4)

Resolution by unification as above will be implemented in two stages. The first step is devising a map for term-matching. Assuming that the substitution instance $A\theta$ of a goal A is already given, we define p performing term-matching of $A\theta$ in a given program $\mathbb{P}$:

$$p(A\theta)\colon \quad \{B_1\tau_i, \dots, B_k\tau_i\}_{i \in I \subseteq \mathbb{N}} \quad \mapsto \begin{cases} r & (r :: H \leftarrow B_1, \dots, B_k) \in \mathbb{P} \text{ and} \\ & \quad I \text{ contains all } i \text{ s.t. } A\theta = H\tau_i \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Intuitively, one application of such map is represented in a tree structure as Example 15 by the first two layers of the subtree rooted at θ. The reason why the domain of $p(A)$ is a *countable* set $\{B_1\tau_i, \dots, B_k\tau_i\}_{i\in I\subseteq\mathbb{N}}$ of instances of the same body $B_1, \dots, B_k$ is that the same clause may match a goal with countably many different substitutions τ_i. For example in the bottom part of (4) there are countably infinite substitutions τ_i matching the head of $\mathsf{Alarm} \leftarrow \mathsf{PassBy}(\mathsf{x})$ to the goal Alarm, substituting x with $\mathsf{Mary}, \mathsf{Neigh}(\mathsf{Mary}), \mathsf{Neigh}(\mathsf{x}), \dots$. This will be reflected in the coalgebraic representation of PLP (see (7) below) by the use of the countable powset functor $\mathcal{P}_c$.

In order to model arbitrary unification, the second step is considering all substitutions θ on the goal A such that a term-matcher for $A\theta$ exists. There is an elegant categorical construction [3] packing together these two steps into a single coalgebra map. We will present it in subsection 4.1, and then use it to present the derivation semantics anticipated by Example 15 (Section 4.2). Finally we will give a coalgebraic view on the distribution semantics for PLP (Section 4.3).

4.1 Coalgebraic Representation of PLP

Towards a categorification of general PLP, the first concern is to account for the presence of variables in atoms. This is standardly done by letting the space of atoms on an alphabet $\mathcal{A}$ be a presheaf $\mathsf{At}\colon \mathbf{L}_\Sigma^{\mathrm{op}} \to \mathbf{Sets}$ rather than a set. Here the index category $\mathbf{L}_\Sigma^{\mathrm{op}}$ is the opposite *Lawvere Theory* of Σ (see Section 2). For each $n \in \mathsf{Ob}(\mathbf{L}_\Sigma^{\mathrm{op}})$, $\mathsf{At}(n)$ is defined as the set of $\mathcal{A}$-atoms in context n. Given a n-tuple $\theta = \langle t_1, \dots, t_n\rangle \in \mathbf{L}_\Sigma^{\mathrm{op}}[n, m]$ of Σ-terms in context m, $\mathsf{At}(\theta)\colon \mathsf{At}(n) \to \mathsf{At}(m)$ is defined by substitution, namely $\mathsf{At}(\theta)(A) = A\theta$, for any $A \in \mathsf{At}(n)$.

As observed in [17] for pure logic programs, if we naively try to model our specification (5) for p as a coalgebra on At, we run into problems: indeed p is not a natural transformation, thus not a morphism between presheaves. Intuitively, this is because the existence of a term-matching for a goal A does not necessarily imply the existence of a term-matching for its substitution instance $A\sigma$. For pure logic programs, this problem can be solved in at least two ways. First, [17] relaxes naturality by changing the base category of presheaves from **Sets** to **Poset**. We take here the second route, namely give a 'saturated' coalgebraic treatment of PLP, generalising the modelling of pure logic programs proposed in [3]. This approach has the advantage of letting us work with **Sets**-based presheaves, and be still able to recover term-matching via a 'desaturation' operation— see [3] and Appendix B.

The Saturation Adjunction. To this aim, we briefly recall the saturated approach from [3]. The central piece is the adjunction $\mathcal{U} \dashv \mathcal{K}$ on presheaf categories, as on the left below.

$$\mathbf{Sets}^{\mathbf{L}_\Sigma^{\mathrm{op}}} \underset{\mathcal{K}}{\overset{\mathcal{U}}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{Sets}^{|\mathbf{L}_\Sigma^{\mathrm{op}}|} \qquad\qquad \begin{array}{ccc} |\mathbf{L}_\Sigma^{\mathrm{op}}| & \overset{\iota}{\hookrightarrow} & \mathbf{L}_\Sigma^{\mathrm{op}} \\ {\scriptstyle \mathsf{F}}\downarrow & \swarrow {\scriptstyle \mathcal{K}(\mathsf{F})} & \\ \mathbf{Sets} & & \end{array} \tag{6}$$

Here $|\mathbf{L}_\Sigma^{\mathrm{op}}|$ is the discretisation of $\mathbf{L}_\Sigma^{\mathrm{op}}$, i.e. all the arrows but the identities are dropped. The left adjoint $\mathcal{U}$ is the forgetful functor, given by precomposition with the obvious inclusion $\iota\colon |\mathbf{L}_\Sigma^{\mathrm{op}}| \to \mathbf{L}_\Sigma^{\mathrm{op}}$. $\mathcal{U}$ has a right adjoint $\mathcal{K} = \mathsf{Ran}\iota\colon \mathbf{Sets}^{|\mathbf{L}_\Sigma^{\mathrm{op}}|} \to \mathbf{Sets}^{\mathbf{L}_\Sigma^{\mathrm{op}}}$, which sends every presheaf $\mathsf{F}\colon |\mathbf{L}_\Sigma^{\mathrm{op}}| \to \mathbf{Sets}$ to its *right Kan extension* along ι, as in the rightmost diagram in (6). The definition of $\mathcal{K}$ can be computed [21] as follows:

- on objects $\mathsf{F} \in \mathsf{Ob}(\mathbf{Sets}^{\mathbf{L}_\Sigma^{\mathrm{op}}})$, the presheaf $\mathcal{K}(\mathsf{F})\colon \mathbf{L}_\Sigma^{\mathrm{op}} \to \mathbf{Sets}$ is defined by letting $\mathcal{K}(\mathsf{F})(n)$ be the product $\mathcal{K}(\mathsf{F})(n) = \prod_{\theta\in\mathbf{L}_\Sigma^{\mathrm{op}}[n,m]} \mathsf{F}(m)$, where m ranges over $\mathsf{Ob}(\mathbf{L}_\Sigma^{\mathrm{op}})$. Intuitively, every element in $\mathcal{K}(\mathsf{F})(n)$ is a tuple with index set $\bigcup_{m\in\mathsf{Ob}(\mathbf{L}_\Sigma^{\mathrm{op}})} \mathbf{L}_\Sigma^{\mathrm{op}}[n, m]$, and its component

at index $\theta\colon n \to m$ is an element in $\mathsf{F}(m)$. We follow the convention of [3] and write $\dot{x}, \dot{y}, \ldots$ for such tuples, and $\dot{x}(\theta)$ for the component of $\dot{x}$ at index θ.
With this convention, given an arrow $\sigma \in \mathbf{L}_\Sigma^{\mathrm{op}}[n, n']$, $\mathcal{K}(\mathsf{F})(\sigma)$ is defined by pointwise substitution as the mapping of the tuple $\dot{x}$ to the tuple $\langle \dot{x}(\theta \circ \sigma)\rangle_{\theta\colon n' \to m}$.

- On arrows, given a morphism $\alpha\colon \mathsf{F} \to \mathsf{G}$ in $\mathbf{Sets}^{|\mathbf{L}_\Sigma^{\mathrm{op}}|}$, $\mathcal{K}(\alpha)$ is a natural transformation $\mathcal{K}(\mathsf{F}) \to \mathcal{K}(\mathsf{G})$ defined pointwisely as $\mathcal{K}(\alpha)(n)\colon \dot{x} \mapsto \langle \alpha_m(\dot{x}(\theta))\rangle_{\theta\colon n \to m}$.

It is also useful to record the unit $\eta\colon 1 \to \mathcal{K}\mathcal{U}$ of the adjunction $\mathcal{U} \dashv \mathcal{K}$. Given a presheaf $\mathsf{F}\colon \mathbf{L}_\Sigma^{\mathrm{op}} \to \mathbf{Sets}$, $\eta_{\mathsf{F}}\colon \mathsf{F} \to \mathcal{K}\mathcal{U}\mathsf{F}$ is a natural transformation defined by $\eta_{\mathsf{F}}(n)\colon x \mapsto \langle \mathsf{F}(\theta)(x)\rangle_{\theta\colon n \to m}$.

Saturation in PLP. We now come back to the question of the coalgebra structure on the presheaf At modelling PLP. First, we are now able to represent p in (5) as a coalgebra map. The aforementioned naturality issue is solved by defining it as a morphism in $\mathbf{Sets}^{|\mathbf{L}_\Sigma^{\mathrm{op}}|}$ rather than in $\mathbf{Sets}^{\mathbf{L}_\Sigma^{\mathrm{op}}}$, thus making naturality trivial. The coalgebra p will have the following type

$$p\colon \mathcal{U}\mathsf{At} \to \widehat{\mathcal{M}_{pr}}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}\mathsf{At} \tag{7}$$

where $\widehat{(\cdot)}$ is the obvious extension of $\mathbf{Sets}$-endofunctors to $\mathbf{Sets}^{|\mathbf{L}_\Sigma^{\mathrm{op}}|}$-endofunctors, defined by functor precomposition. With respect to the ground case, note the insertion of $\widehat{\mathcal{P}_c}$, the lifting of the *countable* powerset functor, in order to account for the countably many instances of a clause that may match the given goal (*cf.* the discussion below (5)).

▶ **Example 16.** Our program $\mathbb{P}^{al}$ (Example 14) is based on $\mathsf{At}_{al}\colon \mathbf{L}_{\Sigma_{\mathsf{al}}}^{\mathrm{op}} \to \mathbf{Sets}$. Some of its values are $\mathsf{At}_{al}(0) = \{\mathsf{Mary}, \mathsf{Neigh}(\mathsf{Mary}), \mathsf{Neigh}(\mathsf{Neigh}(\mathsf{Mary})), \ldots\}$ and $\mathsf{At}_{al}(1) = \{\mathsf{x}, \mathsf{Mary}, \mathsf{Neigh}(\mathsf{x}), \mathsf{Neigh}(\mathsf{Mary}), \ldots\}$. Part of the coalgebra p_{al} modelling the program $\mathbb{P}^{al}$ is as follows (*cf.* the tree (4)).

$$\begin{aligned}(p_{al})_0(\mathsf{Hear\_alarm}(\mathsf{Mary})) &= 0.8\{\{\mathsf{Alarm}, \mathsf{Wake}(\mathsf{Mary})\}\} + 0.3\{\{\mathsf{Parasusia}(\mathsf{Mary})\}\}\\ (p_{al})_1(\mathsf{Alarm}) &= 0.5\{\{\mathsf{Earthquake}\}\} + 0.9\{\{\mathsf{Burglary}\}\}\\ &\quad + 0.1\{\{\mathsf{PassBy}(\mathsf{Mary})\}, \{\mathsf{PassBy}(\mathsf{Neigh}(\mathsf{Mary}))\}, \{\mathsf{PassBy}(\mathsf{Neigh}(\mathsf{x}))\}, \ldots\}\end{aligned}$$

The universal property of the adjunction (6) gives a canonical 'lifting' of p to a $\mathcal{K}\widehat{\mathcal{M}_{pr}}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}$-coalgebra $p^\sharp$ on At, performing unification rather than just term-matching:

$$p^\sharp := \mathsf{At} \xrightarrow{\eta_{\mathsf{At}}} \mathcal{K}\mathcal{U}\mathsf{At} \xrightarrow{\mathcal{K}p} \mathcal{K}\widehat{\mathcal{M}_{pr}}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}\mathsf{At} \tag{8}$$

where η is the unit of the adjunction, as defined above. Spelling it out, $p^\sharp$ is the mapping

$$p_n^\sharp \ : \ A \in \mathsf{At}(n) \mapsto \langle p_m(A\theta)\rangle_{\theta\colon n \to m} .$$

Intuitively, $p_n^\sharp$ retrieves all the unifiers $\langle \theta, \tau\rangle$ of A and head H in $\mathbb{P}$: first, we have $A\theta \in \mathsf{At}(m)$ as a component of the saturation of A by η_{At}; then we term-match H with $A\theta$ by $\mathcal{K}p_m$.

▶ Remark 17. Note that the parameter $n \in \mathsf{Ob}(\mathbf{L}_\Sigma^{\mathrm{op}})$ in the natural transformation $p^\sharp$ fixes the pool $\{x_1, \ldots, x_n\}$ of variables appearing in the atoms (and relative substitutions) that are considered in the computation. Analogously to the case of pure logic programs [17, 3], it is intended that such n can always be chosen "big enough" so that all the relevant substitution instances of the current goal and clauses in the program are covered—note the variables occurring therein always form a *finite* set, included in $\{x_1, \ldots, x_m\}$ for some $m \in \mathbb{N}$.

4.2 Derivation Semantics

Once we have identified our coalgebra type, the construction leading to the derivation semantics $[\![-]\!]_{p^\sharp}$ for general PLP is completely analogous to the ground case. One can define the cofree coalgebra for $\mathcal{K}\widehat{\mathcal{M}_{pr}}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(-)$ by terminal sequence, similarly to Construction 7. For simplicity, henceforth we denote the functor $\mathcal{K}\widehat{\mathcal{M}_{pr}}\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(-)$ by $\mathcal{S}$.

▶ **Construction 18.** The terminal sequence for $\mathsf{At} \times \mathcal{S}(-) : \mathbf{Sets}^{\mathbf{L}_\Sigma^{op}} \to \mathbf{Sets}^{\mathbf{L}_\Sigma^{op}}$ consists of a sequence of objects X_α and morphisms $\delta^\beta_\alpha \colon X_\beta \to X_\alpha$, for $\alpha < \beta \in \mathbf{Ord}$, defined analogously to Construction 7, with $p^\sharp$ and $\mathcal{S}$ replacing p and $\mathcal{M}_{pr}\mathcal{P}_f$.

This terminal sequence converges by the following lemma.

▶ **Proposition 19.** *$\mathcal{S}$ is accessible, and preserves monomorphisms.*

Proof. Since both properties are preserved by composition, it suffices to show that they hold for all the component functors. For $\widehat{\mathcal{M}_{pr}}$, $\widehat{\mathcal{P}_c}$ and $\widehat{\mathcal{P}_f}$, they follow from accessibility and mono-preservation of $\mathcal{M}_{pr}$, $\mathcal{P}_c$ and $\mathcal{P}_f$ (see Proposition 8), as (co)limits in presheaf categories are computed pointwise. For $\mathcal{K}$ and $\mathcal{U}$, these properties are proven in [3]. ◀

Therefore the terminal sequence for $\mathsf{At} \times \mathcal{S}(-)$ converges at some limit ordinal, say γ, yielding the final $\mathsf{At} \times \mathcal{S}(-)$-coalgebra $X_\gamma \xrightarrow{\cong} \mathsf{At} \times \mathcal{S}(X_\gamma)$. The derivation semantics is then defined $[\![-]\!]_{p^\sharp} : \mathsf{At} \to X_\gamma$ by universal property, as on the right.

$$\begin{array}{ccc} \mathsf{At} & \xrightarrow{[\![-]\!]_{p^\sharp}} & X_\gamma \\ {\scriptstyle \langle \mathsf{id}_{\mathsf{At}}, p^\sharp\rangle}\downarrow & & \downarrow{\scriptstyle \cong} \\ \mathsf{At} \times \mathcal{S}(\mathsf{At}) & \xrightarrow[\mathsf{id}_{\mathsf{At}} \times [\![-]\!]_{p^\sharp}]{} & \mathsf{At} \times \mathcal{S}(X_\gamma) \end{array} \qquad (9)$$

A careful inspection of the terminal sequence constructing X_γ allows to infer a representation of its elements as trees, among which we have those representing computations by unification of goals in a PLP program. We call these *stochastic saturated derivation trees*, as they extend the derivation trees of Definition 5 and are the probabilistic variant of saturated and-or trees in [3]. Using (9) one can easily verify that $[\![A]\!]$ is indeed the stochastic saturated derivation tree for a given goal A. Example 15 provides a pictorial representation of one such tree.

▶ **Definition 20** (Stochastic saturated derivation trees). *Given a probabilistic logic program $\mathbb{P}$, a natural number n and an atom $A \in \mathsf{At}(n)$. The* stochastic saturated derivation tree *for A in $\mathbb{P}$ is the possibly infinite tree $\mathcal{T}$ satisfying the following properties:*

1. *There are four kinds of nodes: atom-node (labelled with an atom), substitution-node (labelled with a substitution), clause-node (labelled with $\bullet$), instance-node (labelled with ⧫), appearing alternatively in depth in this order. The root is an atom-node with label A.*
2. *Each clause-node is labelled with a probability value.*
3. *Suppose an atom-node s is labelled with $A' \in \mathsf{At}(n')$. For every substitution $\theta \colon n' \to m'$, s has exactly one (substitution-node) child t labelled with θ. For every clause $r :: H \leftarrow B_1, \dots, B_k$ in $\mathbb{P}$ such that H matches $A'\theta$ (via some substitution), t has exactly one (clause-)child u, and edge $t \to u$ is labelled with r. Then for every substitution τ such that $A'\theta = H\tau$ and $B_1\tau, \dots, B_k\tau \in \mathsf{At}(m')$, u has exactly one (instance-)child v. Also v has exactly $|\{B_1\tau, \dots, B_k\tau\}|$-many (atom-)children, each labelled with one element in $\{B_1\tau, \dots, B_k\tau\}$.*

4.3 Distribution Semantics

In this section we conclude by giving a coalgebraic perspective on the distribution semantics $\langle\!\langle - \rangle\!\rangle$ for general PLP. Mimicking the ground case (Section 3.3), this will be presented as an extension of the derivation semantics, via a 'possible worlds' natural transformation. Also in the general case, we want to guarantee that a single probability value is computable for a given goal A from the corresponding tree $\langle\!\langle A \rangle\!\rangle$ in the final coalgebra— whenever this probability is also computable in the 'traditional' way (see (1)) of giving distribution semantics to PLP. In this respect, the presence of variables and substitutions poses additional challenges, for which we refer to Appendices A and B. In a nutshell, the issue is that the distribution semantics counts the use of a clause in the program at most once, independently from how many times that clause is used again in the computation. To account for this aspect in our tree representation, we need to give enough information to determine which clause is used at each step of the computation, so that a second use can be easily detected. Note that neither our saturated derivation trees, nor a 'naive' extension of them to distribution trees, carry such information: what appears in there is only the instantiated heads and bodies, but in general one cannot retrieve A from a substitution θ and the instantiation $A\theta$. This is best illustrated via a simple example.

▶ **Example 21.** Consider the following program, based on the signature $\Sigma = \{a^0\}$ and two 1-ary predicates P, Q. It consists of two clauses:

$$0.5 :: \quad P(x_1) \leftarrow Q(x_1) \;\Big|\; 0.5 :: \quad P(x_1) \leftarrow Q(x_2)$$

The goal $P(a)$ matches the head of both clauses. However, given the sole information of the next goal being $Q(a)$, it is impossible to say whether the first clause has been used, instantiated with $x_1 \mapsto a$, or the second clause has been used, instantiated with $x_1 \mapsto a, x_2 \mapsto a$.

This observation motivates, as intermediate step towards the distribution semantics, the addition of labels to clause-nodes in derivation trees, in order to make explicit which clause is being applied. From the coalgebraic viewpoint, this just amounts to an extension of the type of the term-matching coalgebra:

$$\widetilde{p}\colon \mathcal{U}\mathsf{At} \to \widehat{\mathcal{M}_{pr}}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}\mathsf{At} \times (\mathcal{U}\mathsf{At} \times \mathcal{U}\widehat{\mathcal{P}_f}\mathsf{At})).$$

Note the insertion of $(-) \times (\mathcal{U}\mathsf{At} \times \mathcal{U}\widehat{\mathcal{P}_f}\mathsf{At})$, which allows us to indicate at each step the head ($\mathcal{U}\mathsf{At}$) and the body ($\mathcal{U}\widehat{\mathcal{P}_f}\mathsf{At}$) of the clause being used, its probability label being already given by $\widehat{\mathcal{M}_{pr}}$. More formally, for any n and atom $A \in \mathsf{At}(n)$, we define[1]

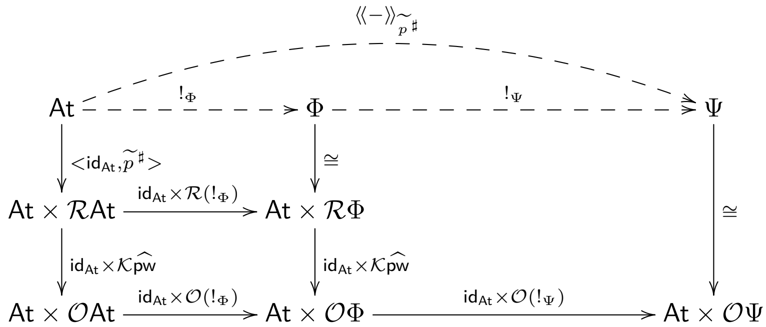
$$\widetilde{p}_n(A)\colon \langle \{B_1\tau_i, \ldots, B_k\tau_i\}_{i \in \mathcal{I} \subseteq \mathcal{N}}, \langle H, \{B_1, \ldots, B_k\}\rangle\rangle \mapsto \begin{cases} r & (r :: H \leftarrow B_1, \ldots, B_k) \in \mathbb{P}, H\tau_i = A \\ 0 & \text{otherwise} \end{cases}$$

As in the case of p in (7), we can move from term-matching to unification by using the universal property of the adjunction $\mathcal{U} \dashv \mathcal{K}$, yielding $\widetilde{p}^\sharp\colon \mathsf{At} \to \mathcal{K}\widehat{\mathcal{M}_{pr}}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}\mathsf{At} \times (\mathcal{U}\mathsf{At} \times \mathcal{U}\widehat{\mathcal{P}_f}\mathsf{At}))$. For simplicity henceforth we denote the functor $\mathcal{K}\widehat{\mathcal{M}_{pr}}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(-) \times (\mathcal{U}\mathsf{At} \times \widehat{\mathcal{P}_f}\mathcal{U}\mathsf{At}))$ by $\mathcal{R}$.

We are now able to conclude our characterisation of the distribution semantics. The 'possible worlds' transformation $\mathsf{pw}\colon \mathcal{M}_{pr} \Rightarrow \mathcal{D}_{\leq 1}\mathcal{P}_f$ (Definition 13) yields a natural transformation $\widehat{\mathsf{pw}}\colon \widehat{\mathcal{M}_{pr}} \to \widehat{\mathcal{D}_{\leq 1}}\widehat{\mathcal{P}_f}$, defined pointwise by pw. We can use $\widehat{\mathsf{pw}}$ to translate $\mathcal{R}$ into the functor $\mathcal{K}\widehat{\mathcal{D}_{\leq 1}}\widehat{\mathcal{P}_f}(\widehat{\mathcal{P}_c}\widehat{\mathcal{P}_f}\mathcal{U}(-) \times (\mathcal{U}\mathsf{At} \times \widehat{\mathcal{P}_f}\mathcal{U}\mathsf{At}))$, abbreviated as $\mathcal{O}$, which is going to give the type of saturated distribution trees for general PLP programs.

[1] As noted in Remark 17, instantiating $\widetilde{p}$ to some $n \in \mathsf{Ob}(\mathbf{L}_\Sigma^{\mathrm{op}})$ fixes a variable context $\{x_1, \ldots, x_n\}$ both for the goal and the clause labels. In practice, because the set of clauses is always finite, it suffices to chose n "big enough" so that the variables appearing in the clauses are included in $\{x_1, \ldots, x_n\}$.

As a simple extension of the developments in Section 4.2, we can construct the cofree $\mathcal{R}$-coalgebra $\Phi \xrightarrow{\cong} \mathsf{At} \times \mathcal{R}(\Phi)$ via a terminal sequence. Similarly, one can obtain the cofree $\mathcal{O}$-coalgebra $\Psi \xrightarrow{\cong} \mathsf{At} \times \mathcal{O}(\Psi)$. By the universal property of Ψ, all these ingredients get together in the definition of the distribution semantics $\langle\langle - \rangle\rangle_{\widetilde{p}^\sharp}$ for arbitrary PLP programs $\widetilde{p}^\sharp$



where $!_\Phi$ and $!_\Psi$ are given by the evident universal properties, and show the role of the cofree $\mathcal{R}$-coalgebra Φ as an intermediate step. The layered construction of final coalgebras Ψ and Φ, together with the above characterisation of $\langle\langle - \rangle\rangle_{\widetilde{p}^\sharp}$, allow to conclude that the distribution semantics for the program $\widetilde{p}^\sharp$ maps a goal A to its *saturated distribution tree* $\langle\langle A \rangle\rangle_{\widetilde{p}^\sharp}$, as formally defined below.

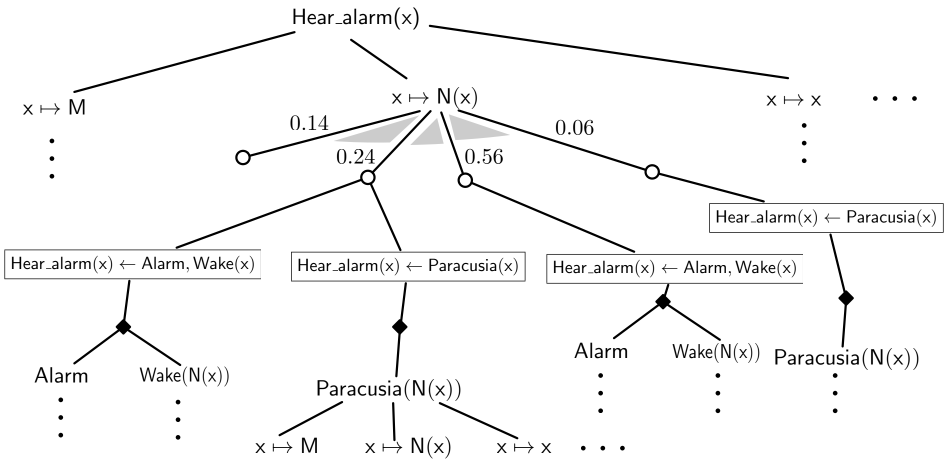
▶ **Definition 22** (Saturated distribution tree). *The* saturated distribution tree *for $A \in \mathsf{At}(n)$ in $\mathbb{P}$ is the possibly infinite $\mathcal{T}$ satisfying the following properties based on Definition 20:*

1. *There are five kinds of nodes: in addition to the atom-, substitution-, clause- and instance-nodes, there are world-nodes. The world-nodes are children of the substitution-nodes, and parents of the clause nodes. The root and the order of the rest nodes are the same as in Definition 20, condition **1**. The clause-nodes are now labelled with clauses of $\mathbb{P}$.*
2. *Suppose s is an atom node labelled with $A' \in \mathsf{At}(n')$, and t is a substitution-child of s labelled with $\theta \colon n' \to m$. Let C be the set of all clauses $\mathcal{C}$ such that $\mathsf{Head}(\mathcal{C})$ matches $A'\theta$. Then t has $2^{|C|}$ world-children, each representing a subset X of C. If a child u represents subset X, then the edge $t \to u$ has probability label $\prod_{\mathcal{C} \in X} \mathsf{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}' \in C \setminus X} \mathsf{Label}(\mathcal{C}')$. Also u has $|X|$ clause-children, one for each clause $\mathcal{C} \in X$, labelled with the corresponding clause. The rest for clause-nodes and instance-nodes are the same as in Definition 20, condition **3**.*

▶ Remark 23. Note that, in principle, saturated distribution trees could be defined coalgebraically without the intermediate step of adding clause labels. This is to be expected: coalgebra typically captures the one-step, "local" behaviour of a system. On the other hand, as explained, the need for clause labels is dictated by a computational aspect involving the depth of distribution trees, that is, a "non-local" dimension of the system.

We conclude with the pictorial representation of the saturated distribution tree of a goal in our leading example.

▶ **Example 24.** In the context of Example 14, the tree $\langle\langle \mathsf{Hear\_alarm(x)} \rangle\rangle$ capturing the distribution semantics of $\mathsf{Hear\_alarm(x)}$ is (partially) depicted as follows. Note the presence of clauses labelling the clause-nodes.



References

1 Falk Bartels, Ana Sokolova, and Erik P. de Vink. A hierarchy of probabilistic system types. *Theor. Comput. Sci.*, 327(1-2):3–22, 2004. URL: `https://doi.org/10.1016/j.tcs.2004.07.019`, `doi:10.1016/j.tcs.2004.07.019`.

2 Filippo Bonchi and Fabio Zanasi. Saturated semantics for coalgebraic logic programming. In *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, pages 80–94, 2013. URL: `https://doi.org/10.1007/978-3-642-40206-7_8`, `doi:10.1007/978-3-642-40206-7\_8`.

3 Filippo Bonchi and Fabio Zanasi. Bialgebraic semantics for logic programming. *Logical Methods in Computer Science*, 11(1), 2015. URL: `https://doi.org/10.2168/LMCS-11(1:14)2015`, `doi:10.2168/LMCS-11(1:14)2015`.

4 Fredrik Dahlqvist, Vincent Danos, Ilias Garnier, and Ohad Kammar. Bayesian inversion by ω-complete cone duality. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 1:1–1:15, 2016. URL: `https://doi.org/10.4230/LIPIcs.CONCUR.2016.1`, `doi:10.4230/LIPIcs.CONCUR.2016.1`.

5 Eugene Dantsin. Probabilistic logic programs and their semantics. In A. Voronkov, editor, *Logic Programming*, pages 152–164, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

6 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, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=1625275.1625673`.

7 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, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL: `http://dl.acm.org/citation.cfm?id=1625275.1625673`.

8 Didier Dubois, Lluas Godo, and Henri Prade. Weighted logics for artificial intelligence : an introductory discussion. *International Journal of Approximate Reasoning*, 55(9):1819 – 1829, 2014. Weighted Logics for Artificial Intelligence. URL: `http://www.sciencedirect.com/science/article/pii/S0888613X14001364`, `doi:https://doi.org/10.1016/j.ijar.2014.08.002`.

9 Gopal Gupta, Ajay Bansal, Richard Min, Luke Simon, and Ajay Mallya. Coinductive logic programming and its applications. In Véronica Dahl and Ilkka Niemelä, editors, *Logic Programming*, pages 27–44, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

10 Gopal Gupta and Vítor Santos Costa. Optimal implementation of and-or parallel prolog. *Future Generation Computer Systems*, 10(1):71 – 92, 1994. PARLE '92. URL: `http://`

www.sciencedirect.com/science/article/pii/0167739X94900523, doi:https://doi.org/10.1016/0167-739X(94)90052-3.

11 Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference by string diagram surgery. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings*, 2019. URL: http://arxiv.org/abs/1811.08338.

12 Bart Jacobs and Fabio Zanasi. The logical essentials of bayesian reasoning. In Joost-Peter Katoen Gilles Barthe and Alexandra Silva, editors, *Probabilistic Programming*. Cambridge University Press, Cambridge, 2019. URL: http://arxiv.org/abs/1804.01193.

13 Kristian Kersting and Luc De Raedt. Bayesian logic programming: Theory and tool. In *Introduction to Statistical Relational Learning*, pages 291–322. MIT Press; Cambridge, 2007. URL: https://lirias.kuleuven.be/retrieve/86539.

14 Ekaterina Komendantskaya and Yue Li. Productive corecursion in logic programming. *TPLP*, 17(5-6):906–923, 2017. URL: https://doi.org/10.1017/S147106841700028X, doi:10.1017/S147106841700028X.

15 Ekaterina Komendantskaya and Yue Li. Towards coinductive theory exploration in horn clause logic: Position paper. In *Proceedings 5th Workshop on Horn Clauses for Verification and Synthesis, HCVS 2018, Oxford, UK, 13th July 2018.*, pages 27–33, 2018. URL: https://doi.org/10.4204/EPTCS.278.5, doi:10.4204/EPTCS.278.5.

16 Ekaterina Komendantskaya, Guy McCusker, and John Power. Coalgebraic semantics for parallel derivation strategies in logic programming. In *Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers*, pages 111–127, 2010. URL: https://doi.org/10.1007/978-3-642-17796-5_7, doi:10.1007/978-3-642-17796-5_7.

17 Ekaterina Komendantskaya and John Power. Coalgebraic semantics for derivations in logic programming. In *Algebra and Coalgebra in Computer Science - 4th International Conference, CALCO 2011, Winchester, UK, August 30 - September 2, 2011. Proceedings*, pages 268–282, 2011. URL: https://doi.org/10.1007/978-3-642-22944-2_19, doi:10.1007/978-3-642-22944-2_19.

18 Ekaterina Komendantskaya and John Power. Logic programming: Laxness and saturation. *J. Log. Algebr. Meth. Program.*, 101:1–21, 2018. URL: https://doi.org/10.1016/j.jlamp.2018.07.004, doi:10.1016/j.jlamp.2018.07.004.

19 Ekaterina Komendantskaya, John Power, and Martin Schmidt. Coalgebraic logic programming: from semantics to implementation. *J. Log. Comput.*, 26(2):745–783, 2016. URL: https://doi.org/10.1093/logcom/exu026, doi:10.1093/logcom/exu026.

20 John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

21 Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971. Graduate Texts in Mathematics, Vol. 5.

22 Søren Mørk and Ian Holmes. Evaluating bacterial gene-finding hmm structures as probabilistic logic programs. *Bioinformatics*, 28(5):636–642, 2012. doi:10.1093/bioinformatics/btr698.

23 Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150 – 201, 1992. URL: http://www.sciencedirect.com/science/article/pii/089054019290061J, doi:https://doi.org/10.1016/0890-5401(92)90061-J.

24 Fabrizio Riguzzi and Terrance Swift. Probabilistic logic programming under the distribution semantics. 2014.

25 Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, pages 715–729, 1995.

26 Alexandra Silva. Kleene coalgebra. Phd thesis, CWI, Amsterdam, The Netherlands, 2010.

27 Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, Cambridge, Mass., 2005.

28 James Worrell. Terminal sequences for accessible endofunctors. *Electronic Notes in Theoretical Computer Science*, 19:24 – 38, 1999. CMCS'99, Coalgebraic Methods in Computer Science. URL: http://www.sciencedirect.com/science/article/pii/S1571066105802671, doi:https://doi.org/10.1016/S1571-0661(05)80267-1.

29 Riccardo Zese. *Probabilistic Semantic Web: Reasoning and Learning*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 2017.

A Computability of the Distribution Semantics (Ground Case)

Computing with distribution trees. As a justification for our tree representation of the distribution semantics, we claimed that the probability $\mathsf{Pr}_{\mathbb{P}}(A)$ associated with a goal (see (1)) can be straightforwardly computed from the corresponding distribution tree $\langle\!\langle A \rangle\!\rangle_p$. This appendix supplies such an algorithm. Note this serves just as a proof of concept, without any claim of efficiency compared to pre-existing implementations. In the sequel we fix a ground PLP program $\mathbb{P}$ with atoms At, a goal $A \in \mathsf{At}$ and the distribution tree $\mathcal{T}$ for A in $\mathbb{P}$ (Definition 9). First, we may assume that $\mathcal{T}$ does not contain loop (which implies that $\mathcal{T}$ is finite). Indeed, in the ground case loops only results from multiple appearance of an atom in some path, which can be easily detected. We can prune the subtrees of $\mathcal{T}$ rooted by atoms that already appeared at an earlier stage: this does not affect the computation of $\mathsf{Pr}_{\mathbb{P}}(A)$, and it makes $\mathcal{T}$ finite. Next, we introduce the concept of *deterministic* subtree. Basically a deterministic subtree selects one world-node at each stage. Recall that every clause-node in $\mathcal{T}$ represents a clause in $|\mathbb{P}|$, whose head is the label of its atom-grandparent, and body consists of the labels of its atom-children.

▶ **Definition 25.** *A subtree $\mathcal{S}$ of $\mathcal{T}$ is* deterministic *if (i) it contains exactly one child (world-node) for each atom-node and all children for other nodes, and (ii) for any distinct atom-nodes s, t in $\mathcal{S}$ with the same label, s and t have their clause-grandchildren representing the same clauses.*

The idea is that $\mathcal{S}$ describes a computation in which the choice of a possible world (i.e., a sub-program of $\mathbb{P}$) associated to any atom B appearing during the resolution is uniquely determined. Because of this feature, each deterministic subtree uniquely identifies a set of sub-programs of $\mathbb{P}$, and together the deterministic subtrees of $\mathcal{T}$ form a *partition* over the set of these sub-programs (see Proposition 27 below).

Since $\mathcal{T}$ is finite, it is clear that we can always provide an enumeration of its deterministic subtrees. We can now present our algorithm, in two steps. First, Algorithm 1 computes the probability associated with a deterministic subtree. Second, Algorithm 2 computes $\mathsf{Pr}_{\mathbb{P}}(A)$ by summing up the probabilities found by Algorithm 1 on all the deterministic subtrees of $\mathcal{T}$ which contains a refutation of A. Below we write $\mathrm{label}(s \to t)$ for the probability labelling the edge from s to t.

Algorithm 1 Compute probability of a deterministic subtree

Input: A deterministic subtree $\mathcal{S}$ of $\mathcal{T}$
Output: The probability of $\mathcal{S}$

```
probList = [ ]
for atom-node s in S do
    if s has child then
        probList += label(s → child(s))
if probList == [ ] then
    return 0
else prob = product of values in probList
    return prob
```

Algorithm 2 Compute probability of a goal

Input: The distribution tree $\mathcal{T}$ of A in $\mathbb{P}$
Output: The success probability $\mathsf{Pr}_{\mathbb{P}}(A)$

```
probSuc = 0
for deterministic subtree S of T do
    if S refutes A then
        probSuc += Algorithm 1(S)
return probSuc
```

The above procedure terminates because $\mathcal{T}$ is finite and every for-loop is finite. We now focus on the correctness of the algorithm.

Correctness. As mentioned, a world-node in a deterministic subtree can be seen as a choice of clauses: one chooses the clauses represented by its clause-children, and discards the clauses represented by its "complement" world. For correctness, we make this precise, via the following definition.

▶ **Definition 26.** *Given a clause $\mathcal{C}$ in $\mathbb{P}$, a deterministic subtree $\mathcal{S}$ of $\mathcal{T}$, a world-node t and its atom-parent s in $\mathcal{S}$, we say t* accepts *$\mathcal{C}$ if $\mathsf{Head}(\mathcal{C}) = \text{label}(s)$ and there is a clause-child of t that represents $\mathcal{C}$; t* rejects *$\mathcal{C}$ if $\mathsf{Head}(\mathcal{C}) = \text{label}(s)$ but no clause-child of t represents $\mathcal{C}$. We say $\mathcal{S}$* accepts *(*rejects*) $\mathcal{C}$ if there exists a world-node t in $\mathcal{S}$ accepts (rejects) $\mathcal{C}$.*

Note that Definition 25, condition (ii) prevents the existence of world-nodes t, t' in $\mathcal{S}$ such that t accepts $\mathcal{C}$ and t' rejects $\mathcal{C}$. Thus the notion that $\mathcal{S}$ accepts (rejects) $\mathcal{C}$ is well-defined. We denote the set of clauses accepted and rejected by $\mathcal{S}$ by $\mathsf{Acc}(\mathcal{S})$ and $\mathsf{Rej}(\mathcal{S})$, respectively. Then we can define the set $\mathsf{SubProg}(\mathcal{S})$ of sub-programs represented by $\mathcal{S}$ as

$$\mathsf{SubProg}(\mathcal{S}) := \{\mathbb{L} \subseteq |\mathbb{P}| \mid \forall \mathcal{C} \in \mathsf{Acc}(\mathcal{S}), \mathcal{C} \in \mathbb{L}; \forall \mathcal{C}' \in \mathsf{Rej}(\mathcal{S}), \mathcal{C}' \notin \mathbb{L}\} \tag{10}$$

We will prove the correctness of the algorithm through the following basic observations on the connection between deterministic subtrees and the sub-programs they represent:

▶ **Proposition 27.** *Suppose $\mathcal{S}$ is a deterministic subtree of the distribution tree $\mathcal{T}$ of A.*

1. *$\{\mathsf{SubProg}(\mathcal{S}) \mid \mathcal{S}$ is deterministic subtree of $\mathcal{T}\}$ forms a partition of $\mathcal{P}(\mathbb{P})$.*
2. *Either $\mathbb{L} \vdash A$ for all $\mathbb{L} \in \mathsf{SubProg}(\mathcal{S})$ or $\mathbb{L} \nvdash A$ for all $\mathbb{L} \in \mathsf{SubProg}(\mathcal{S})$.*

3. $\sum_{\mathbb{L}\in\mathsf{SubProg}(\mathcal{S})} \mathsf{Pr}_{\mathbb{P}}(\mathbb{L}) = \prod_{r_i\in\mathcal{S}} r_i$, *where the* r_i*s are all the probability labels appearing in* $\mathcal{S}$ *(on the atom-node* $\to$ *world-node edges).*

Proof.

1. Given any two distinct deterministic subtrees, there is an atom-node s such that the subtrees include distinct world-child of s. So by (10) the sub-programs they represent do not share at least one clause. Moreover, given a sub-program $\mathbb{L}$, one can always identify a deterministic subtree $\mathcal{S}$ such that $\mathbb{L} \in \mathsf{SubProg}(\mathcal{S})$, as follows: given the A-labelled root of $\mathcal{T}$, select the world-child w of A representing the (possibly empty) set X of all clauses in $\mathbb{L}$ whose head is A; then select the children (if any) of w, and repeat the procedure.
2. Note that a sub-program $\mathbb{L} \in \mathsf{SubProg}(\mathcal{S})$ refutes the goal A iff $\mathcal{S}$ contains a successful refutation of A, and the latter property is independent of the choice of $\mathbb{L}$.
3. We refer to $\prod_{r_i\in\mathcal{S}} r_i$ as the probability of the deterministic subtree $\mathcal{S}$. For each sub-program $\mathbb{L} \in \mathsf{SubProg}(\mathcal{S})$, its probability can be written as

$$\mathsf{Pr}_{\mathbb{P}}(\mathbb{L}) = \prod_{\mathcal{C}\in\mathsf{Acc}(\mathcal{S})} \mathsf{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}'\in\mathsf{Rej}(\mathcal{S})} (1 - \mathsf{Label}(\mathcal{C}')) \cdot \mathsf{Pr}_{\mathbb{P}\setminus(\mathsf{Acc}\cup\mathsf{Rej})}(\mathbb{L} \setminus \mathsf{Acc}(\mathcal{S})) \tag{11}$$

Note that $\mathsf{SubProg}(\mathcal{S})$ can also be written as $\{X \cup \mathsf{Acc}(\mathcal{S}) \mid X \subseteq \mathbb{P} \setminus (\mathsf{Acc}(\mathcal{S}) \cup \mathsf{Rej}(\mathcal{S}))\}$, so

$$\sum_{\mathbb{L}\in\mathsf{SubProg}(\mathcal{S})} \mathsf{Pr}_{\mathbb{P}\setminus(\mathsf{Acc}\cup\mathsf{Rej})}(\mathbb{L} \setminus \mathsf{Acc}(\mathcal{S})) = 1. \tag{12}$$

Applying equation (12) to the sum of (11) over all $\mathbb{L} \in \mathsf{SubProg}(\mathcal{S})$, we get

$$\sum_{\mathbb{L}\in\mathsf{SubProg}(\mathcal{S})} \mathsf{Pr}_{\mathbb{P}}(\mathbb{L}) = \prod_{\mathcal{C}\in\mathsf{Acc}(\mathcal{S})} \mathsf{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}'\in\mathsf{Rej}(\mathcal{S})} (1 - \mathsf{Label}(\mathcal{C}')) \tag{13}$$

For each world-node t and its atom-parent s, we can use the terminology in Definition 26, and express label$(s \to t)$ (see Definition 9) as

$$\text{label}(s \to t) = \prod_{t \text{ accepts } \mathcal{C}} \mathsf{Label}(\mathcal{C}) \cdot \prod_{t \text{ rejects } \mathcal{C}'} (1 - \mathsf{Label}(\mathcal{C}')). \tag{14}$$

Applying (14) to the whole deterministic subtree $\mathcal{S}$, we obtain

$$\begin{aligned}\sum_{\mathbb{L}\in\mathsf{SubProg}(\mathcal{S})} \mathsf{Pr}_{\mathbb{P}}(\mathbb{L}) &\overset{(13)}{=} \prod_{\mathcal{C}\in\mathsf{Acc}(\mathcal{S})} \mathsf{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}'\in\mathsf{Rej}(\mathcal{S})} (1 - \mathsf{Label}(\mathcal{C}')) \\ &\overset{\text{Def.26}}{=} \prod_{(\text{world-node } t \text{ in } \mathcal{S})} [\prod_{t \text{ accepts } \mathcal{C}} \mathsf{Label}(\mathcal{C}) \cdot \prod_{t \text{ rejects } \mathcal{C}'} (1 - \mathsf{Label}(\mathcal{C}'))] \\ &\overset{(14)}{=} \prod_{r_i\in\mathcal{S}} r_i\end{aligned}$$

If we say two world-nodes t and t' are equivalent if their clause-children represent exactly the same clauses in $\mathbb{P}$, then the $\prod_{(\text{world-node } t \text{ in } \mathcal{S})}$ in the above calculation visits every world-node exactly once modulo equivalence. ◀

We can now formulate the success probability of A as follows

$$\begin{aligned}\mathsf{Pr}_{\mathbb{P}}(A) = \sum_{|\mathbb{P}|\supseteq\mathbb{L}\vdash A} \mathsf{Pr}_{\mathbb{P}}(\mathbb{L}) &\overset{(\text{Prop.27,1\&2})}{=} \sum_{\mathcal{S}\vdash A} \sum_{\mathbb{L}\in\mathsf{SubProg}(\mathcal{S})} \mathsf{Pr}_{\mathbb{P}}(\mathbb{L}) \\ &\overset{(13)}{=} \sum_{\mathcal{S}\vdash A} [\prod_{\mathcal{C}\in\mathsf{Acc}(\mathcal{S})} \mathsf{Label}(\mathcal{C}) \cdot \prod_{\mathcal{C}'\in\mathsf{Rej}(\mathcal{S})} (1 - \mathsf{Label}(\mathcal{C}'))] \overset{(\text{Prop.27,3})}{=} \sum_{\mathcal{S}\vdash A} \prod_{r_i\in S} r_i\end{aligned}$$

In words, this is exactly Algorithm 2: we sum up the probabilities of all deterministic subtrees $\mathcal{S}$ of the distribution tree $\mathcal{T}$ which contain a proof of A.

B Computability of the Distribution Semantics (General Case)

Computability of the distribution semantics for arbitrary PLP programs relies on the substitution mechanism employed in the resolution. This aspect deserves a preliminary discussion. Traditionally, logic programming has both the theorem-proving and problem-solving perspectives [18]. From the problem-solving perspective, the aim is to find a refutation of the goal $\leftarrow G$, which amounts to finding a proof of *some substitution instance* of G. From the theorem-proving perspective, the aim is to search for a proof of the goal G itself as an atom. The main difference is in the substitution mechanism of resolution: unification for the problem-solving and term-matching for the theorem-proving perspective. We will first explore computability within the theorem-proving perspective. As resolution tehrein is by term-matching, the probability $\mathsf{Pr}^{\mathsf{TM}}_{\mathbb{P}}(A)$ of proving a goal A in a PLP program $\mathbb{P}$ is formulated as $\mathsf{Pr}^{\mathsf{TM}}_{\mathbb{P}}(A) := \sum_{|\mathbb{P}| \supseteq \mathbb{L} \Rightarrow A} \mathsf{Pr}_{\mathbb{P}}(\mathbb{L})$, where $\mathbb{L} \Rightarrow A$ means that A is derivable in the sub-program $\mathbb{L}$ (not to be confused with $\mathbb{L} \vdash A$, which stands for *some substitution instance* of A being derivable in $\mathbb{L}$, see (1)).

In our coalgebraic framework, the distribution semantics for general PLP programs is represented on "saturated" trees, in which computations are performed by unification. However, following [3], one can define the *TM (**T**erm **M**atching) distribution tree* of a goal A in a program $\mathbb{P}$ by "desaturation" of the saturated distribution tree for A in $\mathbb{P}$. The coalgebraic definition, for which we refer to [3], applies pointwise on the saturated tree the counit $\epsilon_{\mathcal{U}\mathsf{At}} \colon \mathcal{U}\mathcal{K}\mathcal{U}\mathsf{At} \to \mathcal{U}\mathsf{At}$ of the adjunction $\mathcal{U} \dashv \mathcal{K}$ (*cf.* (6)). The TM distribution tree which results from 'desaturation' can be described very simply: at each layer of the starting saturated distribution tree, one prunes all the subtrees which are not labelled with the identity substitution $\mathsf{id} := x_1 \mapsto x_1, x_2 \mapsto x_2, \dots$. In this way, the only remaining computation are those in which resolution only applies a non-trivial substitution on the clause side, that is, in which unification is restricted to term-matching.

Computability of term-macthing distribution semantics. One may compute the success probability $\mathsf{Pr}^{\mathsf{TM}}_{\mathbb{P}}(A)$ in $\mathbb{P}$ from the TM distribution tree of A in $\mathbb{P}$. The computation goes similarly to Algorithm 2 : the problem amounts to calculating the probabilities of those deterministic subtrees of the distribution tree which prove the goal. We confine ourselves to some remarks on the aspects that require extra care, compared to the ground case.

1. The probability $\mathsf{Pr}^{\mathsf{TM}}_{\mathbb{P}}(A)$ is not computable in whole generality. It depends on whether one can decide all the proofs of A in the pure logic program $|\mathbb{P}|$, and there are various heuristics in logic programming for this task.
2. It is still possible to decide whether a subtree is deterministic, but the algorithm in the general case is a bit subtler, as it is now possible that two different goals match the same clause (instantiated in two different ways).
3. When calculating the probability of a deterministic subtree in the TM distribution tree, multiple appearances of a single clause (possibly instantiated with different substitutions) should be counted only once. In order to ensure this one needs to be able to identify which clause is applied at each step of the computation described by the distribution trees: this is precisely the reason of the addition of the clause labels in the coalgebra type of these trees, as discussed in Section 4.3.

We conclude by briefly discussing the problem-solving perspective, in which resolution is based on arbitrary unification rather than just term-matching. In standard SLD-resolution, computability relies on the possibly of identifying the *most general* unifier between a goal and the head of a given clause. This can be done also within saturated distribution trees, since saturation supplies *all* the unifiers, thus in particular the most general one. This means that, on principle, one may compute the distribution semantics based on most general unification from the saturated distributed tree associated with a goal, with similar caveats as the ones we described for the term-matching case. However, the lack of a satisfactory coalgebraic treatment of most general unifiers [3] makes us privilege the theorem-proving perspective discussed above, for which desaturation provides an elegant categorical formalisation. This is also in line with the series of works [17, 19] on coalgebraic (pure) logic programming, all based on term-matching as substitution mechanism.