A Semantics for Hybrid Probabilistic Logic Programs with Function Symbols

Damiano Azzolini[a,*], Fabrizio Riguzzi[b], Evelina Lamma[a]

[a] *Dipartimento di Ingegneria - University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy*
[b] *Dipartimento di Matematica e Informatica - University of Ferrara, Via Saragat 1, I-44122, Ferrara, Italy*

Abstract

Probabilistic Logic Programming (PLP) is a powerful paradigm for the representation of uncertain relations among objects. Recently, programs with continuous variables, also called hybrid programs, have been proposed and assigned a semantics. Hybrid programs are capable of representing real world measurements but unfortunately the semantics proposal was imprecise so the definition did not assign a probability to all queries. In this paper, we remedy this and formally define a new semantics for hybrid programs. We prove that the semantics assigns a probability to all queries for a large class of programs. We also propose a concrete syntax for these programs and present several examples.

Keywords: Probabilistic Logic Programming, Hybrid Programs

1. Introduction

Probabilistic Logic Programming (PLP) [13, 37] has been attracting a growing interest for its ability of representing both relationships among entities and uncertainty over such relationships. Among the semantics proposed for probabilistic logic programs, the distribution semantics [40] gained prominence thanks to its intuitiveness and simplicity. The distribution semantics underlies many languages such as Probabilistic Horn Abduction [32], PRISM [40], Inde-

*Corresponding author
Email addresses: `damiano.azzolini@unife.it` (Damiano Azzolini), `fabrizio.riguzzi@unife.it` (Fabrizio Riguzzi), `evelina.lamma@unife.it` (Evelina Lamma)

Preprint submitted to Artificial Intelligence *November 15, 2023*

pendent Choice Logic [33], Logic Programs with Annotated Disjunctions [42], ProbLog [14] and CP-logic [43]. All these languages allow a countable number of discrete random variables. The semantics was proven well-defined for these programs in [36, 37].

The main limitation of these languages is that they do not allow continuous random variables and so they cannot properly represent several real world scenarios characterized, for instance, by temporal or physical models. However, in the last few years, languages that overcome this limitation have appeared: Hybrid ProbLog [15], Distributional Clauses (DC) [16], Extended PRISM [19], `cplint` hybrid programs [37], HAL-ProbLog [45] and Probabilistic Constraint Logic Programming (PCLP, for short, in the following) [24, 25, 26].

The semantics that have been proposed for such programs are able to consider a countable number of continuous random variables. However, none of the above proposals prove that the semantics is well-defined for a large class of programs.

In particular, here we consider the semantics of PCLP proposed in [26] that is one of the more detailed. The authors define a probability space for such programs composed of a sample space, a set of events and a measure. Events are the entities that can be assigned a measure value, i.e., a probability. However, the authors of [26] didn't prove that every query can be associated to an event, i.e., that every query can be assigned a probability.

In this paper, we remedy this by providing a new semantics, based on the Well-founded Semantics, that, for a large class of programs (for all the programs that provide, for every world, a two valued Well-founded model), assigns every query to an event and thus to a probability for the PCLP [24, 25, 26] language. Moreover, we provide a concrete language called `cplint` *hybrid programs* [37] for representing PCLP programs in a computer interpretable way. `cplint` is a suite of programs for reasoning and learning with PLP. It also has an online interface called cplint on SWISH [1, 38] available at `http://cplint.eu`.

The paper is structured as follows. In Sections 2 and 3 we review background theory about Well-founded Semantics and probability. The distribution semantics for programs with function symbols and Probabilistic Constraint Logic Programming are introduced respectively in Section 4 and 5. Some motivating examples can be found in Section 6, where we also illustrate the PCLP expressive power. In Sections 7 and 8 we introduce a new semantics for PCLP, we prove that it is well-defined and we provide a concrete, running system for querying PCLP. Finally, in Section 9 we discuss several related semantics proposals (Section 9.1) and existing inference algorithms for hybrid programs (Section 9.2). Section 10 concludes the paper.

2. Logic Programming and Well-founded Semantics

A normal logic program [23] is a set of normal *clauses* of the form

$$h \leftarrow l_1, \ldots, l_n$$

with ($n \geqslant 0$), where h is an *atom* and each l_i is a *literal*. An atom is an expression of the form $p(t_1, \ldots, t_n)$ where p is a predicate name and $t_1, \ldots, t_n$ are *terms*. If the terms do not contain variables, the atom is called *ground*. A literal is an atom a or its negation (denoted with $\sim a$). Variables and constants are terms and, if f is a function symbol with arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is also a term. In this example, h is called the *head* of the clause, while the conjunctions of literal $l_1, \ldots, l_n$ represents the *body*. A *substitution* $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ is a function mapping variables (X_i) to terms (t_i), i.e., it replaces all the occurrences of variables X_i in a formula with terms t_i, where a formula can be a term, atom, literal, clause or program. Given a formula F, the result of the substitution is denoted with $F\theta$ and is called *instance* of F. A substitution θ is *grounding* for a formula F if $F\theta$ is ground, i.e., $F\theta$ does not contain variables.

The *Herbrand universe* $\mathcal{U}_P$ of a program P is the set of all the ground terms obtained by the possible combinations of the symbols in the program. Similarly, the *Herbrand base* $\mathcal{B}_P$ of a program P is the set of all ground atoms constructed using the symbols in the program. The grounding of a program is obtained by replacing the variables of clauses in the program with the terms from the Herbrand universe in all possible ways. A *two-valued* interpretation $I \subseteq \mathcal{B}_P$ represents the set of true atoms: a is true in I if $a \in I$ and a is false in I if $a \notin I$. Given an interpretation I, a ground atom $p(t_1, \ldots, t_n)$ is true in I if $p(t_1, \ldots, t_n) \in I$, a ground clause $h_1; \ldots; h_m \leftarrow b_1, \ldots, b_n$, where semicolons denote disjunctions, is true in I if at least one of the h_i is true when $b_1, \ldots, b_n$ are true in I, a clause c is true in I if all of its groundings with terms from $\mathcal{U}_P$ are true in I and a set of clauses C is true in I if $\forall c \in C$, c is true in I.

An interpretation I is a *model* for a set of clauses Σ, denoted with $I \models \Sigma$, if Σ is true in I. We call Int_2^P the set of two-valued interpretations for a program P. The set Int_2^P forms a complete lattice (see Appendix A for a definition of lattice) where the partial order $\leqslant$ is defined by the subset relation $\subseteq$. A *three-valued interpretation* $\mathcal{I}$ is a pair $\langle I_T, I_F \rangle$ where I_T and I_F are subsets of $\mathcal{B}_P$ and represent respectively the set of true and false atoms. a is true in $\mathcal{I}$ if $a \in I_T$ and is false in $\mathcal{I}$ if $a \in I_F$. $\sim a$ is true in $\mathcal{I}$ if $a \in I_F$ and is false in $\mathcal{I}$ if $a \in I_T$. If $a \notin I_T$ and $a \notin I_F$, then a assumes the third truth value, *undefined*. We also write $\mathcal{I} \models a$ if $a \in I_T$ and $\mathcal{I} \models \sim a$ if $a \in I_F$. We call Int_3^P the set of three-valued interpretations for a program P. A three-valued interpretation $\mathcal{I} = \langle I_T, I_F \rangle$ is *consistent* if $I_T \cap I_F = \varnothing$. The union of two three-valued interpretations $\langle I_T, I_F \rangle$ and $\langle J_T, J_F \rangle$ is defined as $\langle I_T, I_F \rangle \cup \langle J_T, J_F \rangle = \langle I_T \cup J_T, I_F \cup J_F \rangle$. The intersection of two three-valued interpretations $\langle I_T, I_F \rangle$ and $\langle J_T, J_F \rangle$ is defined as $\langle I_T, I_F \rangle \cap \langle J_T, J_F \rangle = \langle I_T \cap J_T, I_F \cap J_F \rangle$. In the following, we represent a three-valued interpretation $\mathcal{I} = \langle I_T, I_F \rangle$ as a single set of literals, i.e.,

$$\mathcal{I} = I_T \cup \{\sim a \mid a \in I_F\}.$$

The set Int_3^P of three-valued interpretations for a program P forms a complete lattice where the partial order $\leqslant$ is defined as $\langle I_T, I_F \rangle \leqslant \langle J_T, J_F \rangle$ if $I_T \subseteq J_T$ and $I_F \subseteq J_F$. The bottom and top element for Int_2^P are respectively $\varnothing$ and $\mathcal{B}_P$ while for Int_3^P are respectively $\langle \varnothing, \varnothing \rangle$ and $\langle \mathcal{B}_P, \mathcal{B}_P \rangle$.

Given a three-valued interpretation $\mathcal{I} = \langle I_T, I_F \rangle$, we define the functions

$true(\mathcal{I}) = I_T$, $false(\mathcal{I}) = I_F$ and $undef(\mathcal{I}) = \mathcal{B}_P \backslash (I_T \cup I_F)$, that return the set of true, false and undefined atoms respectively.

The Well-founded semantics (WFS) [41] assigns a three-valued model to a normal logic program, i.e., it identifies a consistent three-valued interpretation as the meaning of the program. The WFS was given in [41] in terms of the least fixpoint of an operator that is composed by two sub-operators, one computing consequences and the other computing unfounded sets. We give here the alternative definition of the WFS of [35] that is based on an iterated fixpoint. See Appendix B for a brief introduction about fixpoints.

Definition 1 ($OpTrue_{\mathcal{I}}^P$ and $OpFalse_{\mathcal{I}}^P$ operators)**.** *For a normal logic program P, sets Tr and Fa of ground atoms, and a 3-valued interpretation $\mathcal{I}$, we define the operators $OpTrue_{\mathcal{I}}^P : Int_2^P \to Int_2^P$ and $OpFalse_{\mathcal{I}}^P : Int_2^P \to Int_2^P$ as*

$$\begin{aligned} OpTrue_{\mathcal{I}}^P(Tr) = \ & \{a \mid a \textit{ is not true in } \mathcal{I} \textit{ and there is a clause } b \leftarrow l_1, \ldots, l_n \textit{ in } P \\ & \textit{and a grounding substitution } \theta \textit{ such that } a = b\theta \textit{ and, for every } 1 \leqslant i \leqslant n, \\ & \textit{either } l_i\theta \textit{ is true in } \mathcal{I} \textit{ or } l_i\theta \in Tr\} \end{aligned}$$

$$\begin{aligned} OpFalse_{\mathcal{I}}^P(Fa) = \ & \{a \mid a \textit{ is not false in } \mathcal{I} \textit{ and for every clause } b \leftarrow l_1, \ldots, l_n \textit{ in } P \\ & \textit{and grounding substitution } \theta \textit{ such that } a = b\theta \textit{ there is some } i \ (1 \leqslant i \leqslant n) \\ & \textit{such that } l_i\theta \textit{ is false in } \mathcal{I} \textit{ or } l_i\theta \in Fa\} \end{aligned}$$

In words, the operator $OpTrue_{\mathcal{I}}^P(Tr)$ extends the interpretation $\mathcal{I}$ to add the new true atoms that can be derived from P knowing $\mathcal{I}$ and true atoms Tr, while $OpFalse_{\mathcal{I}}^P(Fa)$ computes new false atoms in P by knowing $\mathcal{I}$ and false atoms Fa. $OpTrue_{\mathcal{I}}^P$ and $OpFalse_{\mathcal{I}}^P$ are both monotonic [35], so they both have least and greatest fixpoint. An iterated fixpoint operator builds up *dynamic strata* by constructing successive three-valued interpretations as follows.

Definition 2 (Iterated fixed point)**.** *For a normal logic program P, let $IFP^P : Int_3^P \to Int_3^P$ be defined as*

$$IFP^P(\mathcal{I}) = \mathcal{I} \cup \langle \mathrm{lfp}(OpTrue_{\mathcal{I}}^P), \mathrm{gfp}(OpFalse_{\mathcal{I}}^P) \rangle$$

where lfp and gfp denote respectively the least and the greatest fixpoint. IFP^P is monotonic [35] and thus it has a least fixpoint lfp(IFP^P). The Well-Founded Model (WFM) of P, denoted as $WFM(P)$, is lfp(IFP^P). Let δ be the smallest ordinal such that $WFM(P) = IFP^P \uparrow \delta$. We refer to δ as the *depth* of P. The *stratum* of atom a is the least ordinal β such that $a \in IFP^P \uparrow \beta$ (where a may be either in the true or false component of $IFP^P \uparrow \beta$). Undefined atoms of the WFM do not belong to any stratum, i.e., they are not added to $IFP^P \uparrow \delta$ for any ordinal δ.

If $undef(WFM(P)) = \varnothing$, then the WFM is called *total* or *two-valued* and the program is *dynamically stratified*.

3. Probability Theory

In this section, we review some background on probability theory, in particular Kolmogorov probability theory, that will be needed in the following. Most of the definitions are taken from [10] and [37].

We define the *sample space* W as the set composed by the elements that are outcomes of the random process we want to model. For instance, if we consider the toss of a coin whose outcome could be heads h or tails t, the sample space is defined as $W^{coin} = \{h, t\}$. If we throw 2 coins, then $W^{2coins} = \{(h,h),(h,t),(t,h),(t,t)\}$. If the number of coins is infinite then $W^{coins} = \{(o_1, o_2, \ldots) \mid o_i \in \{h,t\}\}$.

Definition 3 (σ-Algebra)**.** *A non-empty set Ω of subsets of W is a σ-algebra on the set W iff:*

- $W \in \Omega$
- Ω *is closed under complementation:* $\omega \in \Omega \Rightarrow \omega^c = \Omega \backslash \omega \in \Omega$
- Ω *is closed under countable union: if* $\omega_i \in \Omega \Rightarrow \bigcup_i w_i \in \Omega$

The elements of a σ-algebra Ω are called *measurable sets* or *events*, Ω is called event space and (W, Ω) is called *measurable space*. When W is finite,

Ω is usually the powerset of W, but, in general, it is not necessary that every subset of W must be present in Ω. For example, to model a coin toss, we can consider the set of events $\Omega^{coin} = \mathscr{P}(W^{coin})$ and $\{h\}$ an event corresponding to the outcome heads.

Definition 4 (Minimal σ-algebra)**.** *Let $\mathcal{C}$ be an arbitrary non-empty collection of subsets of W. The intersection of all σ-algebras containing all the elements of $\mathcal{C}$ is called the σ-algebra generated by $\mathcal{C}$ or the minimal sigma-algebra containing $\mathcal{C}$. It is denoted by $\sigma(\mathcal{C})$. Moreover, $\sigma(\mathcal{C})$ always exists and is unique [10].*

Now we introduce the definition of probability measure:

Definition 5 (Probability measure)**.** *Given a measurable space (W, Ω), a probability measure is a finite set function $\mu : \Omega \to \mathbb{R}$ that satisfies the following three axioms (called Kolmogorov axioms):*

- *a_1: $\mu(\omega) \geqslant 0 \ \forall \ \omega \in \Omega$*
- *a_2: $\mu(W) = 1$*
- *a_3: μ is countably additive (or σ-additive): if $O = \{\omega_1, \omega_2, \ldots\} \subseteq \Omega$ is a countable collection of pairwise disjoint sets, then $\mu(\bigcup_{\omega \in O}) = \sum_i \mu(\omega_i)$*

Axioms a_1 and a_2 state that we measure the probability of an event with a number between 0 and 1. Axiom a_3 states that the probability of the union of disjoint events is equal to the sum of the probability of every single event. (W, Ω, μ) is called a *probability space*.

For example, if we consider the toss of a coin, $(W^{coin}, \Omega^{coin}, \mu^{coin})$ with $\mu^{coin}(\varnothing) = 0$, $\mu^{coin}(\{h\}) = 0.5$, $\mu^{coin}(\{t\}) = 0.5$ and $\mu^{coin}(\{h, t\}) = 1$ is a probability space.

Definition 6 (Measurable function)**.** *Given a probability space (W, Ω, μ) and a measurable space (S, Σ), a function $X : W \to S$ is measurable if $X^{-1}(\sigma) = \{w \in W \mid X(w) \in \sigma\} \in \Omega, \ \forall \sigma \in \Sigma$.*

Definition 7 (Random variable)**.** *Let (W, Ω, μ) be a probability space and (S, Σ) be a measurable space. A measurable function $X : W \to S$ is a random variable. The elements of S are called values of X. We indicate with $P(X \in \sigma)$ for all $\sigma \in \Sigma$ the probability that a random variable X has value in σ, that is, $\mu(X^{-1}(\sigma))$. If S is finite or countable, X is a discrete random variable. If S is uncountable, X is a continuous random variable.*

The *probability distribution* of a discrete random variable is defined as $P(X \in \{x\})\ \forall x \in S$ and it is often abbreviated with $P(X = x)$ or $P(x)$. The *probability density* $p(X)$ of a continuous random variable $X : (W, \Omega) \to (\mathbb{R}, \mathcal{B})$ is defined such as $P(X \in A) = \int_A p(x)dx$ for any measurable set $A \in \mathcal{B}$.

In the following, we will need to consider the product of measurable spaces.

Definition 8 (Product σ-algebra)**.** *Given two measurable spaces (W_1, Ω_1) and (W_2, Ω_2), the product σ-algebra $\Omega_1 \otimes \Omega_2$ is defined as $\Omega_1 \otimes \Omega_2 = \sigma(\{\omega_1 \times \omega_2 \mid \omega_1 \in \Omega_1, \omega_2 \in \Omega_2\})$. The result of $\Omega_1 \otimes \Omega_2$ is different from the Cartesian product $\Omega_1 \times \Omega_2$ because it is the minimal σ-algebra generated by all the possible couples of elements from Ω_1 and Ω_2. $\Omega_1 \otimes \Omega_2$ is also called a tensor product.*

Theorem 1 (Theorem 6.3.1 from [10])**.** *Given two probability spaces (W_1, Ω_1, μ_1) and (W_2, Ω_2, μ_2), there exists a unique probability space (W, Ω, μ) such that $W = W_1 \times W_2$, $\Omega = \Omega_1 \otimes \Omega_2$ and*

$$\mu(\omega_1 \times \omega_2) = \mu_1(\omega_1) \cdot \mu_2(\omega_2)$$

for $\omega_1 \in \Omega_1$ and $\omega_2 \in \Omega_2$. Measure μ is called the product measure *of μ_1 and μ_2 and is denoted also by $\mu_1 \times \mu_2$. Moreover, for any $\omega \in \Omega$, let's define its* sections *as*

$$\omega^{(1)}(w_1) = \{w_2 \mid (w_1, w_2) \in \omega\} \qquad \omega^{(2)}(w_2) = \{w_1 \mid (w_1, w_2) \in \omega\}.$$

Then, both $\omega^{(1)}(w_1)$ and $\omega^{(2)}(w_2)$ are measurable according to (W_2, Ω_2, μ_2) and (W_1, Ω_1, μ_1) respectively, i.e., $\omega^{(1)}(w_1) \in \Omega_2$ and $\omega^{(2)}(w_2) \in \Omega_1$. $\mu_2(\omega^{(1)}(w_1))$

and $\mu_1(\omega^{(2)}(w_2))$ are well-defined real functions, the first on W_1 and the second on W_2.

Measure $\mu = \mu_1 \times \mu_2$ for every $\omega \in \Omega$ also satisfies

$$\mu(\omega) = \int_{W_2} \mu_1(\omega^{(2)}(w_2)) d\mu_2 = \int_{W_1} \mu_2(\omega^{(1)}(w_1)) d\mu_1.$$

4. The Distribution Semantics for Programs with Function Symbols

Probabilistic Logic Programming (PLP) extends logic programming with the possibility of expressing uncertain relations. Several PLP languages has been proposed during the years. The language we propose is based on ProbLog syntax and semantics. In this section we present the distribution semantics for ProbLog programs with function symbols. Let us consider first ProbLog programs without function symbols.

A probabilistic logic program P is composed by a set of clauses (or rules) R and a set of probabilistic facts F which are of the form

$$\Pi :: f$$

where Π is a probability and f is an atom. If f is not ground, the fact stands for a set of facts, one for each grounding.

Let us consider an example.

Example 1 (Graph)**.** *Consider a probabilistic graph where the edges have a probability of existing.*

$F_1 = 1/3 :: edge(a, b).$
$F_2 = 1/2 :: edge(b, c).$
$F_3 = 1/4 :: edge(a, c).$
$path(X, X).$
$path(X, Y) \leftarrow edge(X, Z), path(Z, Y).$

This program has three ground probabilistic facts, each corresponding to one edge, and two clauses. With this program we can compute the probability of the existence of a path between two nodes, for example, by asking for the probability of $path(a, c)$ being true.

In order to give a semantics to ProbLog programs without function symbols, let us introduce some terminology. An *atomic choice* indicates whether a grounding $f\theta$ of a probabilistic fact $\Pi :: f$ is selected or not and is represented with the triple (f, θ, k) where $k \in \{0, 1\}$. $k = 1$ means that the fact is selected, $k = 0$ that it is not. A set of atomic choices is *consistent* if only one alternative is selected for a grounding of a probabilistic fact, i.e., it does not contain atomic choices such as (f, θ, j) and (f, θ, k) with $j \neq k$. Finally, we define a *composite choice* κ as a consistent set of atomic choices. Given a composite choice κ we can define its probability as

$$P(\kappa) = \prod_{(f_i,\theta,1)\in\kappa} \Pi_i \prod_{(f_i,\theta,0)\in\kappa} 1 - \Pi_i.$$

A *selection* σ (also called total composite choice) contains one atomic choice for every grounding of every probabilistic fact. A selection σ identifies a *world* w_σ, i.e., a logic program containing the rules R and atoms corresponding to each atomic choice $(f, \theta, 1)$ of σ. The way to assign a probability to composite choices applies also to selections, so we have a way of assigning a probability to worlds.

Since there are no function symbols, the Herbrand universe is finite and so is the set of groundings of probabilistic facts. Therefore, the set of worlds is finite, and each world is determined by a finite number of choices. $P(\sigma)$ as defined above is a probability distribution over the worlds.

We want to assign a probability to ground atoms. We assume that each world has a total well-founded model, i.e., each ground atom is either true or false in the world, but it cannot be undefined. We call programs satisfying this property *sound*.

Given a ground atom q and a world w we can thus define the conditional probability $P(q \mid w)$ as 1 if $w \models q$ and 0 otherwise.

The probability of q can be computed by summing out the worlds from the joint distribution of the query and the worlds:

$$P(q) = \sum_{w} P(q, w) = \sum_{w} P(q \mid w)P(w) = \sum_{w \models q} P(w). \tag{1}$$

Example 2 (Graph, continued)**.** *The program of Example 1 has three ground probabilistic facts so it has* $2^3 = 8$ *worlds. The query* $path(a, c)$ *is true in 5 of them and its probability is*

$$\begin{aligned} P(path(a,c)) &= 1/3 \cdot 1/2 \cdot 3/4 + 2/3 \cdot 1/2 \cdot 1/4 + 2/3 \cdot 1/2 \cdot 1/4 \\ &+ 1/3 \cdot 1/2 \cdot 1/4 + 1/3 \cdot 1/2 \cdot 1/4 \\ &= 0.375 \end{aligned}$$

If the program contains function symbols, the Herbrand universe is countable and the set of groundings of probabilistic facts is countable as well. The set of worlds in this case is uncountable, as will be shown later by Theorem 5, and the probability of each world is 0, as it is given by an infinite product of numbers all bounded away from 1. Therefore, the semantics cannot be given as above.

Let us consider an example with function symbols.

Example 3 (Game of dice)**.** *Consider the game of dice proposed in [42]: the player repeatedly throws a six-sided die. The game stops when the outcome is six. If we consider a game played with a three sided die, where the game stops when the outcome is three, a possible ProbLog encoding could be:*

$$\begin{aligned}
&F_1 = 1/3 :: one(X).\\
&F_2 = 1/2 :: two(X).\\
&on(0,1) \leftarrow one(0).\\
&on(0,2) \leftarrow \sim one(0), two(0).\\
&on(0,3) \leftarrow \sim one(0), \sim two(0).\\
&on(s(X),1) \leftarrow on(X,_), \sim on(X,3), one(s(X)).\\
&on(s(X),2) \leftarrow on(X,_), \sim on(X,3), \sim one(s(X)), two(s(X)).\\
&on(s(X),3) \leftarrow on(X,_), \sim on(X,3), \sim one(s(X)), \sim two(s(X)).
\end{aligned}$$

If we add the clauses

$$\begin{aligned}
&at_least_once_1 \leftarrow on(_,1).\\
&never_1 \leftarrow \sim at_least_once_1.
\end{aligned}$$

we can ask for the probability that the die landed at least once on face 1 and that the die never landed on face 1.

Let us introduce some more definitions. With W_P we denote the set of all worlds of a probabilistic logic program P. The *set of worlds* ω_κ *compatible with a composite choice* κ is $\omega_\kappa = \{w_\sigma \in W_P \mid \kappa \subseteq \sigma\}$. Therefore, a composite choice identifies a set of worlds. For programs with function symbols, ω_κ may be uncountable so it is not guaranteed that $\sum_{w\in\omega_\kappa} P(w)$ can be defined, since $P(w) = 0$. However, $P(\kappa)$ is still well-defined. Let us call $\mu(\kappa) = P(\kappa)$.

Given a set of composite choices K, the *set of worlds* ω_K *compatible with* K is defined as $\omega_K = \bigcup_{\kappa\in K} \omega_\kappa$. Two sets K_1 and K_2 of composite choices are *equivalent* if $\omega_{K_1} = \omega_{K_2}$, that is, if they correspond to the same set of worlds. If the union of two composite choices κ_1 and κ_2 is not consistent, then κ_1 and κ_2 are *incompatible*. We define *pairwise incompatible* a set K of composite choices if $\forall \kappa_1 \in K, \forall \kappa_2 \in K$, $\kappa_1 \neq \kappa_2$ implies that κ_1 and κ_2 are incompatible.

Obtaining pairwise incompatible sets of composite choices (for both probabilistic logic programs with finite and infinite number of worlds) is a crucial problem, since the *probability of a pairwise incompatible set* K *of composite choices* for programs without function symbols can be defined as $P(K) = \sum_{\kappa\in K} P(\kappa)$, which can be easily computed. $P(K)$ is still well-defined for programs with function symbols if K is countable. Let us call it μ so $\mu(K) = P(K)$.

We can assign probabilities to a general set K of composite choices by constructing a pairwise incompatible equivalent set through the technique of *splitting*. In detail, if $f\theta$ is an instantiated fact and κ is a composite choice that does not contain an atomic choice (f, θ, k) for any k, the *split* of κ on $f\theta$ can be defined as the set of composite choices $S_{\kappa,f\theta} = \{\kappa \cup \{(f, \theta, 0)\}, \kappa \cup \{(f, \theta, 1)\}\}$. In this way, κ and $S_{\kappa,f\theta}$ identify the same set of possible worlds, i.e., $\omega_\kappa = \omega_{S_{\kappa,f\theta}}$, and $S_{\kappa,f\theta}$ is pairwise incompatible. It turns out that, given a set of composite choices, by repeatedly applying splitting it is possible to obtain an equivalent mutually incompatible set of composite choices [34].

Theorem 2 (Existence of a pairwise incompatible set of composite choices [34])**.** *Given a finite set* K *of composite choices, there exists a finite set* K' *of pairwise incompatible composite choices equivalent to* K*.*

Theorem 3 (Equivalence of the probability of two equivalent pairwise incompatible finite set of finite composite choices [31])**.** *If K_1 and K_2 are both pairwise incompatible finite sets of finite composite choices such that they are equivalent, then $P(K_1) = P(K_2)$.*

Given a finite pairwise incompatible set K' of composite choices equivalent to K, a measure for a probabilistic logic program P is defined as $\mu_P(\omega_K) = \mu(K')$.

We say that a composite choice κ is an *explanation* for a query q if $\forall w \in \omega_\kappa : w \models q$. Moreover, a set K of composite choices is *covering* with respect to a query q if every world in which q is true belongs to ω_K.

Example 4 (Pairwise incompatible covering set of explanations for Example 3)**.** *In Example 3, the query $at\_least\_once\_1$ has the pairwise incompatible covering set of explanations*

$$K^+ = \{\kappa_0^+, \kappa_1^+, \ldots\}$$

with

$$\begin{aligned}
\kappa_0^+ &= \{(f_1, \{X/0\}, 1)\} \\
\kappa_1^+ &= \{(f_1, \{X/0\}, 0), (f_2, \{X/0\}, 1), (f_1, \{X/s(0)\}, 1)\} \\
&\ldots \\
\kappa_i^+ &= \{(f_1, \{X/0\}, 0), (f_2, \{X/0\}, 1), \ldots, (f_1, \{X/s^{i-1}(0)\}, 0), \\
&\quad (f_2, \{X/s^{i-1}(0)\}, 1), (f_1, \{X/s^i(0)\}, 1)\} \\
&\ldots
\end{aligned}$$

So K^+ is countable and infinite. The query $never\_1$ has the pairwise incompatible covering set of explanations

$$K^- = \{\kappa_0^-, \kappa_1^-, \ldots\}$$

with

$$\begin{aligned}
\kappa_0^- &= \{(f_1, \{X/0\}, 0), (f_2, \{X/0\}, 0)\} \\
\kappa_1^- &= \{(f_1, \{X/0\}, 0), (f_2, \{X/0\}, 1), (f_1, \{X/s(0)\}, 0), \\
&\quad (f_2, \{X/s(0)\}, 0)\} \\
&\dots \\
\kappa_i^- &= \{(f_1, \{X/0\}, 0), (f_2, \{X/0\}, 1), \dots, (f_1, \{X/s^{i-1}(0)\}, 0), \\
&\quad (f_2, \{X/s^{i-1}(0)\}, 1), (f_1, \{X/s^i(0)\}, 0), (f_2, \{X/s^i(0)\}, 0)\} \\
&\dots
\end{aligned}$$

For a probabilistic logic program P and a ground atom q, we define the function $Q : W_P \to \{0, 1\}$ as

$$Q(w) = \begin{cases} 1 & \text{if } w \models q \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Given a probabilistic logic program P, we call Ω_P the set of sets of worlds identified by countable sets of countable composite choices, i.e., $\Omega_P = \{\omega_K \mid K$ is a countable set of countable composite choices$\}$.

Lemma 1 (σ-algebra of a Program, Lemma 2 of [37]). *Ω_P is a σ-algebra over W_P.*

We can define a probability measure μ_P as follows: $\mu_P : \Omega_P \to [0, 1]$. Given $K = \{\kappa_1, \kappa_2, \dots\}$ (K may be also infinite, i.e., it may contain an infinite number of κ_i), consider the sequence $\{K_n \mid n \geqslant 1\}$ where $K_n = \{\kappa_1, \dots, \kappa_n\}$. K_n is an increasing sequence and so $\lim_{n\to\infty} K_n$ exists and is equal to $\bigcup_{n=1}^{\infty} K_n = K$ [10]. Consider the sequence $\{K'_n \mid n \geqslant 1\}$ constructed as follows: $K'_1 = \{\kappa_1\}$ and K'_n is obtained by the union of K'_{n-1} with the splitting of each element of K'_{n-1} with κ_n. It is possible to prove by induction that K'_n is pairwise incompatible and equivalent to K_n.

Since $\mu(\kappa) = 0$ for infinite composite choices, we can compute $\mu(K'_n)$ for each K'_n. Consider $\lim_{n\to\infty} \mu(K'_n)$, then the following lemma holds:

Lemma 2 (Existence of the limit of the measure of countable union of countable composite choices, Lemma 3 from [37])**.** $\lim_{n\to\infty} \mu(K'_n)$ *exists.*

We can now introduce the definition of the probability space of a program.

Theorem 4 (Probability space of a program, Theorem 8 from [37])**.** *Given a set of composite choices $K = \{\kappa_1, \kappa_2, \ldots\}$ and a pairwise incompatible set of composite choices K'_n equivalent to $\{\kappa_1, \ldots, \kappa_n\}$, the triple $\langle W_P, \Omega_P, \mu_P\rangle$ with*

$$\mu_P(\omega_K) = \lim_{n\to\infty} \mu(K'_n)$$

is a probability space.

Given a probabilistic logic program P and a ground atom q with a countable set K of explanations that is covering with respect to q, Equation 2 represents a random variable, since $\{w \mid w \in W_P \wedge w \models q\} = \omega_K \in \Omega_P$.

For brevity, we indicate $P(Q = 1)$ with $P(q)$ and we say that q is *well-defined* according to the distribution semantics. If the probability of all ground atoms in the grounding of a probabilistic logic program P is well-defined, then P is *well-defined*.

Example 5 (Probability of the query for Example 3)**.** *From Example 4, the explanations in K^+ are pairwise incompatible, so the probability of the query $at\_least\_once\_1$ can be computed as*

$$\begin{aligned} P(at\_least\_once\_1) &= \frac{1}{3} + \frac{1}{3} \cdot \left(\frac{2}{3} \cdot \frac{1}{2}\right) + \frac{1}{3} \cdot \left(\frac{2}{3} \cdot \frac{1}{2}\right)^2 \\ &= \frac{1}{3} + \frac{1}{3} \cdot \left(\frac{1}{3}\right) + \frac{1}{3} \cdot \left(\frac{1}{3}\right)^2 + \ldots \\ &= \frac{1}{3} \cdot \frac{1}{1 - \frac{1}{3}} = \frac{1}{3} \cdot \frac{3}{2} = \frac{1}{2} \end{aligned}$$

since the sum represents a geometric series and $\sum_{n=0}^{\infty} k \cdot q^n = k \cdot \frac{1}{1-q}$.

Analogously, for the query $never\_1$, the explanations in K^- are pairwise *incompatible, so the probability of never_1 can be computed as*

$$
\begin{aligned}
P(never\_1) &= \frac{2}{3} \cdot \frac{1}{2} + \frac{2}{3} \cdot \frac{1}{2} \cdot \left(\frac{2}{3} \cdot \frac{1}{2}\right) + \\
&\quad \frac{2}{3} \cdot \frac{1}{2} \cdot \left(\frac{2}{3} \cdot \frac{1}{2}\right)^2 + \ldots \\
&= \frac{1}{3} + \frac{1}{3} \cdot \left(\frac{1}{3}\right) + \frac{1}{3} \cdot \left(\frac{1}{3}\right)^2 + \ldots \\
&= \frac{1}{3} \cdot \frac{1}{1 - \frac{1}{3}} = \frac{1}{3} \cdot \frac{3}{2} = \frac{1}{2}
\end{aligned}
$$

As expected, $P(never\_1) = 1 - P(at\_least\_once\_1)$*.*

In [36, 37] it was proved that any query to a sound ProbLog program can be assigned a probability so that the program is well-defined. In this paper, we want to do the same for PCLP.

5. Probabilistic Constraint Logic Programming (PCLP)

In this section, we introduce the basic concepts described in [26].

A program P in PCLP is composed by a set of *rules* (R) and a countable set of random variables (X). The rules define the truth value of the atoms in the Herbrand base of the program given the values of the random variables. Let $\mathrm{X} = \{\mathrm{X}_1, \mathrm{X}_2, \ldots\}$ be the countable set of random variables. Each random variable X_i has an associated range $Range_i$ that can be discrete, $\mathbb{R}$ or $\mathbb{R}^n$.

The sample space of a set X is defined as $W_{\mathrm{X}} = Range_1 \times Range_2 \times \ldots$. Each random variable X_i is associated to a probability space $(Range_i, \Omega_i, \mu_i)$. The measure space (W_{X}, Ω) is the product of measure spaces $(Range_i, \Omega_i)$, so it is an infinite-dimensional product measure space [10]. It is possible to build a probability space for any finite subset of X as a product probability space. Theorem 6.4.1 from [10] states that these finite dimensional probability spaces can be extended to an infinite dimensional probability space $(W_{\mathrm{X}}, \Omega_{\mathrm{X}}, \mu_{\mathrm{X}})$.

A *constraint* φ is a function $\varphi : W_{\mathrm{X}} \to \{true, false\}$, i.e., a function from $\mathrm{X}_1 = x_1$, $\mathrm{X}_2 = x_2$, …, to $\{true, false\}$, where $x_i \in Range_i$. Given a sample space W_{X}, and a constraint φ, we can define the *constraint solution space*

$CSS(\varphi)$ as the subset of the sample space W_{X} where the constraint φ holds: $CSS(\varphi) = \{x \in W_{\mathrm{X}} \mid \varphi(x)\}$.

We indicate with *satisfiable*(ω_{X}) the set of all constraints that are satisfiable given a valuation w_{X} of the random variables in X.

We can now define a *probabilistic constraint logic theory.*

Definition 9 (Probabilistic Constraint Logic Theory)**.** *A probabilistic constraint logic theory P is a tuple $(\mathrm{X}, W_{\mathrm{X}}, \Omega_{\mathrm{X}}, \mu_{\mathrm{X}}, Constr, R, F)$ where:*

- X *is a countable set of random variables $\{X_1, X_2, \ldots\}$. Each random variable X_i has a non-empty range $Range_i$;*
- $W_{\mathrm{X}} = Range_1 \times Range_2 \times \ldots = \times_{i\in \mathrm{X}} Range_i$ *is the sample space of the random variables* X*;*
- Ω_{X} *is the event space;*
- μ_{X} *is a probability measure, i.e.,* $(W_{\mathrm{X}}, \Omega_{\mathrm{X}}, \mu_{\mathrm{X}})$ *is a probability space;*
- *Constr is a set of constraints closed under conjunction, disjunction and negation such that $\forall \varphi \in Constr,\ CSS(\varphi) \in \Omega_{\mathrm{X}}$, i.e., such that $CSS(\varphi)$ is measurable for all φ;*
- *R is a set of rules with logical constraints, i.e., rules of the form:*

  $h \leftarrow l_1, \ldots, l_n, \langle\varphi_1(\mathrm{X})\rangle, \ldots, \langle\varphi_m(\mathrm{X})\rangle$*, where l_i is a literal for $i = 1, \ldots, n$, $\varphi_j \in Constr$; $\langle\varphi_j(\mathrm{X})\rangle$ is called constraint atom for $j = 1, \ldots, m$.*

Each atom in the Herbrand base $\mathcal{B}_P$ of R is a Boolean random variable. There is a countable number of them. The sample space W_R is defined as

$$W_R = \prod_{a\in\mathcal{B}_P} \{true, false\}.$$

The authors of [26] define the event space of the logic part of the theory as

$$\Omega_R = \mathscr{P}(W_R)$$

because they say that the sample space W_R is countable. However, this is not true and can be proved with Cantor's diagonal argument: it is not possible to put in a one-to-one correspondence the elements of W_R with the set of natural numbers $\mathbb{N}$.

Theorem 5 (From [36, 37])**.** *W_R is uncountable.*

Proof. If the program contains at least one function symbol and one constant, the Herbrand base $\mathcal{B}_P$ is countable. We can thus represent each element of W_R as a countable sequence of Boolean values. Equivalently, we can represent it with a countable sequence of bits $b_1, b_2, b_3, \ldots$

Suppose W_R is countable. Then it is possible to write its element in a list such as

$$
\begin{array}{l}
b_{1,1}, b_{1,2}, b_{1,3}, \ldots \\
b_{2,1}, b_{2,2}, b_{2,3}, \ldots \\
b_{3,1}, b_{3,2}, b_{3,3}, \ldots \\
\ldots
\end{array}
$$

Since W_R is countable, the list should contain all of its elements.

Now, pick element $\neg b_{1,1}, \neg b_{2,2}, \neg b_{3,3}, \ldots$ This element belongs to W_R because it is a countable sequence of Booleans. However, it is not in the list, because it differs from the first element in the first bit, from the second element in the second bit, and so on. So it differs from each element of the list. This is against the hypothesis that the list contains all elements of W_R. Thus, W_R is not countable. □

The sample space of the entire theory is

$$W_P = W_{\mathrm{X}} \times W_R$$

and the event space of the entire theory is

$$\Omega_P = \Omega_{\mathrm{X}} \otimes \Omega_R.$$

The probability measure μ_{X} is extended to a probability measure of the entire theory μ_P by observing that knowing which constraints are true uniquely determines the truth value of all atoms in the entire theory.

An element w_{X} of the sample space W_{X} uniquely determines which constraints are true: we assume that the logic theory $R \cup \mathit{satisfiable}(w_{\mathrm{X}})$ has a unique well-founded model which we denote by $\mathit{WFM}(w_{\mathrm{X}})$.

The probability measure on the entire theory P's event space is defined as

$$\mu_P(\omega) = \mu_{\mathrm{X}}(\{w_{\mathrm{X}} \mid (w_{\mathrm{X}}, w_R) \in \omega, \mathit{WFM}(w_{\mathrm{X}}) \models w_R\}).$$

The probability of a query q is defined as

$$P(q) = \mu_P(\{(w_{\mathrm{X}}, w_R) \in W_P \mid w_R \models q\}).$$

The authors of [26] (pp. 11-12) say:

> We further know that the event defined by the equation above is an element of the event space Ω_P, since we do not put any restrictions on values of random variables and the event space concerning the logic atoms is defined as the powerset of the sample space [...] thus each subset of the sample space is in the event space.

Since the event space of the logic atoms cannot be defined as the powerset of the sample space, the fact that $\{(w_{\mathrm{X}}, w_R) \in W_P \mid w_R \models q\}$ is measurable is not obvious and must be proved.

6. PCLP Examples

In this section, we show some examples of PCLP. Discrete and continuous random variables are described by their distribution with facts of the form

$$\mathbf{Variable} \sim \mathit{distribution}$$

where variable names start with an uppercase character and are bold. For example,

$$\mathbf{Time\_comp} \sim \mathit{exp}(1)$$

represents a continuous random variable **Time_comp** that follows an exponential distribution with parameter 1. Moreover, the body of rules can contains special atoms enclosed in square brackets $\langle\ \rangle$, encoding constraints among random variables.

The following two examples are taken from [26]. The first one describes the development of fire on a ship, while the second models the behavior of a consumer.

Example 6 (Fire on a ship [26]). *Suppose a fire breaks out in a compartment of a ship. After 0.75 minutes also the next compartment will be on fire. After 1.25 minutes the fire will breach the hull. With this information, we know for sure that if the fire is under control within 0.75 minutes the ship is saved. This can be represented as:*

$$saved \leftarrow \langle \mathbf{Time\_comp_1} < 0.75 \rangle$$

In detail, the previous line says that the value of the continuous random variable $\mathbf{Time\_comp_1}$ *should be less than 0.75 in order to saved to be true.*
The second compartment is more fragile than the first one, and the fire must be extinguished within 0.625 minutes. However, to reach the second compartment, the fire in the first one must be under control. This means that both fires must be extinguished in 0.75 + 0.625 = 1.375 minutes. In the second compartment four people can work simultaneously, since it is not as isolated as the first one. This means that the fire will be extinguished four times faster. We can encode this situation with:

$$\begin{aligned} saved \leftarrow\ & \langle \mathbf{Time\_comp_1} < 1.25 \rangle, \\ & \langle \mathbf{Time\_comp_1} + 0.25 \cdot \mathbf{Time\_comp_2} < 1.375 \rangle \end{aligned}$$

We also suppose that both time durations to extinguish the fire are exponentially distributed:

$$\mathbf{Time\_comp_1} \sim exp(1)$$

$$\mathbf{Time\_comp_2} \sim exp(1)$$

Given these time constraints and these distributions, we want to know the probability that the ship is saved, i.e., P(saved).

Example 7 (Fruit selling [26]). *We want to compute the likelihood of a consumer buying a certain fruit. The price of the fruit depends on its yield, which is modeled with a Gaussian distribution. For apples and bananas, we have:*

$$\mathbf{Yield}(apple) \sim gaussian(12000.0, 1000.0)$$
$$\mathbf{Yield}(banana) \sim gaussian(10000.0, 1500.0)$$

The government may or may not support the market, this is modeled with discrete random variables:

$$\mathbf{Support}(apple) \sim \{0.3 : yes,\ 0.7 : no\}$$
$$\mathbf{Support}(banana) \sim \{0.5 : yes,\ 0.5 : no\}$$

The basic price is computed on the basis of the yield with a linear function:

$$basic\_price(apple) \leftarrow$$
$$\langle \mathbf{Basic\_price}(apple) = 250 - 0.007 \times \mathbf{Yield}(apple)\rangle$$
$$basic\_price(banana) \leftarrow$$
$$\langle \mathbf{Basic\_price}(banana) = 200 - 0.006 \times \mathbf{Yield}(banana)\rangle$$

Constraints of the form $\langle \mathbf{Variable} = Expression\rangle$ *are special as they give a name to an expression involving random variables that can be reused afterwards in other constraints. In fact, we do not have to specify a density for* **Variable** *as its density is completely determined by that of the variables in Expression.*

The actual price is computed from the basic price by raising it by a fixed amount in case of government support:

$$price(Fruit) \leftarrow basic\_price(Fruit),$$
$$\langle \mathbf{Price}(Fruit) = \mathbf{Basic\_price}(Fruit) + 50\rangle, \langle \mathbf{Support}(Fruit) = yes\rangle$$
$$price(Fruit) \leftarrow basic\_price(Fruit),$$
$$\langle \mathbf{Price}(Fruit) = \mathbf{Basic\_price}(Fruit)\rangle, \langle \mathbf{Support}(Fruit) = no\rangle$$

Note that variable Fruit is not bold, since it is a logical variable, and not a random variable.

A customer buys a certain fruit provided that its price is below a maximum:

$$buy(Fruit) \leftarrow price(Fruit), \langle \mathbf{Price}(Fruit) \leqslant \mathbf{Max\_price}(Fruit)\rangle$$

The maximum price follows a gamma distribution:

$$\mathbf{Max\_price}(apple) \sim \Gamma(10.0, 18.0)$$
$$\mathbf{Max\_price}(banana) \sim \Gamma(12.0, 10.0)$$

We can now ask for the probability of the customer of buying a certain fruit, P(buy(apple)) or P(buy(banana)).

The previous two examples illustrate the expressive power of PCLP. However, they do not contain function symbols, so the set of random variables is finite. A semantics for such programs was given in [16]. The following two examples use integers that are representable only by using function symbols (for example, 0 for 0, $s(0)$ for 1, $s(s(0))$ for 2, ...).

Example 8 (Gambling)**.** *Consider a gambling game that involves spinning a roulette wheel and drawing a card from a deck. The player repeatedly spins the wheel and draws a card. The card is reinserted in the deck after each play. The player records the position of the axis of the wheel when it stops, i.e., the angle it creates with the geographic east. If the player draws a red card the game ends, otherwise he keeps playing. The angle of the wheel and the color of the card define four available prizes. In particular, prize a if the card is black and the angle is less than* π*, prize b if the card is black and the angle is greater than* π*, prize c if the card is red and the angle is less than* π *and prize d otherwise. The angle of the wheel can be described with an uniform distribution in* $[0, 2\pi)$ *and the color of the card with a Bernoulli distribution with* $P(red) = P(black) = 0.5$*.*

$$
\begin{aligned}
&\mathbf{Card}(\_) \sim \{red : 0.5,\ black : 0.5\}\\
&\mathbf{Angle}(\_) \sim uniform(0, 2\pi)\\
&prize(0, a) \leftarrow \langle \mathbf{Card}(0) = black\rangle, \langle \mathbf{Angle}(0) < \pi\rangle\\
&prize(0, b) \leftarrow \langle \mathbf{Card}(0) = black\rangle, \langle \mathbf{Angle}(0) \geqslant \pi\rangle\\
&prize(0, c) \leftarrow \langle \mathbf{Card}(0) = red\rangle, \langle \mathbf{Angle}(0) < \pi\rangle\\
&prize(0, d) \leftarrow \langle \mathbf{Card}(0) = red\rangle, \langle \mathbf{Angle}(0) \geqslant \pi\rangle\\
&prize(s(X), a) \leftarrow prize(X), \langle \mathbf{Card}(X) = black\rangle,\\
&\quad \langle \mathbf{Card}(s(X)) = black\rangle, \langle \mathbf{Angle}(s(X)) < \pi\rangle\\
&prize(s(X), b) \leftarrow prize(X), \langle \mathbf{Card}(X) = black\rangle,\\
&\quad \langle \mathbf{Card}(s(X)) = black\rangle, \langle \mathbf{Angle}(s(X)) \geqslant \pi\rangle\\
&prize(s(X), c) \leftarrow prize(X), \langle \mathbf{Card}(X) = black\rangle,\\
&\quad \langle \mathbf{Card}(s(X)) = red\rangle, \langle \mathbf{Angle}(s(X)) < \pi\rangle\\
&prize(s(X), d) \leftarrow prize(X), \langle \mathbf{Card}(X) = black\rangle,\\
&\quad \langle \mathbf{Card}(s(X)) = red\rangle, \langle \mathbf{Angle}(s(X)) \geqslant \pi\rangle\\
&at\_least\_once\_prize\_a \leftarrow prize(X, a)\\
&never\_prize\_a \leftarrow \sim at\_least\_once\_prize\_a
\end{aligned}
$$

We can ask for the probability that the player wins at least one time prize a with

$P(at\_least\_once\_prize\_a)$. Similarly, we can ask the probability that the player never wins price a with $P(never\_prize\_a)$.

Example 9 (Hybrid Hidden Markov Model)**.** *A Hybrid Hidden Markov Model (Hybrid HMM) combines a Hidden Markov Model (HMM, with discrete states) and a Kalman Filter (with continuous states). At every integer time point t, the system is in a state $[\mathbf{S}(t), \mathbf{Type}(t)]$ which is composed of a discrete random variable $\mathbf{Type}(t)$, taking values in $\{a, b\}$, and a continuous variable $\mathbf{S}(t)$ taking values in $\mathbb{R}$. At time t it emits one value $\mathbf{V}(t) = \mathbf{S}(t) + \mathbf{Obs\_err}(t)$, where $\mathbf{Obs\_err}(t)$ is an error that follows a probability distribution that does not depend on time but depends on $\mathbf{Type}(t)$, a or b. At time $t' = t + 1$, the systems transitions to a new state $[\mathbf{S}(t'), \mathbf{Type}(t')]$, with $\mathbf{S}(t') = \mathbf{S}(t) + \mathbf{Trans\_err}(t)$ where $\mathbf{Trans\_err}(t)$ is also an error that follows a probability distribution that does not depend on time but depends on $\mathbf{Type}(t)$. $\mathbf{Type}(t')$ depends on $\mathbf{Type}(t)$. The state at time 0 is described by random variable* **Init**. *Here, all the random variables except* **Init** *are indexed by the integer time.*

$$
\begin{aligned}
&ok \leftarrow kf(2), \langle \mathbf{V}(2) > 2 \rangle \\
&kf(N) \leftarrow \langle \mathbf{S}(0) = \mathbf{Init} \rangle, \langle \mathbf{Type}(0) = \mathbf{TypeInit} \rangle, kf\_part(0, N) \\
&kf\_part(I, N) \leftarrow I < N, NextI \ is \ I + 1, \\
&\quad trans(I, NextI), emit(I), \\
&\quad kf\_part(NextI, N) \\
&kf\_part(N, N) \leftarrow N \neq 0 \\
&trans(I, NextI) \leftarrow \\
&\quad \langle \mathbf{Type}(I) = a \rangle, \langle \mathbf{S}(NextI) = \mathbf{S}(I) + \mathbf{Trans\_err\_a}(I) \rangle, \\
&\quad \langle \mathbf{Type}(NextI) = \mathbf{Type\_a}(NextI) \rangle \\
&trans(I, NextI) \leftarrow \\
&\quad \langle \mathbf{Type}(I) = b \rangle, \langle \mathbf{S}(NextI) = \mathbf{S}(I) + \mathbf{Trans\_err\_b}(I) \rangle \\
&\quad \langle \mathbf{Type}(NextI) = \mathbf{Type\_b}(NextI) \rangle \\
&emit(S, I, V) \leftarrow \\
&\quad \langle \mathbf{Type}(I) = a \rangle, \langle \mathbf{V}(I) = \mathbf{S}(I) + \mathbf{Obs\_err\_a}(I) \rangle \\
&emit(S, I, V) \leftarrow \\
&\quad \langle \mathbf{Type}(I) = b \rangle, \langle \mathbf{V}(I) = \mathbf{S}(I) + \mathbf{Obs\_err\_b}(I) \rangle \\
&\mathbf{Init} \sim gaussian(0, 1) \\
&\mathbf{Trans\_err\_a}(\_) \sim gaussian(0, 2) \\
&\mathbf{Trans\_err\_b}(\_) \sim gaussian(0, 4) \\
&\mathbf{Obs\_err\_a}(\_) \sim gaussian(0, 1) \\
&\mathbf{Obs\_err\_b}(\_) \sim gaussian(0, 3) \\
&\mathbf{TypeInit} \sim \{a : 0.4, b : 0.6\} \\
&\mathbf{Type\_a}(I) \sim \{a : 0.3, b : 0.7\} \\
&\mathbf{Type\_b}(I) \sim \{a : 0.7, b : 0.3\}
\end{aligned}
$$

7. A New Semantics for Probabilistic Constraint Logic Programming

This section represents the core of our work. Here we provide a new semantics for PCLP and prove that it is well-defined, i.e., each query can be assigned a probability. In giving a new semantics for PCLP, we consider discrete and continuous random variables separately. Discrete random variables are encoded using probabilistic facts as in ProbLog. With Boolean probabilistic facts it is possible to encode any discrete random variable: if the variable V has n values $v_1, \ldots, v_n$, we can use $n-1$ ProbLog probabilistic facts f_i and encode that $V = v_i$ for $i = 1, \ldots, n-1$ with the conjunction

$$\sim f_1, \ldots, \sim f_{i-1}, f_i$$

and $V = v_n$ with the conjunction

$$\sim f_1, \ldots, \sim f_{n-1}$$

with the probability π_i of fact f_i given by

$$\pi_i = \frac{\Pi_i}{\prod_{j=1}^{i-1}(1-\pi_j)}$$

where Π_i is the probability of value v_i of variable V.

We consider that a program P in PCLP is composed by a set of *rules* R, a set of Boolean *probabilistic facts* F and a countable set of continuous random variables X. The rules define the truth value of the atoms in the Herbrand base of the program given the values of the random variables. Let $\mathrm{X} = \{\mathrm{X}_1, \mathrm{X}_2, \ldots\}$ be the countable set of continuous random variables. Each random variable X_i has an associated range $Range_i$ that can be $\mathbb{R}$ or $\mathbb{R}^n$.

The sample space for the continuous variables is defined as $W_\mathrm{X} = Range_1 \times Range_2 \times \ldots$ As shown in Section 5, the probability spaces of individual variables generate an infinite dimensional probability space $(W_\mathrm{X}, \Omega_\mathrm{X}, \mu_\mathrm{X})$.

We can now define a *Probabilistic Constraint Logic Theory.*

Definition 10 (Probabilistic constraint logic theory)**.** *A* probabilistic constraint logic theory P *is a tuple* $(\mathrm{X}, W_\mathrm{X}, \Omega_\mathrm{X}, \mu_\mathrm{X}, Constr, R, F)$ *where:*

- X *is a countable set of continuous random variables* $\{\mathrm{X}_1, \mathrm{X}_2, \ldots\}$*. Each random variable* X_i *has a non-empty range* $Range_i$*;*
- $W_\mathrm{X} = Range_1 \times Range_2 \times \ldots$ *is the sample space;*
- Ω_X *is the event space;*

- μ_{X} *is a probability measure, i.e.,* $(W_{\mathrm{X}}, \Omega_{\mathrm{X}}, \mu_{\mathrm{X}})$ *is a probability space;*

- *Constr is a set of constraints closed under conjunction, disjunction and negation such that* $\forall \varphi \in Constr$, $CSS(\varphi) \in \Omega_{\mathrm{X}}$, *i.e., such that* $CSS(\varphi)$ *is measurable for all* φ*;*

- R *is a set of rules with logical constraints of the form:*

  $h \leftarrow l_1, \ldots, l_n, \langle \varphi_1(\mathrm{X}) \rangle, \ldots, \langle \varphi_m(\mathrm{X}) \rangle$ *where* l_i *is a literal for* $i = 1, \ldots, n$, $\varphi_j \in Constr$ *and* $\langle \varphi_j(X) \rangle$ *is called constraint atom for* $j = 1, \ldots, m$*;*

- F *is a set of probabilistic facts.*

Note that our definition differs from Definition 9 since we define X as the set containing continuous random variables only. Moreover, we also introduce a set of discrete probabilistic facts F. That is, we consider separately discrete and continuous random variables. The probabilistic facts of a program form a countable set of Boolean random variables $\mathrm{Y} = \{\mathrm{Y}_1, \mathrm{Y}_2, \ldots\}$ with sample space $W_{\mathrm{Y}} = \{(y_1, y_2, \ldots) \mid y_i \in \{0, 1\}, i \in 1, 2, \ldots\}$. The event space Ω_{Y} is the σ-algebra of set of worlds identified by countable set of countable composite choices. A composite choice $\kappa = \{(f_1, \theta_1, y_1), (f_2, \theta_2, y_2), \ldots\}$ can be interpreted as the assignments $Y_1 = y_1$, $Y_2 = y_2$, ... if the random variable Y_1 is associated to $f_1\theta_1$, Y_2 to $f_2\theta_2$ and so on. The sample space for the entire program is defined as $W_P = W_{\mathrm{X}} \times W_{\mathrm{Y}}$ and the event space Ω_P is the σ-algebra generated by the tensor product of Ω_{X} and Ω_{Y}: $\Omega_P = \Omega_{\mathrm{X}} \otimes \Omega_{\mathrm{Y}} = \sigma(\{\omega_{\mathrm{X}} \times \omega_{\mathrm{Y}} \mid \omega_{\mathrm{X}} \in \Omega_{\mathrm{X}}, \omega_{\mathrm{Y}} \in \Omega_{\mathrm{Y}}\})$.

We indicate with *satisfiable*(w_{X}) the set of all constraints that are satisfiable given a valuation w_{X} of the random variables in X. We say that a world *satisfies* a constraint if the values of the continuous variables in the world satisfy the constraint.

Given a sample $w = (w_{\mathrm{X}}, w_{\mathrm{Y}})$ from W_P, a ground normal logic program P_w is defined by:

- the grounding of the rules whose constraints belong to *satisfiable*(w_{X}), with the constraints removed from the body of the rules;

- the probabilistic facts that are associated to random variables Y_i whose value is 1.

We define the well-founded model $\mathit{WFM}(w)$ of $w \in W_P$ as the well-founded model of P_w, $\mathit{WFM}(P_w)$, and we require that it is two-valued. We call *sound* the programs that satisfy this constraint for each sample w from W_P.

An explanation for an atom (a query) q of a PCLP program is a set of worlds ω_i such that the query is true in every element of the set, i.e., $\forall w \in \omega_i : w \models q$. A covering set of explanation is such that every world in which the query is true belongs to the set. A set $\omega = \bigcup_j \omega_j$ is pairwise incompatible if $\omega_j \cap \omega_k = \varnothing$ for $j \neq k$. The probability of a query can be defined as the measure of a covering set of explanations, $P(q) = \mu(\{w \mid w \models q\})$ where, from Theorem 1, $\mu(w)$ is the product of measures $\mu(w_\mathrm{X})$ and $\mu(w_\mathrm{Y})$.

In the following examples we show how to compute the probability of a query.

Example 10 (Pairwise incompatible covering set of explanations for Example 8)**.** *For Example 8, the extraction of a black card can be represented with* $F1 = \mathit{black}(\_) : 0.5$*. Then,* $(f_1, \theta, 1)$ *means that the card is black and* $(f_1, \theta, 0)$ *means that the card is not black (red). Let us use random variable* Y_i *to represent* $\mathit{black}(s^i(0))$*, with value* $y_i = 1$ *meaning that in round* i *a black card was picked. The query at_least_once_prize_a has the mutually disjoint covering set of explanations*

$$\omega^+ = \omega_0^+ \cup \omega_1^+ \cup \ldots$$

with

$$
\begin{aligned}
\omega_0^+ = \{&(w_1, w_2) \mid w_1 = (x_1, x_2, \ldots), w_2 = (y_1, y_2, \ldots),\\
&x_1 \in [0, \pi], y_1 = 1\}\\
\omega_1^+ = \{&(w_1, w_2) \mid w_1 = (x_1, x_2, \ldots), w_2 = (y_1, y_2, \ldots),\\
&x_1 \in [\pi, 2\pi], y_1 = 1, x_2 \in [0, \pi], y_2 = 1\}\\
&\ldots
\end{aligned}
$$

Similarly, the query never_prize_a has the pairwise incompatible covering set of explanations

$$\omega^- = \omega_0^- \cup \omega_1^- \cup \omega_2^- \cup \omega_3^- \cup \ldots$$

with

$$
\begin{aligned}
\omega_0^- = \{&(w_1, w_2) \mid w_1 = (x_1, x_2, \ldots), w_2 = (y_1, y_2, \ldots),\\
&x_1 \in [0, \pi], y_1 = 0\}\\
\omega_1^- = \{&(w_1, w_2) \mid w_1 = (x_1, x_2, \ldots), w_2 = (y_1, y_2, \ldots),\\
&x_1 \in [\pi, 2\pi], y_1 = 0\}\\
\omega_2^- = \{&(w_1, w_2) \mid w_1 = (x_1, x_2, \ldots), w_2 = (y_1, y_2, \ldots),\\
&x_1 \in [\pi, 2\pi], y_1 = 1, x_2 \in [0, \pi], y_2 = 0\}\\
\omega_3^- = \{\{&(w_1, w_2) \mid w_1 = (x_1, x_2, \ldots), w_2 = (y_1, y_2, \ldots),\\
&x_1 \in [\pi, 2\pi], y_1 = 1, x_2 \in [\pi, 2\pi], y_2 = 0\}\\
&\ldots
\end{aligned}
$$

Example 11 (Probability of the query for Example 8)**.** *For example, consider sets ω_0^+ and ω_0^- from Example 10. From Theorem 1,*

$$\mu(\omega_0^+) = \int_{W_1} \mu_2(\omega^{(1)}(w_1))d\mu_1 = \int_{W_1} \mu_2(\{w_2 \mid (w_1, w_2) \in \omega\})d\mu_1$$

and so

$$\mu(\omega_0^+) = \int_0^\pi \mu_2(\{(y_1, y_2, \ldots) \mid y_1 = 1\}) d\mu_1$$
$$= \int_0^\pi \frac{1}{2} \cdot \frac{1}{2\pi} dx_1 = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

since, for the discrete variables, $\mu_2(\{(y_1, y_2, \ldots) \mid y_1 = 0\}) = \mu_2(\{(y_1, y_2, \ldots) \mid y_1 = 1\}) = 1/2$ *and* μ_1 *is the Lebesgue measure of the set* $[0, \pi]$*. Similarly,*

$$\mu(\omega_0^-) = \int_0^\pi \mu_2(\{(y_1, y_2, \ldots) \mid y_1 = 0\}) d\mu_1$$
$$= \int_0^\pi \frac{1}{2} \cdot \frac{1}{2\pi} dx_1 = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$

From Example 10, the sets ω_i^+ *are pairwise incompatible so measure of* ω^+ *can be computed by summing the measures of* ω_i^+*. Thus, iteratively applying the previous computations, the probability of the query at_least_once_prize_a can be computed as:*

$$P(at\_least\_once\_prize\_a) = \frac{1}{4} + \frac{1}{4} \cdot \frac{1}{4} + \frac{1}{4} \cdot \left(\frac{1}{4} \cdot \frac{1}{4}\right) + \ldots$$
$$= \frac{1}{4} + \frac{1}{4} \cdot \left(\frac{1}{4}\right) + \frac{1}{4} \cdot \left(\frac{1}{4}\right)^2 + \ldots$$
$$= \frac{1}{4} \cdot \frac{1}{1 - \frac{1}{4}} = \frac{1}{4} \cdot \frac{4}{3} = \frac{1}{3}$$

since the sum represents a geometric series. Similarly, for the query never_prize_a, the sets forming ω^- *are pairwise incompatible, so its probability can be computed as*

$$P(never\_prize\_a) = \left(\frac{1}{4} + \frac{1}{4}\right) + \left(\frac{1}{4} + \frac{1}{4}\right) \cdot \frac{1}{4} +$$
$$\left(\frac{1}{4} + \frac{1}{4}\right) \cdot \left(\frac{1}{4} \cdot \frac{1}{4}\right) + \ldots$$
$$= \frac{1}{2} + \frac{1}{2} \cdot \left(\frac{1}{4}\right) + \frac{1}{2} \cdot \left(\frac{1}{4}\right)^2 + \ldots$$
$$= \frac{1}{2} \cdot \frac{1}{1 - \frac{1}{4}} = \frac{1}{2} \cdot \frac{4}{3} = \frac{2}{3}$$

As expected, $P(never\_prize\_a) = 1 - P(at\_least\_once\_prize\_a)$*.*

Example 12 (Pairwise incompatible covering set of explanations for Example 9)**.** *Consider Example 9. The discrete state variable can be represented with* $F1 = type(\_) : P$*. Then,* $(f_1, \theta, 0)$ *means that the filter is of type* a *and* $(f_1, \theta, 1)$ *means that the filter is of type* b*. A covering set of explanations for the query* ok *is:*

$$\omega = \omega_0 \cup \omega_1 \cup \omega_2 \cup \omega_3$$

with

$$\begin{aligned}
\omega_0 = \{(w_1, w_2) \mid \\
& w_1 = (Init, Trans_err_a(0), Trans_err_a(1), Obs_err_a(1), \ldots), \\
& w_2 = (TypeInit, Type(1), \ldots), \\
& Init + Trans_err_a(0) + Trans_err_a(1) + Obs_err_a(1) > 2, \\
& TypeInit = 0, Type(1) = 0\} \\
\omega_1 = \{(w_1, w_2) \mid \\
& w_1 = (Init, Trans_err_a(0), Trans_err_b(1), Obs_err_b(1), \ldots), \\
& w_2 = (TypeInit, Type(1), \ldots), \\
& Init + Trans_err_a(0) + Trans_err_b(1) + Obs_err_b(1) > 2, \\
& TypeInit = 0, Type(1) = 1\} \\
\omega_2 = \{(w_1, w_2) \mid \\
& w_1 = (Init, Trans_err_b(0), Trans_err_a(1), Obs_err_a(1), \ldots), \\
& w_2 = (TypeInit, Type(1), \ldots), \\
& Init + Trans_err_b(0) + Trans_err_a(1) + Obs_err_a(1) > 2, \\
& TypeInit = 1, Type(1) = 0\} \\
\omega_3 = \{(w_1, w_2) \mid \\
& w_1 = (Init, Trans_err_b(0), Trans_err_b(1), Obs_err_b(1), \ldots), \\
& w_2 = (TypeInit, Type(1), \ldots), \\
& Init + Trans_err_b(0) + Trans_err_b(1) + Obs_err_b(1) > 2,
\end{aligned}$$

$TypeInit = 1, Type(1) = 1\}$

Example 13 (Probability of the query for Example 9)**.** *Consider the set* ω_0 *from Example 12. Let us denote discrete random variables Type(i) with* y_i*. So, TypeInit* $= y_0$ *and Type*$(1) = y_1$*. From Theorem 1,*

$$\mu(\omega_0) = \int_{W_1} \mu_2(\omega^{(1)}(w_1))d\mu_1 = \int_{W_1} \mu_2(\{w_2 \mid (w_1, w_2) \in \omega\})d\mu_1.$$

In this example, continuous random variables are independent and normally distributed. Recall that, if $X \sim gaussian(\mu_X, \sigma_X^2)$*,* $Y \sim gaussian(\mu_Y, \sigma_Y^2)$ *and* $Z = X + Y$*, then* $Z \sim gaussian(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$*. We indicate with* $\mathcal{N}(x, \mu, \sigma^2)$ *the Gaussian pdf with mean* μ *and variance* σ^2*. We have:*

$$\begin{aligned}
\mu(\omega_0) &= \int_{-\infty}^{2} \mu_2(\{(y_1, y_2, \ldots) \mid y_1 = 0, y_2 = 0\})d\mu_1 \\
&= \int_{-\infty}^{2} 0.4 \cdot 0.3 \cdot \mathcal{N}(x, 0, 1 + 2 + 2 + 1)dx = 0.12 \cdot 0.207 = 0.0248.
\end{aligned}$$

The computation is similar for ω_1*,* ω_2 *and* ω_3*. The probability of* ω *can be computed as:*

$$P(\omega) = \mu(\omega_0) + \mu(\omega_1) + \mu(\omega_2) + \mu(\omega_3) = 0.25.$$

We now want to show that every sound program is well-defined, i.e., each query can be assigned a probability. In the following part of the section we consider only ground programs. This is not a restriction since they may be the result of the grounding of a program also with function symbols, and so they can be countably infinite.

Definition 11 (Parameterized two-valued interpretations)**.** *Given a ground probabilistic constraint logic program* P *with Herbrand base* $\mathcal{B}_P$*, a* parameterized positive two-valued interpretation *Tr is a set of pairs* (a, ω_a) *with* $a \in \mathcal{B}_P$ *and* $\omega_a \in \Omega_P$*. Similarly, a* parameterized negative two-valued interpretation *Fa is a set of pairs* $(a, \omega_{\sim a})$ *with* $a \in atoms$ *and* $\omega_{\sim a} \in \Omega_P$*.*

Parameterized two-valued interpretations form a complete lattice where the partial order is defined as $I \leqslant J$ if $\forall (a, \omega_a) \in I, (a, \theta_a) \in J : \omega_a \subseteq \theta_a$. For a set T of parameterized two-valued interpretations, the least upper bound and greatest lower bound always exist and are respectively

$$\mathrm{lub}(T) = \{(a, \bigcup_{I \in T, (a,\omega_a) \in I} \omega_a) \mid a \in \mathcal{B}_P\}$$

and

$$\mathrm{glb}(T) = \{(a, \bigcap_{I \in T, (a,\omega_a) \in I} \omega_a) \mid a \in \mathcal{B}_P\}.$$

The top element $\top$ is

$$\{(a, W_\mathrm{X} \times W_\mathrm{Y}) \mid a \in \mathcal{B}_P\}$$

and the bottom element $\bot$ is

$$\{(a, \varnothing) \mid a \in \mathcal{B}_P\}.$$

Definition 12 (Parameterized three-valued interpretations)**.** *Given a ground probabilistic constraint logic program P with Herbrand base $\mathcal{B}_P$, a* parameterized three-valued interpretation $\mathcal{I}$ *is a set of triples $(a, \omega_a, \omega_{\sim a})$ with $a \in \mathcal{B}_P$, $\omega_a \in \Omega_P$ and $\omega_{\sim a} \in \Omega_P$. A parameterized three-valued interpretation $\mathcal{I}$ is* consistent *if $\forall (a, \omega_a, \omega_{\sim a}) \in \mathcal{I} : \omega_a \cap \omega_{\sim a} = \varnothing$.*

Parameterized three-valued interpretations form a complete lattice where the partial order is defined as $I \leqslant J$ if $\forall (a, \omega_a, \omega_{\sim a}) \in I, (a, \theta_a, \theta_{\sim a}) \in J : \omega_a \subseteq \theta_a$ and $\omega_{\sim a} \subseteq \theta_{\sim a}$. For a set T of parameterized three-valued interpretations, the least upper bound and greatest lower bound always exist and are respectively

$$\mathrm{lub}(T) = \{(a, \bigcup_{I \in T, (a,\omega_a,\omega_{\sim a}) \in I} \omega_a, \bigcup_{I \in T, (a,\omega_a,\omega_{\sim a}) \in I,} \omega_{\sim a}) \mid a \in \mathcal{B}_P\}$$

and

$$\mathrm{glb}(T) = \{(a, \bigcap_{I \in T, (a,\omega_a,\omega_{\sim a}) \in I} \omega_a, \bigcap_{I \in T, (a,\omega_a,\omega_{\sim a}) \in I} \omega_{\sim a}) \mid a \in \mathcal{B}_P\}.$$

The top element $\top$ is

$$\{(a, W_\mathrm{X} \times W_\mathrm{Y}, W_\mathrm{X} \times W_\mathrm{Y}) \mid a \in \mathcal{B}_P\}$$

and the bottom element $\bot$ is

$$\{(a, \varnothing, \varnothing) \mid a \in \mathcal{B}_P\}.$$

Definition 13 ($OpTrueP_{\mathcal{I}}^{P}(Tr)$ and $OpFalseP_{\mathcal{I}}^{P}(Fa)$)**.** *For a ground probabilistic constraint logic program P with rules R and facts F, a parameterized two-valued positive interpretation Tr with pairs (a, θ_a), a parameterized two-valued negative interpretation Fa with pairs $(a, \theta_{\sim a})$ and a parameterized three-valued interpretation $\mathcal{I}$ with triplets $(a, \omega_a, \omega_{\sim a})$, we define $OpTrueP_{\mathcal{I}}^{P}(Tr) = \{(a, \gamma_a) \mid a \in \mathcal{B}_P\}$ where*

$$\gamma_a = \begin{cases} W_{\mathrm{X}} \times \omega_{\{\{(a,\varnothing,1)\}\}} & \text{if } a \in F \\ \bigcup_{a \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,c_m,\varphi_1,\ldots,\varphi_l \in R}((\theta_{b_1} \cup \omega_{b_1}) \cap \ldots & \\ \cap(\theta_{b_n} \cup \omega_{b_n}) \cap \omega_{\sim c_1} \cap \ldots \cap \omega_{\sim c_m} & \text{if } a \in \mathcal{B}_P \backslash F \\ \cap CSS(\varphi_1) \times W_{\mathrm{Y}} \cap \ldots \cap CSS(\varphi_l) \times W_{\mathrm{Y}}) & \end{cases}$$

and $OpFalseP_{\mathcal{I}}^{P}(Fa) = \{(a, \gamma_{\sim a}) \mid a \in \mathcal{B}_P\}$ where

$$\gamma_{\sim a} = \begin{cases} W_{\mathrm{X}} \times \omega_{\{\{(a,\varnothing,0)\}\}} & \text{if } a \in F \\ \bigcap_{a \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,c_m,\varphi_1,\ldots,\varphi_l \in R}((\theta_{\sim b_1} \cap \omega_{\sim b_1}) \cup \ldots & \\ \cup(\theta_{\sim b_n} \cap \omega_{\sim b_n}) \cup \omega_{c_1} \cup \ldots \cup \omega_{c_m} & \text{if } a \in \mathcal{B}_P \backslash F \\ \cup(W_{\mathrm{X}} \backslash CSS(\varphi_1)) \times W_{\mathrm{Y}} \cup \ldots \cup (W_{\mathrm{X}} \backslash CSS(\varphi_l)) \times W_{\mathrm{Y}} & \end{cases}$$

Proposition 1 (Monotonicity of $OpTrueP_{\mathcal{I}}^{P}$ and $OpFalseP_{\mathcal{I}}^{P}$)**.** *$OpTrueP_{\mathcal{I}}^{P}$ and $OpFalseP_{\mathcal{I}}^{P}$ are monotonic.*

Proof. Here we only consider $OpTrueP_{\mathcal{I}}^{P}$, since the proof for $OpFalseP_{\mathcal{I}}^{P}$ can be constructed in a similar way. We have to prove that if $Tr_1 \leqslant Tr_2$ then $OpTrueP_{\mathcal{I}}^{P}(Tr_1) \leqslant OpTrueP_{\mathcal{I}}^{P}(Tr_2)$. By definition, $Tr_1 \leqslant Tr_2$ means that

$$\forall (a, \omega_a) \in Tr_1, (a, \theta_a) \in Tr_2 : \omega_a \subseteq \theta_a.$$

Let (a, ω'_a) be the elements of $OpTrueP_{\mathcal{I}}^{P}(Tr_1)$ and (a, θ'_a) the elements of $OpTrueP_{\mathcal{I}}^{P}(Tr_2)$. To prove the monotonicity, we have to prove that $\omega'_a \subseteq \theta'_a$

If $a \in F$ then $\omega'_a = \theta'_a = W_{\mathrm{X}} \times \omega_{\{\{(a,\varnothing,1)\}\}}$. If $a \in \mathcal{B}_P \backslash F$, then ω'_a and θ'_a have the same structure. Since $\forall b \in \mathcal{B}_P : \omega_b \subseteq \theta_b$, then $\omega'_a \subseteq \theta'_a$.

□

$OpTrueP_{\mathcal{I}}^{P}$ and $OpFalseP_{\mathcal{I}}^{P}$ are monotonic so they both have a least fixpoint and a greatest fixpoint.

Definition 14 (Iterated fixed point for probabilistic constraint logic programs)**.** *For a ground probabilistic constraint logic program P, and a parameterized three-valued interpretation $\mathcal{I}$, let $IFPCP^P(\mathcal{I})$ be defined as*

$$IFPCP^P(\mathcal{I}) = \{(a, \omega_a, \omega_{\sim a}) \mid (a, \omega_a) \in \text{lfp}(OpTrueP^P_{\mathcal{I}}),$$
$$(a, \omega_{\sim a}) \in \text{gfp}(OpFalseP^P_{\mathcal{I}})\}.$$

Proposition 2 (Monotonicity of $IFPCP^P$)**.** *$IFPCP^P$ is monotonic.*

Proof. As above, we have to prove that, if $\mathcal{I}_1 \leqslant \mathcal{I}_2$, then $IFPCP^P(\mathcal{I}_1) \leqslant IFPCP^P(\mathcal{I}_2)$. By definition, $\mathcal{I}_1 \leqslant \mathcal{I}_2$ means that

$$\forall (a, \omega_a, \omega_{\sim a}) \in I_1, (a, \theta_a, \theta_{\sim a}) \in I_2 : \omega_a \subseteq \theta_a, \omega_{\sim a} \subseteq \theta_{\sim a}.$$

Let $(a, \omega'_a, \omega'_{\sim a})$ be the elements of $IFPCP^P(\mathcal{I}_1)$ and $(a, \theta'_a, \theta'_{\sim a})$ the elements of $IFPCP^P(\mathcal{I}_2)$. We have to prove that $\omega'_a \subseteq \theta'_a$ and $\omega'_{\sim a} \subseteq \theta'_{\sim a}$. This is a direct consequence of the monotonicity of $OpTrueP^P_{\mathcal{I}}$ and $OpFalseP^P_{\mathcal{I}}$ in $\mathcal{I}$, which can be proved as in Proposition 1. □

The monotonicity property ensures that $IFPCP^P$ has a least fixpoint. Let us identify lfp($IFPCP^P$) with $WFMP(P)$. We call *depth* of P the smallest ordinal δ such that $IFPCP^P \uparrow \delta = WFMP(P)$.

Now we prove that $OpTrueP^P_{\mathcal{I}}$ and $OpFalseP^P_{\mathcal{I}}$ are sound.

Lemma 3 (Soundness of $OpTrueP^P_{\mathcal{I}}$)**.** *For a ground probabilistic constraint logic program P with probabilistic facts F, rules R, and a parameterized three-valued interpretation $\mathcal{I}$, denote with θ^α_a the set associated to atom a in $OpTrueP^P_{\mathcal{I}} \uparrow \alpha$. For every atom a, world w and iteration α, the following holds:*

$$w \in \theta^\alpha_a \rightarrow WFM(w \mid \mathcal{I}) \vDash a$$

where $w \mid \mathcal{I}$ is obtained by adding to w the atoms a for which $(a, \omega_a, \omega_{\sim a}) \in \mathcal{I}$ and $w \in \omega_a$, and by removing all the rules with a in the head for which $(a, \omega_a, \omega_{\sim a}) \in \mathcal{I}$ and $w \in \omega_{\sim a}$.

Proof. We prove the lemma by transfinite induction (see Appendix B for its definition): we assume that the thesis is true for all ordinals $\beta < \alpha$ and we prove it for α. We need to consider two cases: α is a successor ordinal and α is a limit ordinal. Consider α a successor ordinal. If $a \in F$ then the statement is easily verified. If $a \notin F$ consider $w \in \theta_a^\alpha$ where

$$\begin{aligned}\theta_a^\alpha = \bigcup_{a \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,c_m,\varphi_1,\ldots,\varphi_l \in R} & ((\theta_{b_1}^{\alpha-1} \cup \omega_{b_1}) \cap \ldots \\ & \cap (\theta_{b_n}^{\alpha-1} \cup \omega_{b_n}) \cap \omega_{\sim c_1} \cap \ldots \cap \omega_{\sim c_m} \\ & \cap CSS(\varphi_1) \times W_{\mathrm{X}} \cap \ldots \cap CSS(\varphi_l) \times W_{\mathrm{X}}).\end{aligned}$$

This means that there is a rule $a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, c_m, \varphi_1, \ldots, \varphi_l \in R$ such that $w \in \theta_{b_i}^{\alpha-1} \cup \omega_{b_i}$ for $i = 1, \ldots, n$, $w \in \omega_{\sim c_j}$ for $j = 1 \ldots, m$ and $w \models \varphi_k$ for $k = 1, \ldots, l$. By the inductive assumption and because of how $w \mid \mathcal{I}$ is built, $WFM(w \mid \mathcal{I}) \models b_i$, $WFM(w \mid \mathcal{I}) \models \sim c_j$ and $w \models \varphi_k$ so $WFM(w \mid \mathcal{I}) \models a$.

Consider now α a limit ordinal. Then,

$$\theta_a^\alpha = \mathrm{lub}(\{\theta_a^\beta \mid \beta < \alpha\}) = \bigcup_{\beta < \alpha} \theta_a^\beta.$$

If $w \in \theta_a^\alpha$ then there must exist a $\beta < \alpha$ such that $w \in \theta_a^\beta$. By the inductive assumption the hypothesis holds. □

Lemma 4 (Soundness of $OpFalseP_{\mathcal{I}}^P$)**.** *For a ground probabilistic constraint logic program P with probabilistic facts F and rules R, and a parameterized three-valued interpretation $\mathcal{I}$, denote with $\theta_{\sim a}^\alpha$ the set associated with atom a in the operator $OpFalseP_{\mathcal{I}}^P \downarrow \alpha$. For every atom a, world w and iteration α, the following holds:*

$$w \in \theta_{\sim a}^\alpha \rightarrow WFM(w \mid \mathcal{I}) \models \sim a$$

where $w \mid \mathcal{I}$ is built as in Lemma 3.

Proof. Similar to the proof of Lemma 3. □

We now introduce two lemmas needed to prove the soundness of $IFPCP^P$.

Lemma 5 (Partial evaluation, Lemma 6 from [37]). *For a ground normal logic program P and a three-valued interpretation $\mathcal{I} = \langle I_T, I_F \rangle$ such that $\mathcal{I} \leqslant WFM(P)$, define $P||\mathcal{I}$ as the program obtained from P by adding all atoms $a \in I_T$ and by removing all rules with atoms $a \in I_F$ in the head. Then $WFM(P) = WFM(P||\mathcal{I})$.*

Lemma 6 (Model equivalence). *Given a ground probabilistic constraint logic program P, for every world w and iteration α, the following holds:*

$$WFM(w) = WFM(w \mid IFPCP^P \uparrow \alpha).$$

Proof. Let $(a, \omega_a^\alpha, \omega_{\sim a}^\alpha)$ be the elements of $IFPCP^P \uparrow \alpha$. Consider a three-valued interpretation $\mathcal{I}_\alpha = \langle I_T, I_F \rangle$ with $I_T = \{a \mid w \in \omega_a^\alpha\}$ and $I_F = \{a \mid w \in \omega_{\sim a}^\alpha\}$. Then, $\forall a \in I_T,\ WFM(w) \vDash a$ and $\forall a \in I_F,\ WFM(w) \vDash \sim a$. Therefore $\mathcal{I}_\alpha \leqslant WFM(w)$.

Since $w \mid IFPCP^P \uparrow \alpha = w||\mathcal{I}_\alpha$, by Lemma 5

$$WFM(w) = WFM(w||\mathcal{I}_\alpha) = WFM(w \mid IFPCP^P \uparrow \alpha).$$

□

Now we can prove the soundness and completeness of $IFPCP^P$.

Lemma 7 (Soundness of $IFPCP^P$). *For a ground probabilistic constraint logic program P with probabilistic facts F and rules R, denote with ω_a^α and $\omega_{\sim a}^\alpha$ the formulas associated with atom a in $IFPCP^P \uparrow \alpha$. For every atom a, world w and iteration α, the following holds:*

$$w \in \omega_a^\alpha \rightarrow WFM(w) \vDash a \tag{3}$$

$$w \in \omega_{\sim a}^\alpha \rightarrow WFM(w) \vDash \sim a \tag{4}$$

Proof. The proof is a consequence of Lemma 6: $w \in \omega_a^\alpha$ means that a is a fact in $w \mid IFPCP^P \uparrow \alpha$. Thus, $WFM(w \mid IFPCP^P \uparrow \alpha) \vDash a$ and $WFM(w) \vDash a$.

Similarly, $w \in \omega_{\sim a}^\alpha$ means that there are no rules for a in $w \mid IFPCP^P \uparrow \alpha$, so $WFM(w \mid IFPCP^P \uparrow \alpha) \vDash \sim a$ and $WFM(w) \vDash \sim a$.

□

Lemma 8 (Completeness of $IFPCP^P$)**.** *For a ground probabilistic constraint logic program P with probabilistic facts F and rules R, let ω_a^α and $\omega_{\sim a}^\alpha$ be the sets associated with atom a in $IFPCP^P \uparrow \alpha$. For every atom a, world w and iteration α, we have:*

$$a \in IFP^w \uparrow \alpha \to w \in \omega_a^\alpha$$
$$\sim a \in IFP^w \uparrow \alpha \to w \in \omega_{\sim a}^\alpha$$

Proof. We prove it by double transfinite induction. If α is a successor ordinal, assume that

$$a \in IFP^w \uparrow (\alpha - 1) \to w \in \omega_a^{\alpha-1}$$
$$\sim a \in IFP^w \uparrow (\alpha - 1) \to w \in \omega_{\sim a}^{\alpha-1}$$

Let us perform transfinite induction on the iterations of $OpTrue^w_{IFP^w \uparrow (\alpha-1)}$ and $OpFalse^w_{IFP^w \uparrow (\alpha-1)}$. Consider a successor ordinal δ and assume that

$$a \in OpTrue^w_{IFP^w \uparrow (\alpha-1)} \uparrow (\delta - 1) \to w \in \omega_a^{\delta-1}$$
$$\sim a \in OpFalse^w_{IFP^w \uparrow (\alpha-1)} \downarrow (\delta - 1) \to w \in \theta_{\sim a}^{\delta-1}$$

where $(a, \omega_a^{\delta-1})$ are the elements of $OpTrue^p_{IFPCP^P \uparrow \alpha-1} \uparrow (\delta - 1)$ and $(a, \theta_{\sim a}^{\delta-1})$ are the elements of $OpFalse^p_{IFPCP^P \uparrow \alpha-1} \downarrow (\delta - 1)$. We now prove that

$$a \in OpTrue^w_{IFP^w \uparrow (\alpha-1)} \uparrow \delta \to w \in \omega_a^{\delta}$$
$$\sim a \in OpFalse^w_{IFP^w \uparrow (\alpha-1)} \downarrow \delta \to w \in \theta_{\sim a}^{\delta}$$

Consider an atom a. If $a \in F$, the previous statement can be easily proved. Otherwise, $a \in OpTrue^w_{IFP^w \uparrow (\alpha-1)} \uparrow \delta$ means that there is a rule $a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, c_m, \varphi_1, \ldots, \varphi_l$ such that for all $i = 1, \ldots, n$,

$$b_i \in OpTrue^w_{IFP^w \uparrow (\alpha-1)} \uparrow (\delta - 1) \vee b_i \in IFP^w \uparrow (\alpha - 1)$$

for all $j = 1, \ldots, m$, $\sim c_j \in IFP^w \uparrow (\alpha - 1)$ and for all $k = 1, \ldots, l$, $\varphi_k(w) = true$. For the inductive hypothesis, $\forall i : w \in \omega_{b_i}^{\delta-1} \vee w \in \omega_{b_i}^{\alpha-1}$ and $\forall j : w \in \omega_{\sim c_j}^{\alpha-1}$ so $w \in \omega_a^\delta$. The proof is similar for $\sim a$.

Consider now δ a limit ordinal, so $\omega_a^\delta = \bigcup_{\mu<\delta} \omega_a^\mu$ and $\theta_{\sim a}^\delta = \bigcap_{\mu<\delta} \theta_{\sim a}^\mu$. If $a \in OpTrue^w_{IFP^w\uparrow(\alpha-1)} \uparrow \delta$, then there exists a $\mu < \delta$ such that

$$a \in OpTrue^w_{IFP^w\uparrow(\alpha-1)} \uparrow \mu.$$

For the inductive hypothesis, $w \in \omega_a^\delta$.

If $\sim a \in OpFalse^w_{IFP^w\uparrow(\alpha-1)} \downarrow \delta$, then, for all $\mu < \delta$,

$$\sim a \in OpFalse^w_{IFP^w\uparrow(\alpha-1)} \downarrow \mu.$$

For the inductive hypothesis, $w \in \theta_a^\delta$.

Consider now α a limit ordinal. Then $\omega_a^\alpha = \bigcup_{\beta<\alpha} \omega_a^\beta$ and $\omega_{\sim a}^\alpha = \bigcup_{\beta<\alpha} \omega_{\sim a}^\beta$. If $a \in IFP^w \uparrow \alpha$, then there exists a $\beta < \alpha$ such that $a \in IFP^w \uparrow \beta$. For the inductive hypothesis $w \in \omega_a^\beta$ so $w \in \omega_a^\alpha$. The proof is similar for $\sim a$. $\square$

Now we can prove that $IFPCP^P$ is sound and complete.

Theorem 6 (Soundness and completeness of $IFPCP^P$)**.** *For a sound ground probabilistic constraint logic program P, let ω_a^α and $\omega_{\sim a}^\alpha$ be the formulas associated with atom a in $IFPCP^P \uparrow \alpha$. For every atom a and world w there is an iteration α_0 such that for all $\alpha > \alpha_0$ we have:*

$$w \in \omega_a^\alpha \leftrightarrow WFM(w) \models a \tag{5}$$

$$w \in \omega_{\sim a}^\alpha \leftrightarrow WFM(w) \models \sim a \tag{6}$$

Proof. The $\rightarrow$ direction of equations 5 and 6 is proven in Lemma 7. In the other direction, $WFM(w) \models a$ implies that there exists a α_0 such that $\forall \alpha : \alpha \geqslant \alpha_0 \rightarrow IFP_w \uparrow \alpha \models a$. For Lemma 8, $w \in \omega_a^\alpha$. Similarly, $WFM(w) \models \sim a$ implies that there exists a α_0 such that $\forall \alpha : \alpha \geqslant \alpha_0 \rightarrow IFP^w \uparrow \alpha \models \sim a$. As before, for Lemma 8, $w \in \omega_{\sim a}^\alpha$. $\square$

Now we can prove that every query for every sound program is well-defined.

Theorem 7 (Well-definedness of the distribution semantics)**.** *For a sound ground probabilistic constraint logic program P, for all ground atoms a, $\mu_P(\{w \mid w \in W_P, w \models a\})$ is well-defined.*

Proof. Let ω_a^δ and $\omega_{\sim a}^\delta$ be the sets associated with atom a in $\mathit{IFPCP}^P \uparrow \delta$ where δ denotes the depth of the program. Since IFPCP^P is sound and complete, $\{w \mid w \in W_P, w \vDash a\} = \omega_a^\delta$.

Each iteration of $\mathit{OpTrueP}^P_{\mathit{IFPCP}^P\uparrow\beta}$ and $\mathit{OpFalseP}^P_{\mathit{IFPCP}^P\uparrow\beta}$ for all β generates sets belonging to Ω_P, since the set of rules is countable. So $\mu_P(\{w \mid w \in W_P, w \vDash a\})$ is well-defined. □

In addition, if the program is sound, for all atoms a, $\omega_a^\delta = (\omega_{\sim a}^\delta)^c$ holds, where δ is the depth of the program. Otherwise, there would exists a world w such that $w \notin \omega_a^\delta$ and $w \notin \omega_{\sim a}^\delta$. But w has a two-valued well-founded model, so either $\mathit{WFM}(w) \vDash a$ or $\mathit{WFM}(w) \vDash\sim a$. In the first case $w \in \omega_a^\delta$ and in the latter $w \in \omega_{\sim a}^\delta$, against the hypothesis.

8. A Concrete Syntax for PCLP

In this section, we present `cplint` hybrid programs [37] that provide a concrete syntax for PCLP.

In `cplint` hybrid programs, logical variables are partitioned into two disjoint sets: those that can assume terms as values and those that can assume continuous values. Let us call the first *term* variables and the latter *continuous* variables.

Continuous random variables are encoded with probabilistic facts of the form

$$A : \mathit{Density}$$

where A is an atom with a continuous variable Var as argument and $\mathit{Density}$ is a special atom identifying a probability density on variable Var. For example,

$$p(X) : \mathit{gaussian}(X, 0, 1)$$

indicates that X in atom $p(X)$ is a continuous variable that follows a Gaussian distribution with mean 0 and variance 1. Each predicate p/n has a signature that specifies which arguments hold continuous values. Only these arguments can contain continuous variables. Continuous values (and variables) can appear inside a term build on function symbol f/n. Each function symbol f/n also has a signature that specifies which arguments hold continuous values. Again only these arguments can contain continuous variables.

ProbLog probabilistic facts of the form $p :: f$ can also be encoded as $f : p$ for uniformity with Logic Programs with Annotated Disjunctions [42] and CP-Logic [43].

Atoms in clauses and probabilistic facts can have both term and continuous variables. However, we impose the constraint that in every world of the program, the values taken by term variables in a ground atom for a predicate p/n that is true in the world, uniquely determine the values taken by the continuous variables.

Continuous variables are introduced by probabilistic facts for continuous random variables and by the special predicate $=:= /2$ that is used to define a new variable based on a formula involving existing continuous variables. Constraints are represented by Prolog comparison predicates. The semantics assigns a probability of being true to any ground atom not having continuous values as arguments. Atoms with continuous values have probability 0 as the probability that a continuous random variable takes a specific value is 0. Inference in `cplint` hybrid programs can be performed using MCINTYRE [2, 38], an algorithm based on Monte Carlo inference. See 9.2 for more details.

Let us see some examples of `cplint` hybrid programs.

Example 14 (Gambling in `cplint` hybrid programs)**.** *Example 8 can be expressed in* `cplint` *hybrid programs as*[1]*:*

[1] The example is available in cplint on SWISH at link `http://cplint.eu/e/gambling.pl`

$$
\begin{aligned}
&black(\_) : 0.5.\\
&angle(\_, A) : uniform(A, 0, 2pi).\\
&prize(0, a) \leftarrow black(0), angle(0, A), A < pi.\\
&prize(0, b) \leftarrow black(0), angle(0, A), A >= pi.\\
&prize(0, c) \leftarrow \sim black(0), angle(0, A), A < pi.\\
&prize(0, d) \leftarrow \sim black(0), angle(0, A), A >= pi.\\
&prize(s(X), a) \leftarrow prize(X, \_), black(X),\\
&\quad black(s(X)), angle(s(X), A), A < pi.\\
&prize(s(X), b) \leftarrow prize(X, \_), black(X),\\
&\quad black(s(X)), angle(s(X), A), A >= pi.\\
&prize(s(X), c) \leftarrow prize(X, \_), black(X),\\
&\quad \sim black(s(X)), angle(s(X), A), A < pi.\\
&prize(s(X), d) \leftarrow prize(X, \_), black(X),\\
&\quad \sim black(s(X)), angle(s(X), A), A >= pi.\\
&at\_least\_one\_prize\_a \leftarrow prize(\_, a).\\
&never\_prize\_a \leftarrow \sim at\_least\_once\_prize\_a.
\end{aligned}
$$

Example 15 (Hybrid Hidden Markov Model (Hybrid HMM) in `cplint` hybrid programs)**.** *The Hybrid HMM of example 9 can be expressed in* `cplint` *hybrid programs as* [2]*:*

[2] `http://cplint.eu/e/hhmm.pl`

$$
\begin{array}{l}
init(S) : gaussian(S, 0, 1).\\
trans\_err\_a(\_, E) : gaussian(E, 0, 2).\\
trans\_err\_b(\_, E) : gaussian(E, 0, 4).\\
obs\_err\_a(\_, E) : gaussian(E, 0, 1).\\
obs\_err\_b(\_, E) : gaussian(E, 0, 3).\\
type(0, a) : 0.4; type(0, b) : 0.6.\\
type(I, a) : 0.3; type(I, b) : 0.7 \leftarrow I > 0, PrevI\ is\ I - 1, type(PrevI, a).\\
type(I, a) : 0.7; type(I, b) : 0.3 \leftarrow I > 0, PrevI\ is\ I - 1, type(PrevI, b).\\
ok \leftarrow kf(2, [\_, A], \_), A > 2.\\
kf(N, O, LS) \leftarrow\\
\quad init(S), kf\_part(0, N, S, O, LS).\\
kf\_part(I, N, S, [V|RO], [S|LS]) \leftarrow\\
\quad I < N, NextI\ is\ I + 1,\\
\quad trans(S, I, NextS), emit(NextS, I, V),\\
\quad kf\_part(NextI, N, NextS, RO, LS).\\
kf\_part(N, N, \_S, [], []).\\
trans(S, I, NextS) \leftarrow\\
\quad type(I, a), trans\_err\_a(I, TE), NextS =:= TE + S.\\
trans(S, I, NextS) \leftarrow\\
\quad type(I, b), trans\_err\_b(I, TE), NextS =:= TE + S.\\
emit(S, I, V) \leftarrow\\
\quad type(I, a), obs\_err\_a(I, OE), V =:= S + OE.\\
emit(S, I, V) \leftarrow\\
\quad type(I, b), obs\_err\_b(I, OE), V =:= S + OE.
\end{array}
$$

Here, variables A, S, $NextS$, V, TE and OE are continuous, variables RO and LS are lists of continuous variables and $PrevI$, I, $NextI$ and N are term variables. The probabilistic facts for $trans\_err\_a/2$, $trans\_err\_b/2$ and $obs\_err\_a/2$ and $obs\_err\_b/2$ define a countable set of continuous random variables, one for each term instantiating their first argument.

Example 16 (Fruit selling in `cplint` hybrid programs)**.** *Example 7 can be* *expressed in* `cplint` *hybrid programs as* [3]*:*

$$
\begin{aligned}
&yield(apple, Y) : gaussian(Y, 12000.0, 1000.0).\\
&yield(banana, Y) : gaussian(Y, 10000.0, 1500.0).\\
&support(apple) : 0.3.\\
&support(banana) : 0.5.\\
&basic\_price(apple, B) \leftarrow yield(apple, Y), B =:= 250 - 0.007 \times Y.\\
&basic\_price(banana, B) \leftarrow yield(banana, Y), B =:= 200 - 0.006 \times Y.\\
&price(Fruit, P) \leftarrow basic\_price(Fruit, B), support(Fruit), P =:= B + 50.\\
&price(Fruit, B) \leftarrow basic\_price(Fruit, B), \sim support(Fruit).\\
&buy(Fruit) \leftarrow price(Fruit, P), max\_price(Fruit, M), P \leqslant M.\\
&max\_price(apple, M) : gamma(M, 10.0, 18.0).\\
&max\_price(banana, M) : gamma(M, 12.0, 10.0).
\end{aligned}
$$

Here, variables Y, B, P and M are continuous variables, while $Fruit$ is a term variable.

Example 17 (Gaussian mixture - `cplint`)**.** *A Gaussian mixture model is a way to generate values of a continuous random variable: a discrete random variable is sampled and, depending on the sampled value, a different Gaussian distribution is selected for sampling the value of the continuous variable.*

A Gaussian mixture model with two components can be expressed in `cplint` *hybrid programs as* [4]*:*

$$
\begin{aligned}
&h : 0.6\\
&heads \leftarrow h.\\
&tails \leftarrow \sim h.\\
&g(X) : gaussian(X, 0, 1).\\
&h(X) : gaussian(X, 5, 2).\\
&mix(X) \leftarrow heads, g(X).\\
&mix(X) \leftarrow tails, h(X).\\
&mix \leftarrow mix(X), X > 2.
\end{aligned}
$$

[3] `http://cplint.eu/e/fruit.swinb`

[4] `http://cplint.eu/e/gaussian_mixture.pl`

The argument X of $mix(X)$ follows a distribution that is a mixture of two Gaussians, one with mean 0 and variance 1 with probability 0.6 and one with mean 5 and variance 2 with probability 0.4. We can then ask for the probability of mix.

Here, predicates $g/1$, $h/1$ and $mix/1$ have a single argument which can hold continuous variable. Since there are no term variables, each atom for these predicates in a world univocally determines its argument. For predicate $mix/1$ this is not obvious as there are two clauses for it. However, the two clauses have mutually exclusive bodies, i.e., in each world only one of them is true.

`cplint` hybrid programs can be translated into PCLP by removing the continuous variables from the arguments of predicates and by replacing constraints with their PCLP form.

Term variables that can take integer values can appear as parameters in constraints for the continuous variables.

Example 18 (Gaussian mixture and constraints, from [19])**.** *Consider a factory with two machines, a and b. Each machine produces a widget with a continuous feature. A widget is produced by machine a with probability 0.3 and by machine b with probability* 0.7. *If the widget is produced by machine a, the feature is distributed as a Gaussian with mean 2.0 and variance 1.0. If the widget is produced by machine b, the feature is distributed as a Gaussian with mean 3.0 and variance 1.0. The widget then is processed by a third machine that adds a random quantity to the feature. The quantity is distributed as a Gaussian with mean 0.5 and variance 1.5. This can be encoded by in* `cplint` *hybrid programs as* [5]*:*

[5] `http://cplint.eu/e/widget.pl`

$$machine(a) : 0.3.$$
$$machine(b) \leftarrow\sim machine(a).$$
$$st(a, Z) : gaussian(Z, 2.0, 1.0).$$
$$st(b, Z) : gaussian(Z, 3.0, 1.0).$$
$$pt(Y) : gaussian(Y, 0.5, 1.5).$$
$$widget(X) \leftarrow machine(M), st(M, Z), pt(Y), X =:= Y + Z.$$
$$ok\_widget \leftarrow widget(X), X > 1.0.$$

We can then ask the probability of ok_widget.

Here, X, Z *and* Y *are continuous variables and* M *is a term variable. Since* X *is a continuous variable, in every world there should be a single value for* X *that makes* $widget(X)$ *true. Predicate* $widget/1$ *has a single clause but the clause has two groundings, one for* $M = a$ *and one for* $M = b$*, so in principle there could be two values for* X *in true groundings of* $widget(X)$*. However, the two groundings of the rule have mutually exclusive bodies, as in each world either* $machine(a)$ *is true or* $machine(b)$ *is true but not both.*

The following example shows that the parameters of the distribution atoms can also be taken from the probabilistic atoms.

Example 19 (Estimation of the mean of a Gaussian - `cplint`)**.** *The program*[6]

$$mean(M) : gaussian(M, 1.0, 5.0).$$
$$value(\_, M, X) : gaussian(X, M, 2.0).$$
$$value(I, X) \leftarrow mean(M), value(I, M, X).$$

states that, for an index I*, the continuous variable* X *is sampled from a Gaussian whose variance is 2.0 and whose mean* M *is sampled from a Gaussian with mean 1.0 and variance 5.0.*

This program can be used to estimate the mean of a Gaussian by querying $mean(M)$ *given observations for atom* $value(I, X)$ *for different values of* I*.*

Here, the first argument of $value/3$ *can hold a term variable while its second and third argument can hold a continuous variable. The second argument is used as a parameter in the probability density of the third argument. It is not*

[6] `http://cplint.eu/e/gauss_mean_est.pl`

immediate to see how this program can be translated into a PCLP. In fact, PCLP does not allow specifying the parameters of continuous distributions with values computed by the program. However, we can see continuous variables M and X as specified by a joint density. Since a Gaussian density with a Gaussian mean is still a Gaussian, the joint density will be a multivariate Gaussian.

9. Related Work

In the following section, we review both existing semantics proposals and existing inference algorithms for hybrid programs.

9.1. Semantics

There are other languages that support the definition of hybrid programs, i.e., programs that allow both discrete and continuous random variables.

Hybrid ProbLog [15] extends ProbLog with *continuous probabilistic facts* of the form $(X, \phi) :: f$, where X is a logical variable, called *continuous variable*, that appears in atom f. ϕ is an atom used to specify the continuous distribution (only Gaussian distributions are allowed). A Hybrid ProbLog program $\mathcal{P}$ is composed by a set of definite rules $\mathcal{R}$ and a set of probabilistic facts $\mathcal{F}$ both discrete $\mathcal{F}^d$ (as in ProbLog) and continuous $\mathcal{F}^c$, such that $\mathcal{F} = \mathcal{F}^d \cup \mathcal{F}^c$. The language offers a set of predefined predicates to impose constraints on continuous variables. Consider a continuous variable V and two numeric constants n_1 and n_2. The predefined predicates are: $\mathit{below}(V, n_1)$ and $\mathit{above}(V, n_2)$, that succeed if V is respectively less than and greater than n_2, and $\mathit{ininterval}(n_1, n_2)$, that succeeds if $n_1 \leqslant V \leqslant n_2$.

The set of continuous variables in Hybrid ProbLog is finite since the semantics only allows a finite set of continuous probabilistic facts and no function symbols. We indicate the set of continuous variables as $\mathrm{X} = \{\mathrm{X}_1, \ldots, \mathrm{X}_n\}$. This set is defined by the set of atoms for probabilistic facts $F = \{f_1, \ldots, f_n\}$ where each f_i is ground except for variable X_i. Each continuous variable X_i has an associated probability density $p_i(\mathrm{X}_i)$. An assignment $\mathrm{x} = \{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$ to X defines a substitution $\theta_{\mathrm{x}} = \{\mathrm{X}_1/\mathrm{x}_1, \ldots \mathrm{X}_n/\mathrm{x}_n\}$ and a set of ground facts $F\theta_{\mathrm{x}}$. A world $w_{\sigma,\mathrm{x}}$ is defined as $w_{\sigma,\mathrm{x}} = \mathcal{R} \cup \{f\theta \mid (f, \theta, 1) \in \sigma\} \cup F\theta_{\mathrm{x}}$ where σ is a selection for discrete facts and x is an assignment to continuous variables.

Since all continuous variables are independent, the probability density of an assignment $p(\mathrm{x})$ can be computed as $p(\mathrm{x}) = \prod_{i=1}^{n} p_i(\mathrm{x}_i)$. Moreover, $p(\mathrm{x})$ is a joint probability density over X and thus $p(x)$ and $P(\sigma)$ define a joint probability density over the worlds:

$$p(w_{\sigma,\mathrm{x}}) = p(\mathrm{x}) \prod_{(f_i,\theta,1)\in\sigma} \Pi_i \prod_{(f_i,\theta,0)\in\sigma} 1 - \Pi_i$$

where Π_i is the probability associated to the discrete fact f_i.

Finally, if we consider a ground atom q which is not an atom of a continuous probabilistic fact and the set $S_{\mathcal{P}}$ of all selections over discrete probabilistic facts, $P(q)$ is defined as in the distribution semantics for discrete programs:

$$P(q) = \sum_{\sigma\in\mathcal{S}_{\mathcal{P}}} \int_{\mathrm{x}\in\mathbb{R}^n : w_{\sigma,\mathrm{x}} \models q} p(w_{\sigma,\mathrm{x}}) \; d\mathrm{x}.$$

A key feature is that, if the set $\{(\sigma, \mathrm{x}) \mid \sigma \in S_{\mathcal{P}}, \mathrm{x} \in \mathbb{R}^n : w_{\sigma,\mathrm{x}} \models q\}$ is measurable, then the probability is well-defined.

Moreover, for each instance σ, the set $\{\mathrm{x} \mid \mathrm{x} \in \mathbb{R}^n : w_{\sigma,\mathrm{x}} \models q\}$ can be considered as a n-dimensional interval of the form $I = \times_{i=1}^{n} [a_i, b_i]$ on $\mathbb{R}^n$, where $-\infty$ and $+\infty$ are allowed for a_i and b_i respectively [15]. The probability that $\mathrm{X} \in I$ is then given by

$$P(\mathrm{X} \in I) = \int_{a_1}^{b_1} \ldots \int_{a_n}^{b_n} p(\mathrm{x}) \; d\mathrm{x}.$$

One limitation of Hybrid ProbLog is that it does not allow function symbols and does not allow continuous variables in expressions involving other continuous variables.

Hybrid programs can also be expressed using Distributional Clauses (DC) [16]. DC are definite clauses of the form $h \sim \mathcal{D} \leftarrow b_1, \ldots, b_n$ where $\mathcal{D}$ is a term used to specify the probability distribution (continuous or discrete) and can be non-ground (i.e., it can be related to conditions in the body). Each ground instance of a distributional clause, call it $C_i\theta$, defines a random variable $h\theta$ with distribution $\mathcal{D}\theta$ if $(b_1, \ldots, b_n)\theta$ holds. As for Hybrid ProbLog, also in DC there is a set of predicates, call it rel_preds, used to compare the outcome of a random variable (indicated with $\simeq/1$) with constants or other random variables.

A DC program $\mathcal{P}$ is composed by a set of definite clauses $\mathcal{R}$ and a set of Distributional Clauses $\mathcal{C}$. The set $\mathcal{R} \cup \mathcal{F}$, where $\mathcal{F}$ is the set of true ground atoms for the predicates in rel_preds for each random variable in the program, defines a world. Furthermore, a valid DC program must satisfy several conditions related to the grounding of variables.

The semantics of DC programs can be described with a stochastic extension of the T_P operator [23], $ST_{\mathcal{P}}$. A function $\textsc{ReadTable}(\cdot)$ is also needed to evaluate probabilistic facts and to store sampled values for the random variables. If this function is applied to a probabilistic fact it returns the truth values of the fact according to the values of the random variables as arguments, by computing them or by looking into the table.

Given a valid DC program $\mathcal{P}$ and a set of ground facts I, the $ST_{\mathcal{P}}$ operator is defined as [16]:

$$
\begin{aligned}
ST_{\mathcal{P}}(I) = \{h \mid h \leftarrow b_1 \ldots, b_n \in ground(\mathcal{P}) \wedge \forall b_i : (&b_i \in I \vee \\
&(b_i = rel(t_1, t_2) \wedge \\
&(t_j = \simeq h \Rightarrow (h \sim \mathcal{D}) \in I) \wedge \textsc{ReadTable}(b_i) = true))\}.
\end{aligned}
$$

One limitation is that negation is not allowed in the body of a clause. The previous definition was further refined to [30]:

$$
\begin{aligned}
ST_{\mathcal{P}}(I) = \{h = v \mid\ & h \sim \mathcal{D} \leftarrow b_1, \ldots, b_n \in ground(\mathcal{P}) \wedge \forall b_i : \\
&(b_i \in I \vee b_i = rel(t_1, t_2) \wedge t_1 = v_1 \in I \wedge t_2 = v_2 \in I \wedge \\
&rel(v_1, v_2) \wedge v \text{ is sampled from } \mathcal{D})\} \cup \\
&\{h \mid h \leftarrow b_1, \ldots, b_n \in ground(\mathcal{P}) \wedge h \neq (r \sim \mathcal{D}) \wedge \forall b_i : \\
&(b_i \in I \vee b_i = rel(t_1, t_2) \wedge t_1 = v_1 \in I \wedge \\
&t_2 = v_2 \in I \wedge rel(v_1, v_2))\}
\end{aligned}
$$

where $rel \in \{=, <, \leqslant, >, \geqslant\}$. In word, for each DC clause, when the body is true in I, we sample a value v from the specified distribution for the random variable in the head and add $head = v$ to the interpretation. For deterministic clauses, when the body is true, new ground atoms are added to the interpretation. Computing the least fixpoint of the $ST_{\mathcal{P}}$ operator returns a model of an instance of the program. The $ST_{\mathcal{P}}$ operator is stochastic, so it defines a sampling process, and, consequently, a probability density over truth values of queries. However, DC programs do not admit negation in the body of rules.

Another proposal based on the DC semantics is HAL-ProbLog [45]. With this language, continuous random variables are represented with clauses of the form $D :: t \leftarrow l_1, \ldots, l_n$. Negative literals are also allowed in the body of clauses. For a grounding substitution θ, if $l_1\theta, \ldots, l_n\theta$ are true, $t\theta$ represents a continuous random variable that follows the distribution $D\theta$. Two built-in predicates allow the management of continuous random variables: $valS/2$, that unifies the random variable as the first argument with a logical variable in the second argument representing its value, and $conS/1$, that represents a constraint imposed on logical variables. Rules with identical head must have mutually exclusive bodies. This feature prevents the definition of a random variable following two different distributions, since only one of the distribution is allowed by the mutual exclusivity of the bodies. In detail, $valS(v, V)$ allows the logic variable V to unify with values for v, where v is a continuous variable that follows a certain distribution. Variable V then appears in predicate $conS/1$, also called *Iverson predicate*, where it is constrained by an algebraic condition. For example, $valS(v, V), conS(V > 10)$ constrain the value of the continuous random variable v to be greater than 10. The semantics of HAL-ProbLog extends those of DC but does not allow function symbols.

Extended PRISM [19] also allows the definition of continuous variables. The authors extended the PRISM language [40] to include continuous random variables with Gamma or Gaussian distributions, specified with the directive set_sw. So, for instance, $set\_sw(p, norm(Mean, Variance))$ states that the outcomes of the random process p follows a Gaussian distribution with the specified parameters. Moreover, it is possible to define linear equality constraints over the reals. The authors also propose an exact inference algorithm that symbolically reasons over the constraints on the random variables, exploiting the restrictions on the allowed continuous distributions and constraints.

The semantics of Extended PRISM is based on an extension of the distribution semantics for programs containing only discrete variables using the least model semantics of constraint logic programs [20]. In this way, the probability space is extended to a probability space of the entire program starting from the one defined for msw. The sample space of a single random variable is defined as $\mathbb{R}$ and it is extended to the product of the sample spaces for a set of random variables. For continuous random variables, the probability space for N random variables is defined as the Borel σ-algebra over $\mathbb{R}^N$ and the Lebesgue measure is used as probability measure. Also in this language, negations are not allowed in the body.

9.2. Inference

Inference for PCLP can be performed exactly or approximately. The main issue in exact inference for PCLP (and hybrid programs in general) is that it is impossible to enumerate all the explanations for a query since there is an uncountable number of them. Thus, exact inference algorithms for non-hybrid programs cannot be directly used. There are several possible solutions to perform inference in these domains.

Traditionally, inference methods for discrete probabilistic logic programs are based on *knowledge compilation* (KC) [11] and *weighted model counting* (WMC) [9]. With these two techniques, the logic program is transformed into a propositional knowledge base and then a weight is associated to each model according to the probabilities specified in the program. The KC steps usually transforms a PLP into a more compact representation such as ordered binary decision diagram (OBDD) or sentential decision diagram (SDD) [44]. Starting from this compact representation, the model counting is performed as follows: given a propositional logical theory Δ, a set of literals L and a weight function

$w : L \to \mathbb{R}^n$,

$$WMC(\Delta, w) = \sum_{M \models \Delta} \prod_{l \in M} w(l).$$

To handle a mixture of discrete and continuous random variables, WMC has been extended to *weighted model integration* (WMI) [7]. WMI allows to constrain the values of continuous variable by means of linear formulas. In the following definition we suppose that constraint are expressed as linear formulas over the reals, i.e., formulas of Satisfiability Modulo Theories of Linear Real Arithmetic (SMT($\mathcal{LRA}$)) [4].

Following the original definition of [7], given a SMT theory Δ over boolean variables $\mathcal{B}$, relational variables $\mathcal{X}$, literals $\mathcal{L}$ and a weight function w from literals to expressions over the set of relational variables, a weighted model integral can be defined. This formulation can theoretically be extended to other types of SMT theories, removing the linearity constraint. However, one of the problem of this approach is the presence of the integral, that usually can be solved exactly only if the integrand function is simple. Several solutions are currently available to solve WMI tasks: to speed up inference, in [8] the authors propose a technique called Component Caching. In [28] the authors proposed a formulation that can exploit predicate abstraction, a method commonly used in SMT. An approximate solution method, based on hashing can be found in [6]. In [22] the authors proposed an algorithm able to exploit factorizability in WMI by using an extended version of decision diagrams [21, 39]. One limitation of this solution is that weight functions must be piecewise polynomial. In [45, 46] the authors embed knowledge compilation and exact symbolic inference into WMI. However, WMI can only be applied to program without function symbols. For WMC, programs with function symbols are considered in [5]. For an extensive overview of WMI, see [29]. An alternative inference method, not based on WMI, is presented in [18, 19], where the authors overcome the enumeration problem by representing derivations in a symbolic way.

`cplint` hybrid programs can be queried using MCINTYRE [1, 2, 38]. The algorithm is based on Monte Carlo inference and program transformation. For example, a clause of the form

$$h(X, Y) : gaussian(Y, 0, 1)$$

is transformed into

$$h(X, Y) \leftarrow sample\_gauss(\_I, [X], 0, 1, Y)$$

where the predicate $sample\_gauss/5$ samples from a Gaussian distribution with, in this example, mean 0 and variance 1 and stores the result in Y. A sample from the program is taken by asking the query to the transformed program. MCINTYRE can be applied also to hybrid programs with function symbols. In fact, the infinite computations, those that are associated to an infinite composite choice, have probability 0 of being selected. Therefore sampling terminates.

Conditional approximate inference in MCINTYRE can be performed using *rejection sampling* or MCMC methods such as *Metropolis Hastings* or *Gibbs sampling* [2, 3]. MCMC methods are particularly useful when direct sampling from a joint distribution is not feasible, due to the complexity of the distribution itself. In Gibbs sampling, each variable is initialized with a random value. Then, for a fixed number of iterations (or until convergence), a sample for each random variable is taken given all the other variables. There are also other variants of Gibbs sampling, such as *blocked* Gibbs sampling, where two or more variables are grouped together, and the samples are computed from their joint distribution and not from each one individually.

Another MCMC algorithm is Metropolis Hastings: it queries the evidence and, if the evidence succeeds, it samples the query. If the query is successful, it is accepted with a probability depending on the number of samples taken in the previous and current sampling processes. The final probability is then computed as the number of successes over the number of samples.

MCMC algorithms have some limitations: usually the first few samples must be discarded, since they do not represent the real distribution, and they may require some time to converge. Moreover, for PCLP, evidence must be on ground atoms that do not contain continuous values as arguments, otherwise the probability of the evidence is 0. In case the evidence is on atoms with continuous values, the conditional probability of the evidence given the query can be defined in a different way [30] and other algorithms, such as *likelihood weighting* (LW), can be used. The basic idea behind LW is to assign a weight to each sample given the evidence and then compute the probability of the query as the sum of the weights of the samples where the query is true divided by the total sum of the weights of the samples. One issue of this algorithm is that weights of the samples may quickly go to 0. A possible solution is to use *particle filtering* [30] in which the individual samples are periodically resampled to reset their weights.

Approximate inference on hybrid programs with iterative interval splitting was proposed in [27]. The authors propose the Iterative Model Counting algorithm, which constructs a tree on the variable's domain. Each node of the tree, called Hybrid Probability Tree (HPT), is associated with a propositional formula and a range for each random variable. At each level, the range is split into two parts and each child node gets the previous propositional formula conditioned on the split made. The next node to expand, the variables and the relative partitions are selected by heuristics. Then, the probability of the event represented by the root of the tree is computed using a standard algorithm to compute a probability interval from a binary decision diagram.

10. Conclusions

In this paper, we have presented a new approach for defining the semantics of hybrid programs, i.e., programs with both discrete and continuous random variables. Our approach assigns a probability value to every query for programs containing negations and function symbols provided they are sound, i.e., each world must have a total well-founded model. Moreover, we have presented a syntax for representing hybrid programs in practice in the cplint[7] framework

[7] `http://cplint.eu`

that also includes algorithms for performing inference in these programs using Monte-Carlo. In the future we plan to develop exact inference algorithms for hybrid programs exploiting weighted model integration, also for programs with function symbols.

References

References

[1] M. Alberti, G. Cota, F. Riguzzi, and R. Zese. Probabilistic logical inference on the web. In G. Adorni, S. Cagnoni, M. Gori, and M. Maratea, editors, *AI*IA 2016: Advances in Artificial Intelligence, 21st Congress of the Italian Association for Artificial Intelligence, Pisa*, volume 10037 of *LNCS*, pages 351–363. Springer International Publishing, 2016. doi: 10.1007/978-3-319-49130-1_26.

[2] M. Alberti, E. Bellodi, G. Cota, F. Riguzzi, and R. Zese. `cplint` on SWISH: Probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11(1):47–64, 2017. doi: 10.3233/IA-170105.

[3] D. Azzolini, F. Riguzzi, E. Lamma, and F. Masotti. A comparison of MCMC sampling for probabilistic logic programming. In M. Alviano, G. Greco, and F. Scarcello, editors, *Proceedings of the 18th Conference of the Italian Association for Artificial Intelligence (AI*IA2019), Rende, Italy 19-22 November 2019*, volume 11946 of *Lecture Notes in Computer Science*, Heidelberg, Germany, 2019. Springer. doi: 10.1007/978-3-030-35166-3_2.

[4] C. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[5] V. Belle. Weighted model counting with function symbols. In G. Elidan, K. Kersting, and A. T. Ihler, editors, *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press, 2017. URL `http://auai.org/uai2017/proceedings/papers/132.pdf`.

[6] V. Belle, G. V. den Broeck, and A. Passerini. Hashing-based approximate probabilistic inference in hybrid domains. In M. Meila and T. Heskes, editors, *31st International Conference on Uncertainty in Artificial Intelligence (UAI 2015)*, pages 141–150. AUAI Press, 2015.

[7] V. Belle, A. Passerini, and G. V. den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In Q. Yang and M. Wooldridge, editors, *24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 2770–2776. AAAI Press, 2015.

[8] V. Belle, G. V. den Broeck, and A. Passerini. Component caching in hybrid domains with piecewise polynomial densities. In D. Schuurmans and M. P. Wellman, editors, *30th National Conference on Artificial Intelligence (AAAI 2015)*, pages 3369–3375. AAAI Press, 2016.

[9] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[10] Y. S. Chow and H. Teicher. *Probability Theory: Independence, Interchangeability, Martingales*. Springer Texts in Statistics. Springer, 2012.

[11] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002. doi: 10.1613/jair.989.

[12] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2 edition, 2002. doi: 10.1017/CBO9780511809088.

[13] L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015. doi: 10.1007/s10994-015-5494-z.

[14] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In M. M. Veloso, editor, *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, volume 7, pages 2462–2467. AAAI Press/IJCAI, 2007.

[15] B. Gutmann, M. Jaeger, and L. De Raedt. Extending problog with continuous distributions. In P. Frasconi and F. A. Lisi, editors, *20th International Conference on Inductive Logic Programming (ILP 2010)*, volume 6489 of *LNCS*, pages 76–91. Springer, 2011. doi: 10.1007/978-3-642-21295-6_12.

[16] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 11(4-5):663–680, 2011.

[17] P. Hitzler and A. Seda. *Mathematical Aspects of Logic Programming Semantics*. Chapman & Hall/CRC Studies in Informatics Series. CRC Press, 2016.

[18] M. A. Islam. *Inference and learning in probabilistic logic programs with continuous random variables*. PhD thesis, State University of New York at Stony Brook, 2012.

[19] M. A. Islam, C. Ramakrishnan, and I. Ramakrishnan. Inference in probabilistic logic programs with continuous random variables. *Theory and Practice of Logic Programming*, 12:505–523, 2012. ISSN 1475-3081. doi: 10.1017/S1471068412000154.

[20] J. Jaffar, M. J. Maher, K. Marriott, and P. J. Stuckey. The semantics of constraint logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998. doi: 10.1016/S0743-1066(98)10002-X.

[21] S. Kolb, M. Mladenov, S. Sanner, V. Belle, and K. Kersting. Efficient symbolic integration for probabilistic inference. In J. Lang, editor, *27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 5031–5037. AAAI Press/IJCAI, 2018.

[22] S. Kolb, P. Zuidberg Dos Martires, and L. De Raedt. How to exploit structure while solving weighted model integration problems. In A. Globerson and R. Silva, editors, *35th International Conference on Uncertainty in Artificial Intelligence (UAI 2019)*, page 262. AUAI Press, 2019.

[23] J. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987. ISBN 3-540-18199-7.

[24] S. Michels. *Hybrid Probabilistic Logics: Theoretical Aspects, Algorithms and Experiments*. PhD thesis, Radboud University Nijmegen, 2016.

[25] S. Michels, A. Hommersom, P. J. F. Lucas, M. Velikova, and P. W. M. Koopman. Inference for a new probabilistic constraint logic. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 2540–2546. AAAI Press/IJCAI, 2013.

[26] S. Michels, A. Hommersom, P. J. F. Lucas, and M. Velikova. A new probabilistic constraint logic programming language based on a generalised distribution semantics. *Artificial Intelligence*, 228:1–44, 2015. doi: 10.1016/j.artint.2015.06.008.

[27] S. Michels, A. Hommersom, and P. J. F. Lucas. Approximate probabilistic inference with bounded error for hybrid probabilistic logic programming. In S. Kambhampati, editor, *25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 3616–3622. AAAI Press/IJCAI, 2016.

[28] P. Morettin, A. Passerini, and R. Sebastiani. Efficient weighted model integration via SMT-based predicate abstraction. In C. Sierra, editor, *26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pages 720–728. IJCAI, 2017. doi: 10.24963/ijcai.2017/100.

[29] P. Morettin, A. Passerini, and R. Sebastiani. Advanced SMT techniques for weighted model integration. *Artificial Intelligence*, 275:1–27, 2019.

[30] D. Nitti, T. De Laet, and L. De Raedt. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 103(3):407–449, 2016. ISSN 1573-0565. doi: 10.1007/s10994-016-5558-8.

[31] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[32] D. Poole. Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. *New Generation Computing*, 11(3):377–400, 1993.

[33] D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94:7–56, 1997.

[34] D. Poole. Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44(1–3):5–35, 2000. doi: 10.1016/S0743-1066(99)00071-0.

[35] T. C. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-1989)*, pages 11–21. ACM Press, 1989.

[36] F. Riguzzi. The distribution semantics for normal programs with function symbols. *International Journal of Approximate Reasoning*, 77:1–19, 2016. doi: 10.1016/j.ijar.2016.05.005.

[37] F. Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning.* River Publishers, Gistrup, Denmark, 2018.

[38] F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, and G. Cota. Probabilistic logic programming on the web. *Software: Practice and Experience*, 46(10): 1381–1396, 10 2016. doi: 10.1002/spe.2386.

[39] S. Sanner, K. V. Delgado, and L. N. de Barros. Symbolic dynamic programming for discrete and continuous state mdps. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 643–652, Arlington, Virginia, USA, 2011. AUAI Press. ISBN 9780974903972.

[40] T. Sato. A statistical learning method for logic programs with distribution semantics. In L. Sterling, editor, *Logic Programming, Proceedings of the*

Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995, pages 715–729. MIT Press, 1995. doi: 10.7551/mitpress/4298.003.0069.

[41] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[42] J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In B. Demoen and V. Lifschitz, editors, *20th International Conference on Logic Programming (ICLP 2004)*, volume 3131 of *LNCS*, pages 431–445. Springer, 2004. doi: 10.1007/978-3-540-27775-0_30.

[43] J. Vennekens, M. Denecker, and M. Bruynooghe. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009. doi: 10.1017/S1471068409003767.

[44] J. Vlasselaer, J. Renkens, G. Van den Broeck, and L. De Raedt. Compiling probabilistic logic programs into sentential decision diagrams. In *1st International Workshop on Probabilistic Logic Programming (PLP 2014)*, pages 1–10, 2014.

[45] P. Zuidberg Dos Martires, A. Dries, and L. De Raedt. Knowledge compilation with continuous random variables and its application in hybrid probabilistic logic programming. *CoRR*, abs/1807.00614, 2018.

[46] P. Zuidberg Dos Martires, A. Dries, and L. De Raedt. Exact and approximate weighted model integration with probability density functions using knowledge compilation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, pages 7825–7833. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33017825.

Appendix A. Set Theory

A *one-to-one* function $f : A \to B$ is such that if $f(a) = f(b)$, then $a = b$, i.e., no element of B is the image of more than one element of A. A set A is *equipotent* with a set B if there exists a one-to-one function from A to B. A set A is *denumerable* if it is equipotent to the set of natural numbers $\mathbb{N}$. A set A is *countable* if there exists a one-to-one correspondence between the elements of A and the elements of some subset B of the set of natural numbers. Otherwise, A is termed *uncountable*. If A is countable and $B = \{1, 2, \ldots, n\}$, then A is called *finite* with n elements. $\varnothing$ (empty set) is considered a finite set with 0 elements. We define *powerset* of any set A, indicated with $\mathscr{P}(A)$, the set of all subsets including the empty set. For any reference space S and subset A of S, we denote with A^c the *complement* of A, i.e., $S \backslash A$, the set of all elements of S that do not belong to A.

An *order* on a set A is a binary relation $\leqslant$ that is reflexive, antisymmetric and transitive. If a set A has an order relation $\leqslant$, it is termed a *partially ordered set*, sometimes abbreviated with *ordered set*. A partial order $\leqslant$ on a set A is called a *total order* if $\forall a, b \in A,\ a \geqslant b$ or $b \geqslant a$. In this case, A is called *totally ordered*. The *upper bound* of a subset A of some ordered set B is an element $b \in B$ such that $\forall a \in A,\ a \leqslant b$. If $b \leqslant b'$ for all upper bounds b', then b is the *least upper bound* (lub). The definitions for *lower bound* and *greatest lower bound* (glb) are similar. If glb and lub exist, they are unique. A partially ordered set $(A, \leqslant)$ is a *complete lattice* if glb and lub exist for every subset S of A. A complete lattice A always has a *top element* $\top$ such that $\forall a \in A,\ a \leqslant \top$ and a *bottom element* $\perp$ such that $\forall a \in A,\ \perp \leqslant a$. A function $f : A \to B$ between two partially order set A and B is called *monotonic* if, $\forall a, b \in A,\ a \leqslant b$ implies that $f(a) \leqslant f(b)$. For an in-depth treatment of this topic see [12].

Appendix B. Ordinal Numbers, Mappings and Fixpoints

We denote the set of *ordinal numbers* with Ω. Ordinal numbers extend the definition of natural numbers. The elements of Ω are called *ordinals* and are represented with lower case Greek letters. Ω is *well-ordered*, i.e., is a totally ordered set and every subset of it has a smallest element. The smallest element of Ω is 0. Given two ordinals α and β, we say that α is a *predecessor* of β, or equivalently β is a *successor* of α, if $\alpha < \beta$. If α is the largest ordinal smaller than β, α is termed *immediate predecessor*. The *immediate successor* of α is the smallest ordinal larger than α, denoted as $\alpha + 1$. Every ordinal has an immediate successor called *successor ordinal*. Ordinals that have predecessors but no immediate predecessor are called *limit ordinals*. So, ordinal numbers can be limit ordinals or successor ordinals.

The first elements of Ω are the naturals $0, 1, 2, \ldots$ After all the natural numbers comes ω, the first *infinite ordinal*. Successors of ω are $\omega + 1$, $\omega + 2$ and so on. The generalization of the concept of sequence for ordinal number is the so-called *transfinite sequence*. The technique of induction for ordinal numbers is called *transfinite induction*: this states that, if a property $P(\alpha)$ is defined for all ordinals α, to prove that it is true for all ordinals we need to assume that $P(\beta)$ is true $\forall \beta < \alpha$ and then prove that $P(\alpha)$ is true. Transfinite induction proofs are usually structured in three steps: prove that $P(0)$ is true and prove $P(\alpha)$ for α both successor and limit ordinal.

Consider a lattice A. A *mapping* is a function $f : A \to A$. It is monotonic if $f(x) \leqslant f(y),\ \forall x, y \in A,\ x \leqslant y$. If $a \in A$ and $f(a) = a$, then a is a *fixpoint*. The *least fixpoint* is the smallest fixpoint. The *greatest fixpoint* can be defined analogously.

We define *increasing ordinal powers* of a monotonic mapping f as $f \uparrow 0 = \bot$, $f \uparrow (\alpha + 1) = f(f(\alpha))$ if α is a successor ordinal and $f \uparrow \alpha = \mathrm{lub}(\{f \uparrow \beta \mid \beta < \alpha\})$ if α is a limit ordinal. Similarly, *decreasing ordinal powers* are defined as $f \downarrow 0 = \top$, $f \downarrow \alpha = f(f(\alpha - 1))$ if α is a successor ordinal and $f \downarrow \alpha = \mathrm{glb}(\{f \downarrow \beta \mid \beta < \alpha\})$ if α is a limit ordinal. If A is a complete lattice and f a monotonic mapping, then the set of fixpoints of f in A is also a lattice (Knaster-Tarski theorem [17]). Moreover, f has a least fixpoint (lfp(A)) and a greatest fixpoint (gfp(A)). See [17] for a complete analysis of the topic.