The Structure and Complexity of Credal Semantics

Fabio G. Cozman and Denis D. Mauá

Universidade de São Paulo, Brazil

Abstract. A flexible semantics has been proposed by Lukasiewicz for probabilistic logic programs where we have a normal logic program augmented with a set of independent probabilistic facts. That semantics, which we call *credal semantics*, is the set of all probability measures (over stable models) that are consistent with a total choice of probabilistic facts. When each total choice produces a definite program, credal semantics is identical to Sato's distribution semantics. However, credal semantics is also defined for programs with cycles and negations. We show that the credal semantics always defines a set containing the probability measures that dominate an infinite monotone Choquet capacity (also known as a belief function). We also show how this result leads to inference algorithms and to an analysis of the complexity of inferences.

Keywords: Probabilistic logic programming, answer sets, stable model semantics, complexity theory.

1 Introduction

The term "probabilistic logic program" encompasses a significant number of distinct syntactic proposals [9,17,22,23]. One particularly successful proposal is jointly represented by Poole's probabilistic Horn abduction [25] and Sato's distribution semantics [28]. There the idea is that a logic program is enlarged with *probabilistic facts* that are associated with probabilities and that are usually assumed stochastically independent. For instance, we may have a rule:

$$\mathsf{up} :- \mathsf{actionUp}, \textbf{not}\ \mathsf{disturbanceUp}. \qquad \text{with } \mathbb{P}(\mathsf{disturbanceUp} = \mathsf{true}) = 0.1.$$

Depending on disturbanceUp, actionUp may succeed or not in leading to up.

Poole and Sato emphasized *acyclic* logic programs [25,26,28,29], even though Sato did handle cyclic ones. Since then, there has been work on cyclic probabilistic logic programs under variants of the distribution semantics [15,18,27,30]. In particular, a semantics for (acyclic and cyclic) probabilistic logic programs can be extracted from the work of Lukasiewicz on probabilistic description logics [18,19]. His proposal is that a probabilistic logic program defines a set of probability measures: for a fixed truth assignment over probabilistic facts, we take the set of all probabilities over stable models of the resulting normal logic program. This approach may seem a little complex, but it stays within two-valued logic and is directly related to answer set programming languages.

Lukasiewicz refers to his approach as the "answer set semantics" for probabilistic logic programs. However, as the approach is certainly distinct from the usual answer set semantics (which does not employ probabilities), and as the approach certainly applies in the presence of functions (usually not allowed in answer set programming), we adopt the name *credal semantics* to refer to it.

In this paper we study the structure and complexity of the credal semantics. We show that the semantics of a consistent probabilistic logic program is, surprisingly, the set of dominating measures of an infinitely monotone Choquet capacity. Such capacities are relatively simple objects that appear in several fields [21,31]. We obtain a straightforward derivation for inference algorithms, and we study the complexity of inferences, obtaining interesting novel results.

The paper is organized as follows. Basic terminology is reviewed in Section 2, and the credal semantics is presented in Section 3. The structure of the credal semantics is derived in Section 4, and its consequences concerning inference algorithms and complexity are discussed in Section 5.

2 A very short review: some syntax, some semantics

An *atom* is written as $\mathsf{r}(t_1, \ldots, t_k)$, where r is a *predicate* and each t_i is a *term*; that is, each t_i is a *constant* or a *logical variable*. A *normal logic program* is a finite set of *rules* such as $A_0 :- A_1, \ldots, A_m, \mathbf{not}\ A_{m+1}, \ldots, \mathbf{not}\ A_n.$, where each A_i is an atom. Atom A_0 is the *head* of the rule; the right hand side is the *body*. The body consists of a set of *subgoals* separated by commas (A_1 is a subgoal, $\mathbf{not}\ A_n$ is a subgoal). If there are no subgoals, the rule is called a *fact* and written simply as $A_0.$. Programs without **not** are *definite*. An atom is *ground* if it does not contain any logical variable; a program without logical variables is *propositional*. The *Herbrand base* of a program is the set of all ground atoms built from constants and predicates mentioned in the program. By grounding every rule with atoms in the Herbrand base, we obtain the (propositional) *grounding* of a program. The *grounded dependency graph* of a normal logic program is a directed graph where each grounded predicate is a node, and where there is an edge from node B to node C if there is a grounded rule where C appears in the head and B appears in the body; if B is preceded by **not** in the grounded rule, then the edge between B and C is *negative*. A program is *acyclic* when its grounded dependency graph is acyclic. An *interpretation* for a normal logic program $\mathbf{P}$ is a subset of the Herbrand base of $\mathbf{P}$; the idea is that atoms in the interpretation are true, and atoms not in the interpretation are false. A *model* is an interpretation such that every grounded rule is satisfied (that is, either the head is true or there is a subgoal that is false in the interpretation).

In this paper we do not allow functions to be used in programs, so as to stay within finite Herbrand bases. We leave the study of functions to future work.

One strategy to define the semantics of normal logic programs is to translate programs into some well-known logic, using a *completion*, and to take the models of the completion as the semantics. Another strategy is to select some "canonical" models as "the" semantics. Two canonical models have attracted most attention

due to their favorable properties: the *well-founded* [33] and the *stable model* semantics [14]. In this paper we are concerned with the latter (stable model) semantics. To define it, take normal logic program $\mathbf{P}$. For interpretation $\mathcal{I}$, define the *reduct* $\mathbf{P}^{\mathcal{I}}$ to be the definite program obtained by (i) grounding $\mathbf{P}$, (ii) removing all rules that contain a subgoal **not** A in the body such that A is in $\mathcal{I}$, (iii) removing all remaining subgoals of the form **not** A from the remaining rules. An interpretation $\mathcal{I}$ is a *stable model* if $\mathcal{I}$ is a *minimal model* for $\mathbf{P}^{\mathcal{I}}$ (minimal according to set inclusion). Note that a definite program always has a single minimal model, so the whole construction is well defined. An *answer set* is exactly a stable model.

3 Probabilistic logic programs

A probabilistic logic program is a pair $\langle \mathbf{P}, \mathbf{PF} \rangle$ where $\mathbf{P}$ is a normal logic program and $\mathbf{PF}$ is a set of *probabilistic facts* (we abbreviate "probabilistic logic program" by PLP). A probabilistic fact is an atom A that does not unify with any rule head (hence with any fact either), and that is associated with a probability assessment $\mathbb{P}(A) = \alpha$. We write a probabilistic fact as $\alpha :: A.$, where we have adopted the syntax of the ProbLog system[1] [13]. If A is a ground atom, then $\alpha :: A$ is a *ground* probabilistic fact. A probabilistic fact with logical variables is interpreted through its groundings (for instance, $\alpha :: \mathsf{r}(X_1, \ldots, X_m)$ is interpreted as the set of $\alpha :: \mathsf{r}(a_1, \ldots, a_k)$ for all groundings of r). A truth assignment for all grounded probabilistic facts in $\mathbf{PF}$ is a *total choice*. If θ is a total choice for $\mathbf{PF}$, we denote by $\mathbf{PF}^{\downarrow\theta}$ the set of atoms assigned true by θ.

We assume that all probabilistic facts are independent, so the probability of total choice $\theta = \{A_1 = t_1, \ldots, A_k = t_k\}$, where each t_i is true or false, is

$$\mathbb{P}(\theta) = \mathbb{P}(A_1 = t_1, \ldots, A_k = t_k) = \mathbb{P}(A_1 = t_1) \times \cdots \times \mathbb{P}(A_n = t_n)\,, \qquad (1)$$

where $\mathbb{P}(A_i = \mathsf{true}) = \alpha_i$ and $\mathbb{P}(A_i = \mathsf{false}) = 1 - \alpha_i$ given $\alpha_i :: A_i$.

Once a total choice is fixed, we can form a normal logic program consisting of $\mathbf{P} \cup \mathbf{PF}^{\downarrow\theta}$. Poole's and Sato's original work focused respectively on acyclic and definite programs, and for these programs the semantics of $\mathbf{P} \cup \mathbf{PF}^{\downarrow\theta}$ is rather uncontroversial, and is given by their *unique* stable model. Hence, every such probabilistic logic program induces a unique probability distribution over all atoms. For cyclic programs, Sato, Kameya and Zhou later adopted Fitting's three-valued semantics for each fixed total choice [30], while Hadjichristodoulou and Warren [15] and Riguzzi [27] have explored the well-founded semantics.

An alternative semantics can be extracted from the work on probabilistic description logic programs by Lukasiewicz [19], as noted in Section 1. To describe that proposal, a few definitions are needed. A PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ is *consistent* if there is at least one stable model for each fixed total choice of $\mathbf{PF}$. A *probability model* for a consistent PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ is a probability measure $\mathbb{P}$ over interpretations of $\mathbf{P}$, such that:

[1] Available at https://dtai.cs.kuleuven.be/problog/index.html.

(i) every interpretation $\mathcal{I}$ with $\mathbb{P}(\mathcal{I}) > 0$ is a stable model of $\mathbf{P} \cup \mathbf{PF}^{\downarrow\theta}$ for the total choice θ that agrees with $\mathcal{I}$ on the probabilistic facts; and
(ii) for every total choice $\theta = \{A_1 = t_1, \ldots, A_k = t_k\}$, where t_i is true or false, we have $\mathbb{P}(\theta) = \prod_{i=1}^{k} \mathbb{P}(A_i = t_i)$.
The set of all probability models for a PLP is the semantics of the program.

Lukasiewicz calls his proposed semantics the *answer set semantics* for probabilistic description logic programs. As we mentioned before, we prefer the term *credal semantics*, which we adopt. The reason for this latter name is that a set of probability measures is often called a *credal set* [2].

Now given a consistent PLP, we may be interested in the smallest possible value of $\mathbb{P}(A)$ for some ground atom A, with respect to the set $\mathbb{K}$ of all probability models of the PLP. This is conveyed by the *lower probability* of A, $\underline{\mathbb{P}}(A) = \inf_{\mathbb{P}\in\mathbb{K}} \mathbb{P}(A)$. Similarly, we have the *upper probability* of A, $\overline{\mathbb{P}}(A) = \sup_{\mathbb{P}\in\mathbb{K}} \mathbb{P}(A)$.

It is perhaps time for a few much needed examples. We start with two examples where each total choice produces a single stable model, and then move to two examples where some total choices lead to several stable models.

Example 1. First, consider an *acyclic* probabilistic logic program $\langle \mathbf{P}, \mathbf{PF} \rangle$. When $\mathbf{P}$ is acyclic, each total choice for $\mathbf{PF}$ yields an acyclic program (with a unique stable model that is also its well-founded semantics [1]), hence a single probability model is obtained (a singleton credal set).

For example, consider a fragment of the famous Bayesian network *Asia* [16] with three atoms, $\mathsf{smoking}$, cancer, and $\mathsf{bronchitis}$, with $\mathbb{P}(\mathsf{smoking}) = 1/2$, $\mathbb{P}(\mathsf{cancer}|\mathsf{smoking}) = 1/10$, $\mathbb{P}(\mathsf{cancer}|\neg\mathsf{smoking}) = 1/100$, $\mathbb{P}(\mathsf{bronchitis}|\mathsf{smoking}) = 3/5$, and $\mathbb{P}(\mathsf{bronchitis}|\neg\mathsf{smoking}) = 3/10$. These probabilities can be expressed through the following PLP:

$$0.5 :: \mathsf{smoking}. \quad 0.1 :: \mathsf{a1}. \quad 0.01 :: \mathsf{a2}. \quad 0.6 :: \mathsf{a3}. \quad 0.3 :: \mathsf{a4}.$$
$$\mathsf{cancer} :\!- \mathsf{smoking}, \mathsf{a1}. \qquad \mathsf{cancer} :\!- \mathbf{not}\ \mathsf{smoking}, \mathsf{a2}.$$
$$\mathsf{bronchitis} :\!- \mathsf{smoking}, \mathsf{a3}. \quad \mathsf{bronchitis} :\!- \mathbf{not}\ \mathsf{smoking}, \mathsf{a4}.$$

This PLP has a single probability model, yielding for instance $\underline{\mathbb{P}}(\mathsf{smoking}|\mathsf{cancer}) = \overline{\mathbb{P}}(\mathsf{smoking}|\mathsf{cancer}) = 1/11$. In fact, any Bayesian network over binary variables can be encoded using an acyclic probabilistic logic program [25,26]. □

Example 2. Sometimes a cyclic probabilistic logic program $\langle \mathbf{P}, \mathbf{PF} \rangle$ still has a single probability model. This happens for instance if $\mathbf{P}$ is *locally stratified*; that is, if its grounded dependency graph has no cycles containing a negative edge. If $\mathbf{P}$ is a locally stratified program, then any total choice of probabilistic facts produces a locally stratified normal logic program. And for locally stratified programs, most existing semantics, and particularly the well-founded and the stable model semantics, coincide and produce a single stable model.

For example, consider a PLP where p means "path" and e means "edge" (based on an example in the ProbLog distribution):

$$\begin{array}{c} \mathsf{p}(X,Y) :\!- \mathsf{e}(X,Y). \qquad \mathsf{p}(X,Y) :\!- \mathsf{e}(X,Y), \mathsf{p}(X,Y). \\ 0.6 :: \mathsf{e}(1,2). \quad 0.1 :: \mathsf{e}(1,3). \quad 0.4 :: \mathsf{e}(2,5). \quad 0.3 :: \mathsf{e}(2,6). \\ 0.3 :: \mathsf{e}(3,4). \quad 0.8 :: \mathsf{e}(4,5). \quad 0.2 :: \mathsf{e}(5,6). \end{array} \tag{2}$$

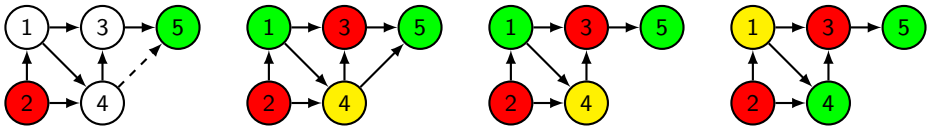

Fig. 1. Graph described in Example 4 (left), and the stable models.

That is, we have a random graph with nodes $1, \ldots, 6$, and probabilities attached to edges. The query $\mathbb{P}(\mathsf{p}(1,6) = \mathsf{true})$ yields the probability that there is a path between nodes 1 and 6. Using ProbLog one obtains $\mathbb{P}(\mathsf{p}(1,6) = \mathsf{true}) = 0.217$. □

Example 3. A common cyclic pattern in the literature is the pair of rules $\mathsf{a} :-$ **not** b. and $\mathsf{b} :-$ **not** a. [33]. Here is a more elaborated version, adapted from Ref. [12]. Take probabilistic fact $0.9 :: \mathsf{man}(\mathsf{dilbert})$. and rules:

$$\mathsf{single}(X) :- \mathsf{man}(X), \textbf{not}\ \mathsf{husband}(X). \quad \mathsf{husband}(X) :- \mathsf{man}(X), \textbf{not}\ \mathsf{single}(X).$$

When $\mathsf{man}(\mathsf{dilbert})$ is false, there is a single stable model $s_1 = \{\mathsf{man}(\mathsf{dilbert}) = \mathsf{false}, \mathsf{husband}(\mathsf{dilbert}) = \mathsf{false}, \mathsf{single}(\mathsf{dilbert}) = \mathsf{false}\}$. When $\mathsf{man}(\mathsf{dilbert})$ is true, there are two stable models, $s_2 = \{\mathsf{man}(\mathsf{dilbert}) = \mathsf{true}, \mathsf{husband}(\mathsf{dilbert}) = \mathsf{true}, \mathsf{single}(\mathsf{dilbert}) = \mathsf{false}\}$, and $s_3 = \{\mathsf{man}(\mathsf{dilbert}) = \mathsf{true}, \mathsf{husband}(\mathsf{dilbert}) = \mathsf{false}, \mathsf{single}(\mathsf{dilbert}) = \mathsf{true}\}$. Now, there is a probability model (that is, a probability measure over the stable models) such that $\mathbb{P}(s_1) = 0.1$ and $\mathbb{P}(s_2) = 0.9$ (hence $\mathbb{P}(s_3) = 0$), and a probability set model such that $\mathbb{P}(s_1) = 0.1$ and $\mathbb{P}(s_3) = 0.9$. In fact, any probability measure such that $\mathbb{P}(s_1) = 0.1, \mathbb{P}(s_2) = 0.9\gamma$, and $\mathbb{P}(s_3) = 0.9(1-\gamma)$, for $\gamma \in [0,1]$, is also a probability model. □

Example 4. Consider a graph coloring problem consisting of the rules:

$$\begin{gathered}\mathsf{coloredBy}(V, \mathsf{red}) :- \textbf{not}\ \mathsf{coloredBy}(V, \mathsf{yellow}), \textbf{not}\ \mathsf{coloredBy}(V, \mathsf{green}), \mathsf{vertex}(V).\\ \mathsf{coloredBy}(V, \mathsf{yellow}) :- \textbf{not}\ \mathsf{coloredBy}(V, \mathsf{red}), \textbf{not}\ \mathsf{coloredBy}(V, \mathsf{green}), \mathsf{vertex}(V).\\ \mathsf{coloredBy}(V, \mathsf{green}) :- \textbf{not}\ \mathsf{coloredBy}(V, \mathsf{red}), \textbf{not}\ \mathsf{coloredBy}(V, \mathsf{yellow}), \mathsf{vertex}(V).\\ \mathsf{noClash} :- \textbf{not}\ \mathsf{noClash}, \mathsf{edge}(V,U), \mathsf{coloredBy}(V,C), \mathsf{coloredBy}(U,C).\end{gathered}$$

and the facts: for $i \in \{1, \ldots, 5\}$, $\mathsf{vertex}(i)$., and

$$\begin{gathered}\mathsf{coloredBy}(2, \mathsf{red}). \quad \mathsf{coloredBy}(5, \mathsf{green}). \quad 0.5 :: \mathsf{edge}(4,5).\\ \mathsf{edge}(1,3). \quad \mathsf{edge}(1,4). \quad \mathsf{edge}(2,1). \quad \mathsf{edge}(2,4). \quad \mathsf{edge}(3,5). \quad \mathsf{edge}(4,3).\end{gathered}$$

The facts mentioning vertex and edge encode the graph in Figure 1 (left); the probabilistic fact is indicated as a dashed edge. For a fixed total choice, the stable models of the program are the 3-colorings of the graph. Thus, if $\mathsf{edge}(4,5)$ is true, there is a single stable model; otherwise, there are two stable models. We obtain $\underline{\mathbb{P}}(\mathsf{coloredBy}(1, \mathsf{yellow})) = 0$ and $\overline{\mathbb{P}}(\mathsf{coloredBy}(1, \mathsf{yellow})) = 1/2$, while $\underline{\mathbb{P}}(\mathsf{coloredBy}(4, \mathsf{yellow})) = 1/2$ and $\overline{\mathbb{P}}(\mathsf{coloredBy}(4, \mathsf{yellow})) = 1$, and finally $\underline{\mathbb{P}}(\mathsf{coloredBy}(3, \mathsf{yellow})) = \overline{\mathbb{P}}(\mathsf{coloredBy}(3, \mathsf{yellow})) = 1$. □

Example 5. Consider a game where one wins whenever the opponent has no moves [33], with rule $\mathsf{wins}(X) :- \mathsf{move}(X,Y), \mathbf{not}\ \mathsf{wins}(Y).$ and the following moves, one of which is probabilistic: $\mathsf{move}(\mathsf{a},\mathsf{b}).\quad \mathsf{move}(\mathsf{b},\mathsf{a}).\quad \mathsf{move}(\mathsf{b},\mathsf{c}).\quad 0.3 :: \mathsf{move}(\mathsf{c},\mathsf{d}).$. If $\mathsf{move}(\mathsf{c},\mathsf{d})$ is false, there is a single stable model (where $\mathsf{wins}(\mathsf{b})$ is the only winning position); otherwise, there are two stable models ($\mathsf{wins}(\mathsf{c})$ is true in both of them; $\mathsf{wins}(\mathsf{a})$ is true in one, while $\mathsf{wins}(\mathsf{b})$ is true in the other). Thus $\underline{\mathbb{P}}(\mathsf{wins}(\mathsf{a})) = 0.0$ and $\overline{\mathbb{P}}(\mathsf{wins}(\mathsf{a})) = 0.3$; $\underline{\mathbb{P}}(\mathsf{wins}(\mathsf{b})) = 0.7$ and $\overline{\mathbb{P}}(\mathsf{wins}(\mathsf{b})) = 1.0$; and $\underline{\mathbb{P}}(\mathsf{wins}(\mathsf{c})) = 0.3$ and $\overline{\mathbb{P}}(\mathsf{wins}(\mathsf{c})) = 0.3$. □

Finally, consider a program without credal semantics:

Example 6. If the barber shaves every villager who does not shave himself, does the barber shave himself? The program

$$\mathsf{shaves}(X,Y) :- \mathsf{barber}(X), \mathsf{villager}(Y), \mathbf{not}\ \mathsf{shaves}(X,Y).$$
$$0.5 :: \mathsf{barber}(\mathsf{bob}). \qquad 0.5 :: \mathsf{villager}(\mathsf{bob}).$$

does not have a stable model when both probabilistic facts are true (we obtain the pattern $\mathsf{p} :- \mathbf{not}\ \mathsf{p}$). Note that even in this case the resulting normal logic program has well-founded semantics (assigning $\mathsf{undefined}$ to $\mathsf{shaves}(\mathsf{bob},\mathsf{bob})$). □

4 The structure of credal semantics

Given the generality of PLPs, one might think that credal sets generated by the credal semantics can have an arbitrarily complex structure. Surprisingly, the structure of the credal semantics of a PLP is a relatively simple object:

Theorem 1. *Given a consistent probabilistic logic program, its credal semantics is a set of probability measures that dominate an infinitely monotone Choquet capacity.*

Before we present a proof of this theorem, let us pause and define a few terms. An infinitely monotone Choquet capacity is a set function $\underline{\mathbb{P}}$ from an algebra $\mathcal{A}$ on a set Ω to the real interval $[0,1]$ such that [2, Definition 4.2]: $\underline{\mathbb{P}}(\Omega) = 1 - \underline{\mathbb{P}}(\emptyset) = 1$ and, for any $A_1, \ldots, A_n$ in the algebra, $\underline{\mathbb{P}}(\cup_i A_i) \geq \sum_{J \subseteq \{1,\ldots,n\}} (-1)^{|J|+1} \underline{\mathbb{P}}(\cap_{j \in J} A_j)$. Infinitely monotone Choquet capacities appear in several formalisms; for instance, they are the *belief functions* of Dempster-Shafer theory [31], and summaries of *random sets* [21]. Given a set-function $\underline{\mathbb{P}}$, we can construct a set of measures that *dominate* $\underline{\mathbb{P}}$ as $\{\mathbb{P} : \forall A \in \mathcal{A} : \mathbb{P}(A) \geq \underline{\mathbb{P}}(A)\}$. We abuse language and say that this set itself is infinitely monotone. If a credal set $\mathbb{K}$ is infinitely monotone, then the *lower probability* $\underline{\mathbb{P}}$, defined as $\underline{\mathbb{P}}(A) = \inf_{\mathbb{P} \in \mathbb{K}} \mathbb{P}(A)$, is the generating infinitely monotone Choquet capacity. We also have the *upper probability* $\overline{\mathbb{P}}(A) = \sup_{\mathbb{P} \in \mathbb{K}} \mathbb{P}(A) = 1 - \underline{\mathbb{P}}(A^c)$.

Proof (Theorem 1). Consider a set Θ containing as states the posssible total choices of the probabilistic facts. Over this space we have a product measure that is completely specified by the probabilities attached to probabilistic facts. Now

consider a multi-valued mapping Γ between Θ and the space Ω of all possible models of our probabilistic logic program. For each element $\theta \in \Theta$, define $\Gamma(\theta)$ to be the set of stable models associated with the total choice θ of the probabilistic facts. Now we use the fact that a probability space and a multi-valued mapping induce an infinite monotone Choquet capacity over the range of the mapping (that is, over Ω) [21]. □

Infinitely monotone credal sets have several useful properties; for one thing they are closed and convex. Convexity here means that if $\mathbb{P}_1$ and $\mathbb{P}_2$ are in the credal set, then $\alpha\mathbb{P}_1 + (1-\alpha)\mathbb{P}_2$ is also in the credal set for $\alpha \in [0,1]$. Thus, as illustrated by Example 3:

Corollary 1. *Given a consistent probabilistic logic program, its credal semantics is a closed and convex set of probability measures.*

There are several additional results concerning the representation of infinitely monotone capacities using their Möbius transforms [2,31]; we refrain from mentioning every possible corollary we might produce by rehashing those results. Instead, we focus on a few important results that can be used to great effect in future applications. First, as we have a finite Herbrand base, we can use the symbols in the proof of Theorem 1 to write, for any event A [2, Section 5.3.2]:

$$\underline{\mathbb{P}}(A) = \sum_{\theta\in\Theta:\Gamma(\theta)\subseteq A} \mathbb{P}(\theta)\,, \quad \overline{\mathbb{P}}(A) = \sum_{\theta\in\Theta:\Gamma(\theta)\cap A\neq\emptyset} \mathbb{P}(\theta)\,. \tag{3}$$

This property directly leads to an inference algorithm described later. In fact, for infinitely monotone credal sets we can find easy expressions for lower and upper *conditional* probabilities (that is, the infimum and supremum of conditional probabilities): if A and B are events, then lower and upper probabilities of A given B are (where the superscript c denotes complement) [2]:

$$\underline{\mathbb{P}}(A|B) = \frac{\underline{\mathbb{P}}(A\cap B)}{\underline{\mathbb{P}}(A\cap B) + \overline{\mathbb{P}}(A^c\cap B)} \text{ and } \overline{\mathbb{P}}(A|B) = \frac{\overline{\mathbb{P}}(A\cap B)}{\overline{\mathbb{P}}(A\cap B) + \underline{\mathbb{P}}(A^c\cap B)}. \tag{4}$$

We also note that the computation of lower and upper *expected values*, and even *conditional expected values*, with respect to infinitely monotone Choquet capacities admits relatively simple expressions [34].

It is convenient to pause here and mention the constraint logic programming language of Michels et el. [20], a significant contribution that is based on credal sets (even though it uses a different syntax and semantics, for instance allowing continuous variables but not allowing multiple stable models per total choice). In that work expressions are (conditional) lower and upper probabilities are derived; those expressions directly mirror the ones presented in this section.

5 The complexity of inferences with credal semantics

In this section we focus on the computation of $\underline{\mathbb{P}}(\mathbf{Q})$ and $\overline{\mathbb{P}}(\mathbf{Q})$, where $\mathbf{Q}$ is a conjunction of assignments to some ground atoms in its Herbrand base. Recall that we *preclude* functions, so as to stay with *finite* Herbrand bases. Hence a direct translation of Expression (3) into an algorithm leads to:

- **Given** a PLP $\langle \mathbf{P}, \mathbf{PF} \rangle$ and $\mathbf{Q}$, initialize a and b with 0.
- For each total choice θ of probabilistic facts, compute the set S of all stable models of $\mathbf{P} \cup \mathbf{PF}^{\downarrow\theta}$, and:
  if $\mathbf{Q}$ is true in every stable model in S, then $a \leftarrow a + \mathbb{P}(C)$;
  if $\mathbf{Q}$ is true in some stable model of S, then $b \leftarrow b + \mathbb{P}(C)$.
- **Return** a and b (the value of a is $\underline{\mathbb{P}}(\mathbf{Q})$, the value of b is $\overline{\mathbb{P}}(\mathbf{Q})$).

Note that to find whether $\mathbf{Q}$ is true in every stable model of a program, we must run *cautious inference*, and to find whether $\mathbf{Q}$ is true in some stable model of a program, we must run *brave inference*; these logical procedures have been studied in depth in the literature [11].

In fact the algorithm above has already been derived by Cali et al. [4], using clever optimization techniques (Cali et al. actually obtain lower and upper conditional probabilities by combining Expressions (4) with (3)). The advantage of our approach is that the algorithm is a transparent consequence of known facts about capacities; other than that, Cali et al. have already presented the algorithm so we do not need to dwell on it. Rather, we wish to focus on the specific question of how complex is the computation of the lower probability $\underline{\mathbb{P}}(\mathbf{Q})$.

To be precise: as **input**, we have a PLP whose probabilities are rational numbers, and an assignment $\mathbf{Q}$ to some ground atoms in the (finite) Herbrand base; as **output**, we want (the rational number) $\underline{\mathbb{P}}(\mathbf{Q})$. We refer to this complexity as the *inferential complexity* of PLPs.

One may also be interested in the complexity of inferences when the PLP is fixed, and the only input is the query $\mathbf{Q}$. This is the *query complexity* of the program: **input** is $\mathbf{Q}$, **output** is $\underline{\mathbb{P}}(\mathbf{Q})$. This concept is based on analogous concepts found in database theory [5]: one may face situations where the program may be small compared to the query, and query complexity is the concept of practical interest.

So, we are ready to state our main results on complexity.

Theorem 2. *In this theorem, assume that input* PLP*s are consistent. Inferential complexity is* $\#\mathsf{NP}$*-equivalent for propositional* PLP*s, and* $\#\mathsf{P}$*-equivalent when restricted to stratified propositional* PLP*s. For* PLP*s where all predicates have a bound on arity, inferential complexity is* $\#\mathsf{NP}^{\mathsf{NP}}$*-equivalent, and* $\#\mathsf{NP}$*-equivalent when restricted to stratified programs. Query complexity is* $\#\mathsf{P}$*-hard; it is in* $\#\mathsf{NP}$ *(up to multiplication by a polynomial number), and is* $\#\mathsf{P}$*-equivalent when restricted to stratified programs.*

Before we prove this theorem, we must define a number of terms that appear in it. We deal with usual complexity classes such as NP, and with the concept of oracles [24]. The *polynomial hierarchy* is the class of languages $\bigcup_i \Delta_i^{\mathsf{P}} = \bigcup_i \Pi_i^{\mathsf{P}} = \bigcup_i \Sigma_i^{\mathsf{P}}$, where $\Delta_i^{\mathsf{P}} = \mathsf{P}^{\Sigma_{i-1}^{\mathsf{P}}}$, $\Pi_i^{\mathsf{P}} = \mathsf{co}\Sigma_i^{\mathsf{P}}$, $\Sigma_i^{\mathsf{P}} = \mathsf{NP}^{\Sigma_{i-1}^{\mathsf{P}}}$ and $\Sigma_0^{\mathsf{P}} = \mathsf{P}$. The complexity class $\#\mathsf{P}$ is the class of integer-valued functions computed by a counting Turing machine in polynomial time; a counting Turing machine is a standard non-deterministic Turing machine that prints in binary notation, on a separate tape, the number of accepting computations induced by the input [32]. Similarly

to the polynomial hierarchy, the counting hierarchy is defined by oracles. The class $\#\Sigma_k^{\mathsf{P}}$ contains the integer-valued functions computed by a counting Turing machine with access to a Σ_k^{P} oracle. The counting hierarchy is the union of all $\#\Sigma_k^{\mathsf{P}}$ (over all integers k). The problems with which we are concerned involve the computation of rational numbers and do not precisely fit into the class of $\#\mathsf{P}$ problems and its extensions, so we resort to *weighted reductions* [3], which are parsimonious reductions (Karp reductions that preserve the number of accepting paths) scaled by a positive rational number. The canonical complete problem for the class $\#\Sigma_k^{\mathsf{P}}$ via parsimonious reductions is $\#\Pi_k\text{SAT}$ [10], that gets, as **input**, a formula $\varphi(Y_1,\ldots,Y_n) = \forall\, X_1 \exists\, X_2 \ldots Q_k\, X_k \psi(X_1,\ldots,X_k,Y_1,\ldots,Y_n)$, where ψ is a CNF formula over variables $X_1,\ldots,X_k,Y_1,\ldots,Y_n$, and produces, as **output**, the number of assignments to the variables $Y_1,\ldots,Y_n$ that make the formula φ evaluate to true. For k odd, the problem remains complete even if the formula is specified in disjunctive normal form [10]. For a complexity class $\#\mathsf{C}$ of integer-valued functions computed by some (oracle) counting Turing machine, we say that a problem is $\#\mathsf{C}$-hard if any problem in $\#\mathsf{C}$ can be reduced to it by a weighted reduction. If a problem is $\#\mathsf{C}$-hard and can be solved with one call to a $\#\mathsf{C}$ oracle and a multiplication by a rational obtained with polynomial effort, then the problem is said to be $\#\mathsf{C}$-equivalent (as inspired by [8]).

Proof (Theorem 2). All results for stratified programs are available in a companion publication on the complexity of distribution semantics [6]. So the remainder of this proof focuses on the general (possibly cyclic) case.
For propositional programs: Membership follows as cautious logical reasoning is coNP-complete [11, Table 2]; inference obtains by counting the result of cautious inference, so we obtain membership in $\#\mathsf{NP}$. Hardness is shown by a reduction from $\#\Pi_1\mathsf{SAT}$: Count the number of assignments of $x_1,\ldots,x_n$ such that, for φ a CNF formula with clauses $c_1,\ldots,c_k$, $\forall y_1,\ldots,y_m \varphi(x_1,\ldots,x_n,y_1,\ldots,y_m)$. Write rules $\mathsf{t_i} :-$ **not** $\mathsf{f_i}$. and $\mathsf{f_i} :-$ **not** $\mathsf{t_i}$. for every variable y_i (thus there are 2^m stable models running through assignments of $y_1,\ldots,y_m$). For each x_i introduce $\mathsf{x_i}$ and probabilistic fact $0.5 :: \mathsf{x_i}$. For every clause c_j write $\mathsf{c_j} :- \ell.$ for each one of the literals in c_j, where ℓ is $\mathsf{x_i}$ or **not** $\mathsf{x_i}$ or $\mathsf{t_i}$ or $\mathsf{f_i}$ respectively if the literal is x_i or $\neg x_i$ or y_i or $\neg y_i$. Finally, write $\mathsf{sat} :- \mathsf{c_1},\ldots,\mathsf{c_k}$. The probability of a total choice of $\mathsf{x_1},\ldots,\mathsf{x_n}$ adds to $\underline{\mathbb{P}}(\mathsf{sat})$ iff every stable model satisfies all clauses. Hence the desired counting is $2^n\underline{\mathbb{P}}(\mathsf{sat})$.
For programs where predicates have bounded arity: Membership follows as cautious logical reasoning is Π_2^P-complete [11, Table 5]; inference obtains by counting the result of cautious inference, so we obtain membership in $\#\mathsf{NP}^{\mathsf{NP}}$. Hardness is shown by a reduction from $\#\Pi_2\mathsf{SAT}$: Count the number of $x_1,\ldots,x_n$ s.t. $\forall y_1,\ldots,y_m \exists z_1,\ldots,z_s \varphi(x_1,\ldots,x_m,y_1,\ldots,y_m,z_1,\ldots,z_s)$, where φ is a 3-CNF formula with clauses $c_1,\ldots,c_k$. The reduction is a combination of the previous reduction with the one used by Eiter et al. [11]. Introduce, as before, predicates $\mathsf{x_i}$, $\mathsf{t_i}$ and $\mathsf{f_i}$, and probabilistic facts $0.5 :: \mathsf{x_i}$ and rules $\mathsf{t_i} :-$ **not** $\mathsf{f_i}$. and $\mathsf{f_i} :-$ **not** $\mathsf{t_i}$.. Introduce predicates $\mathsf{c_j}$ but now the arity of $\mathsf{c_j}$ equals the number of variables z_i in c_j. Denote by Z_i a logical variable that corresponds to the propositional variable z_i, and by $\mathbf{Z}_j$ the logical variables corresponding to

propositional variables z_i appearing in clause c_j. For each clause c_j, go over the possible assignments $\mathbf{z_j}$ of the propositional variables associated with $\mathbf{Z}_j$. If this assignment makes the clause true, add the fact $\mathsf{c_j}(\mathbf{z}_j)$.. If instead the assignment leaves the clause dependent on some propositional variables x_i or y_i, then for every literal l_i (over a variable x_i or y_i) introduce the rule: $\mathsf{c_j}(\mathbf{z}_j) :- \ell.$, where ℓ is x_i or **not** x_i or $\mathsf{t_i}$ or $\mathsf{f_i}$ exactly as in the previous reduction. Finally, introduce $\mathsf{sat} :- \mathsf{c_1}(\mathbf{Z}_1), \ldots, \mathsf{c_k}(\mathbf{Z}_k)$.. As before, the desired count is obtained by $2^n\underline{\mathbb{P}}(\mathsf{sat})$. Concerning query complexity, membership follows from the fact that data complexity of logical cautious reasoning is coNP [7, Theorem 5.8]. □

We leave to future work a number of questions. First, we conjecture that the complexity of computing $\overline{\mathbb{P}}(\mathbf{Q})$ is the same as computing $\underline{\mathbb{P}}(\mathbf{Q})$ (as we only need to change from cautious to brave reasoning). Also, we did not present results on the complexity of probabilistic logic programs *without* a bound on arity (without such a bound, *logical* cautious reasoning is coNEXP-complete, so we conjecture that exponentially bounded counting Turing machines will be needed here). Finally, our complexity results were obtained assuming that PLPs were consistent: of course, in practice one must consider the problem of checking consistency. We just indicate a few facts about consistency checking:

Proposition 1. *Consistency checking is in Π_2^P for propositional* PLP*s and in Π_3^P for* PLP*s where predicates have a bound on arity.*

Proof. Consistency checking of a propositional PLP obtains by verifying whether logical consistency holds for each total choice of probabilistic facts, and this can be accomplished by deciding whether all total choices satisfy consistency; because logical consistency checking for this language is NP-complete [11, Table 1], we obtain membership to Π_2^P. An analogue reasoning leads to Π_3^P as logical consistency checking with bounded arity is Σ_2^P-complete [11, Table 4].

6 Discussion: moving to answer set programming

We have studied the credal semantics for probabilistic logic program, as proposed by Lukasiewicz [19]. This semantics is quite attractive and is not as well known as it deserves. We have shown that the credal semantics of a PLP is an infinite monotone credal set, and we have derived novel complexity results (using non-trivial classes in the counting hierarchy). In future work we plan to look at consistency checking, and also at the consequences of allowing functions. Moreover, we must include in the analysis a number of useful constructs adopted by answer set programming [12]. There, *classic negation*, such as $\neg\mathsf{wins}(X)$, is allowed on top of **not**. Also, constraints, such as $:- \phi$, are allowed to mean that ϕ is false. More substantial is the presence, in answer set programming, of *disjunctive heads*. With such a machinery, we can for instance rewrite the rules in Example 3 as a single rule $\mathsf{single}(X) \vee \mathsf{husband}(X) :- \mathsf{man}(X).$, and the rules in Example 4 as the pair:

$$\mathsf{coloredBy}(V, \mathsf{red}) \vee \mathsf{coloredBy}(V, \mathsf{yellow}) \vee \mathsf{coloredBy}(V, \mathsf{green}) :- \mathsf{vertex}(V).$$
$$:- \mathsf{edge}(V, U), \mathsf{coloredBy}(V, C), \mathsf{coloredBy}(U, C).$$

Now the point to be made is this. Suppose we have a probabilistic logic program $\langle \mathbf{P}, \mathbf{PF} \rangle$, where as before we have independent probabilistic facts, but where $\mathbf{P}$ is now a logic program with classic negation, constraints, disjuctive heads, and $\mathbf{P}$ is consistent in that it has stable models for every total choice of probabilistic facts. The proof of Theorem 1 can be reproduced in this setting, and hence *the credal semantics (the set of measures over stable models) of these probabilistic answer set programs is again an infinite monotone credal set.* The complexity of inference with these constructs is left for future investigation.

Acknowledgements

The first author is partially supported by CNPq, grant 308433/2014-9. The second author received financial support from the São Paulo Research Foundation (FAPESP), grant 2016/01055-1.

References

1. K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.
2. T. Augustin, F. P. A. Coolen, G. de Cooman, and M. C. M. Troffaes. *Introduction to Imprecise Probabilities*. Wiley, 2014.
3. A. Bulatov, M. Dyer, L. Ann Goldberg, M. Jalsenius, M. Jerrum, and D. Richerby. The complexity of weighted and unweighted #CSP. *J. of Computer and System Sciences*, 78:681–688, 2012.
4. A, Calì, T. Lukasiewicz, L. Predoiu, and H. Stuckenschmidt. Tightly coupled probabilistic description logic programs for the semantic web. In *J. on Data Semantics XII*, pages 95–130. Springer, 2009.
5. F. G. Cozman and D. D. Mauá. Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity. In *AAAI Conf. on Artificial Intelligence*, 2015.
6. F. G. Cozman and D. D. Mauá. Probabilistic graphical models specified by probabilistic logic programs: semantics and complexity. In *Int. Conf. on Probabilistic Graphical Models*, 2016.
7. E. Dantsin, T. Eiter, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
8. C. P. de Campos, G. Stamoulis, and D. Weyland. A structured view on weighted counting with relations to quantum computation and applications. Technical Report 133, Electronic Colloquium on Computational Complexity, 2013.
9. L. de Raedt, P. Frasconi, K. Kersting, and S. H. Muggleton. *Probabilistic Inductive Logic Programming*. Springer, 2010.
10. A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theoretical Computer Science*, 340(3):496–513, 2005.
11. T. Eiter, W. Faber, M. Fink, and S. Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, 5:123–165, 2007.
12. T. Eiter, G. Ianni, and T. Krennwalner. Answer set programming: a primer. In *Reasoning Web*, pages 40–110. Springer-Verlag, 2009.

13. D. Fierens, G. van den Broeck, J. Renkens, D. Shrerionov, B. Gutmann, Gerda Janssens, and Luc de Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2014.
14. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Int. Logic Programming Conf. and Symp.*, pages 1070–1080, 1988.
15. S. Hadjichristodoulou and D. S. Warren. Probabilistic logic programming with well-founded negation. In *Symp. on Multiple-Valued Logic*, pages 232–237, 2012.
16. S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society B*, 50(2):157–224, 1988.
17. T. Lukasiewicz. Probabilistic logic programming. In *European Conf. on Artificial Intelligence*, pages 388–392, 1998.
18. T. Lukasiewicz. Probabilistic description logic programs. In *European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pages 737–749, Barcelona, Spain, July 2005. Springer.
19. T. Lukasiewicz. Probabilistic description logic programs. *Int. J. of Approximate Reasoning*, 45(2):288–307, 2007.
20. S. Michels, A. Hommersom, P. J. F. Lucas, M. Velikova. A new probabilistic constraint logic programming language based on a generalised distribution semantics. *Artificial Intelligence Journal*, 228:1–44, 2015.
21. I. Molchanov. *Theory of Random Sets*. Springer, 2005.
22. R. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
23. L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1–2):147–177, 1997.
24. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
25. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.
26. D. Poole. The Independent Choice Logic and beyond. In *Probabilistic Inductive Logic Programming*, pages 222–243. Springer, 2008.
27. F. Riguzzi. The distribution semantics is well-defined for all normal programs. *Int. Workshop on Probabilistic Logic Programming*, pages 69–84, 2015.
28. T. Sato. A statistical learning method for logic programs with distribution semantics. In *Int. Conf. on Logic Programming*, pages 715–729, 1995.
29. T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. of Artificial Intelligence Research*, 15:391–454, 2001.
30. T. Sato, Y. Kameya, and N.-Fa Zhou. Generative modeling with failure in PRISM. In *Int. Joint Conf. on Artificial Intelligence*, pages 847–852, 2005.
31. G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.
32. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. of Computing*, 8(3):410–421, 1979.
33. A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Association for Computing Machinery*, 38(3):620–650, 1991.
34. L. Wasserman and J. B. Kadane. Computing bounds on expectations. *J. of the American Statistical Association*, 87(418):516–522, 1992.