# Specifying Credal Sets With Probabilistic Answer Set Programming

**Denis Deratani Mauá** DDM@IME.USP.BR
*Institute of Mathematics and Statistics, University of São Paulo, Brazil*

**Fabio Gagliardi Cozman** FGCOZMAN@USP.BR
*Escola Politécnica, University of São Paulo, Brazil*

## Abstract

Probabilistic Answer Set Programming offers an intuitive and powerful declarative language to represent uncertainty about combinatorial structures. Remarkably, under the credal semantics, such programs can specify any infinitely monotone Choquet Capacity in an intuitive way. Yet, one might be interested in specifying more general credal sets. We examine how probabilistic answer set programs can be extended to represent more general credal sets with constructs that allow for imprecise probability values. We characterize the credal sets that can be captured with various languages, and discuss the expressivity and complexity added by the use of imprecision in probabilistic constructs.

**Keywords:** probabilistic logics, answer set programming, credal networks.

## 1. Introduction

Probabilistic logic programming languages [24, 26, 27, 28, 32, 31] facilitate the specification of probabilistic scenarios by mixing probabilistic choices and logical rules. Probabilistic graphical models such as Bayesian networks can be easily specified with probabilistic logic programs [6]. In addition, such programs allow relational constructs and cyclic dependencies between variables, thus offering resources that go beyond (propositional) Bayesian networks [26]. For example, here is a program in Problog [14], a well known probabilistic programming framework, that encodes a random graph and the concept of reachability:

```
0.6::edge(1,2). 0.1::edge(1,3). 0.4::edge(1,4).
0.3::edge(2,3). 0.3::edge(2,4). 0.8::edge(3,4).
reachable(X,Y) :- edge(X,Y).
reachable(X,Y) :- edge(X,Z), reachable(Z,Y).
```

The first two lines state the marginal probabilities that each specific edge is present in the graph. The remaining lines specify if a node is `reachable` from another in a recursive (cyclic) manner.

Probabilistic *answer set* programs [3, 9, 10, 25] go one step further. They adopt expressive logic constructs such as disjunctions and negative cycles between variables [15], thus enabling one to express nondeterminism as well as probabilistic knowledge. For instance, the following program defines the concept of 3-colorability of a graph:

```
color(X,red);color(X,green);color(X,blue).
fail      :- edge(X,Y), color(X,C), color(Y,C).
colorable :- not fail.
```

The first line specifies that a node must have at least one of three `color`s (note the semicolon indicating logical disjunction). The remaining lines define a `fail`ure as an edge with both endpoints of the same color, and a `colorable` graph as one that does not contain `fail`ures.

Under the credal semantics [9], such programs specify an infinitely mononote Choquet capacity. For example, the program obtained by combining the two previous programs specify that $\Pr(\texttt{colorable}) \in [0.021, 0.998]$. These probabilities have an appealing interpretation: the lower endpoint is the probability that a random graph is generated such that no assignment of colors fail, whereas the upper probability is the probability of generating a 3-colorable random graph.

Given the ability of probabilistic answer set programming to represent Bayesian networks and infinitely monotone Choquet capacities, one might wonder whether such languages specify various structured credal sets, including those specified by credal networks [5]. That is the question we address in this work.

We start with a simple observation: probabilistic answer set programs under the credal semantics can specify *any* and *only* infinitely-monotone Choquet capacities, and thus have limited expressivity. To allow for more general credal sets that can represent meaningful credal networks and more, we revisit a previous proposal for interval-valued probabilistic facts [4]. For example, to specify that an earthquake occurs with probability between 0.001 and 0.1 we write:

```
[0.001,0.1]::earthquake.
```

We show that interval-valued probabilistic facts suffice to specify any credal network with finitely generated credal sets, even if we are restricted to nondisjunctive acyclic programs. Moreover, we show that the addition of (annotated) disjunctions, negative cycles and other expressive logical constructs do not add to expressivity, although they allow for more comfortable description of probabilistic phenomena such as coarsening.

While expressive, probabilistic answer set programs with interval-valued probabilistic facts specify credal sets only by enumeration of their extreme distributions. To obtain a more practical language, we propose *parameterized annotated disjunctions* that allow for the specification of credal sets as linear probability inequalities. For example, here is how we may encode that a team is more likely to win than to draw a match, and more likely to win than to lose:

```
P1::win;P2::draw;P3::lose :- P1 > P2, P1 > P3.
```

Credal sets defined by comparative probability judgments can have an exponential number of extreme points [20], hence rules as the above can lead to enormous savings in knowledge representation.

We also discuss the inclusion of variables (i.e., relational programs), inferential complexity, and the connection with credal networks and DTProblog [13]; the latter is an extension of Problog for representing decision-making situations.

## 2. Background

We review here two topics we rely on: imprecise probabilities and (probabilistic) answer set programming.

### 2.1. Imprecise Probability Models

A *Choquet capacity* is a nonnegative set-valued function $\mu(A)$ over some algebra $\mathcal{A}$ on $\Omega$ that satisfies $\mu(\emptyset) = 0$, $\mu(A) \leq \mu(B)$ for any $A \subseteq B$ in $\mathcal{A}$, and $\mu(\Omega) = 1$. We focus on finite spaces $\Omega$ in this paper, and we assume w.l.o.g. that $\mathcal{A}$ is the power set $2^\Omega$ of the sample space $\Omega$. The *conjugate* of $\mu$ is the $\mathcal{A}$-valued function $\overline{\mu}$ satisfying $\overline{\mu}(A) = 1 - \mu(A^c)$, where $A^c$ is the complement of $A$ w.r.t. $\Omega$. A capacity is $k$-monotone ($k \geq 2$) if for any collection of $n \leq k$ events, $B = \{A_1, \ldots, A_n\} \in \mathcal{A}$, we have that:

$$\mu(A_1 \cup \cdots \cup A_n) \geq \sum_{S \subseteq B} (-1)^{|S|+1} \mu(\cap_{A_i \in S} A_i). \qquad (1)$$

An *infinitely monotone* capacity is $k$-monotone for any $k$.

A probability distribution Pr *dominates* a capacity $\mu$ if $\mu(A) \leq \Pr(A)$ for all $A$. A *credal set* is a set of probability distributions on the same algebra. Given a credal set $\mathcal{M}$, the associated *lower probability* $\underline{\Pr}$ is the Choquet capacity defined as the lower envelope, $\underline{\Pr}(A) = \min_{\Pr \in \mathcal{M}} \Pr(A)$, and the *upper probability* is defined as its conjugate: $\overline{\Pr}(A) = 1 - \underline{\Pr}(A^c)$. Conversely, given a lower probability, the credal set formed by all dominating probability distributions is a closed and convex polyhedron [33].

A lower probability may satisfy the property of $k$-monotonicity (Expression (1)). In that case [34]:

$$\underline{\Pr}(A \mid B) = \frac{\underline{\Pr}(A \cap B)}{\underline{\Pr}(A \cap B) + \overline{\Pr}(A^c \cap B)}, \qquad (2)$$

provided $\overline{\Pr}(B) > 0$. Equation (2) reduces the computation of conditional lower probabilities to the computation of two unconditional lower/upper probabilities.

Let $\Lambda$ be some finite set called the initial space and consider a multi-valued mapping $\Gamma : \Lambda \to 2^\Omega$ to events of $\Omega$ such that $\Gamma(\lambda) \neq \emptyset$ for all $\lambda \in \Lambda$. Consider a probability measure Pr on $\Lambda$. The triple $\Lambda$, $\Gamma$ and Pr induce infinitely-monotone lower and upper probabilities by[1]

$$\underline{\Pr}(A) = \Pr(\{\lambda \in \Lambda : \Gamma(\lambda) \subseteq A\}), \qquad (3)$$

$$\overline{\Pr}(A) = \Pr(\{\lambda \in \Lambda : \Gamma(\lambda) \cap A \neq \emptyset\}). \qquad (4)$$

Augustin [2] and Miranda & de Cooman [21] showed that if we substitute Pr in the definition of Equation (3) above by some (other) $k$-monotone lower probability $\underline{\Pr}'$ on $\Lambda$, then the resulting lower probability $\underline{\Pr}$ on $\Omega$ is also $k$-monotone. Also, if $\underline{\Pr}'$ is coherent, then so is $\underline{\Pr}$ [21].

A common way to specify infinitely-monotone lower probabilities is by means of a normalized $m$-function, also called a basic belief assignment, which is a nonnegative function on $2^\Omega$ such that $m(\emptyset) = 0$ and $\sum_{A \subseteq \Omega} m(A) = 1$. The events $A$ for which $m(A) > 0$ are called *focal sets*. Given such a function, we define $\underline{\Pr}(A) = \sum_{B \subseteq A} m(B)$. The lower probability is then called a belief function [22, 29].

A *vacuous lower probability* is the one induced by $m(\Omega) = 1$ and $m(A) = 0$ for all other $A$. A *vacuous credal set* contains all probability distributions on some sample space. It is also the set of dominating distributions for a vacuous lower probability.

### 2.2. Answer Set Programming

*Answer Set Programming* (ASP) is a declarative programming language [15] that has its roots in logic programming and relational database theory. Its main use is in solving combinatorial search problems, usually by means of a guess-and-check strategy. Here we use the fragment of ASP that captures most of its expressivity and functionality.

**Syntax** An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where $p$ is a string starting with a lower case letter denoting the predicate name and $t_1, \ldots, t_n$ are each either a constant, which begins with a non-capital letter, or a variable, which begins with a capital letter. For example, `route(From, myCity,20)` is an atom with predicate name `route`, variable `From` and constants `myCity` and `20`. A *disjunctive normal rule* is an expression of the form

$$A_1; \ldots; A_m \text{ :- } B_1, \ldots, B_n$$

where $A_1, \ldots, A_m$ are atoms and each $B_i$ is a *literal*, that is, either an atom or an atom preceded by not (which indicates default negation). The atoms $A_i$ are collectively called the

[1] The induced capacity is coherent in Walley's sense [33], so it is a lower probability in our definition here.

*head* of the rule, and $B_1, \ldots, B_n$ are collectively called the *body* of the rule. We denote the set of atoms in the head as of a rule $r$ as head($r$) and the set of atoms in the body as body($r$). We allow the body to be empty (i.e., $n = 0$), but require the head to be non-empty (i.e., $m \geq 1$). A rule with a single atom in the head is called a *normal rule*. A normal rule with an empty body is a *fact*. An *answer set program* is a finite set of disjunctive normal rules.

**Semantics** A *grounding* of an atom is a substitution of variables by constants. The grounding of a rule applies the same grounding to all its atoms. The *Herbrand base* is the set with all possible groundings of atoms in the program, using the constants that appear in the program. A program is *propositional* if it does not contain any variables. The semantics of a program with variables is the semantics of its grounding, with grounded atoms from the Herbrand base. Hence, for the rest of this section, we assume programs are propositional. An *interpretation* is a two-valued mapping from atoms in the Herbrand base to true or false assignments, represented as a set of true atoms (this allows us to define a partial ordering over interpretations by subset inclusion). An interpretation $\omega$ satisfies a ground atom $A$ if $A \in \omega$, written as $\omega \models A$. The interpretation satisfies a negated literal not $A$ if $A \notin \omega$, written $\omega \models \neg A$. Finally, $\omega$ satisfies a disjunctive normal rule if $\omega \cap \{A_1, \ldots, A_m\} \neq \emptyset$ or $\omega \not\models B_i$, for some $i = 1, \ldots, n$ in the body. That is, the rule is satisfied if the body is false or (the body is true and) some atom in the head is satisfied. A *model* is an interpretation that satisfies all rules of the program. A model is *minimal* if there is no other model that is strictly contained in it. Given program $P$ and interpretation $I$, their reduct $P^I$ is the program obtained by removing all rules with not $A$ in their body and $A \in I$, and then by removing each not $A$ from all remaining rules. A *stable model* of $P$ is a minimal model of the reduct $P^I$ [16].

The dependency structure of a logic program is a directed graph containing a node for each (ground) atom and an edge from $B$ to $A$ if there is a rule with $A$ in the head and $B$ in the body. Moreover, we label edges as negative if $B$ appears negated (i.e., preceded by not) in some rule, otherwise the edge is label positive. A program is *acyclic* iff its dependency graph is acyclic and rules are normal. A normal program is *stratified* iff there are no negative cycles in the graph. Figure 1 shows an example program and its dependency graph; solid (resp., dashed) edges represent positive (resp., negative) labels. The program is not acyclic nor stratified, due the negative cycle between $c$ and $d$. Removing the second line of the program makes the program acyclic (hence stratified). Keeping those rules but removing the not operations makes the program stratified (but not acyclic).

```
a.
c :- not d.     d :- not c.
c :- a.         d :- a.
b :- a.
b :- not a, c.
```

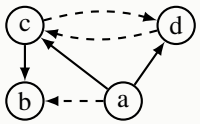


Figure 1: An example program and its dependency graph.

### 2.3. Probabilistic Logic Programming

*Probabilistic Answer Set Programming* (PASP) extends ASP with probabilistic constructs that allow the truth-value of some atoms to be selected according to some distribution. This in effect makes PASP a declarative probabilistic programming language; in particular, PASP provides an intuitive and powerful language for describing probability functions over rich combinatorial objects such as graphs, routes, ordering, etc [12, 17, 23, 24, 27].

We follow Sato's suggestions on semantics [28], hence a *probabilistic logic program* is a set of independent random choices that when realized produce a logic program. We adopt a syntax that is based on *annotated disjunctive rules*, which are constructs of the form [31]

$$\alpha_1 :: A_1 ; \ldots ; \alpha_m :: A_m \ :- \ B_1, \ldots, B_n$$

where $A_1, \ldots, A_m$ are different standard atoms, $\alpha_i$ are non-negative real values such that $\sum_i \alpha_i = 1$, and $B_1, \ldots, B_n$ are literals. Intuitively, an annotated disjunctive rule specifies a categorical random variable with $m$ values that are distributed according to $\alpha_1, \ldots, \alpha_m$, respectively. We allow the body to be empty ($n = 0$), but require the head to be non-empty ($m \geq 1$). An annotated disjunctive rule with an empty body is an *annotated disjunction*. A *probabilistic answer set program* (or PASP program, for short) is a finite set of disjunctive normal rules and annotated disjunctive rules.

The (credal) semantics of a (propositional) PASP program $\theta$ is built from three ingredients. The first ingredient is a translation of $\theta$ into different ASP programs, each consisting of different transformations of the annotated disjunctive rules, as follows. Define an *atomic choice* as a selection of exactly one atom $A_i$ from the head of an annotated disjunctive rule. A *composite choice* $\lambda$ maps each annotated disjunctive rule in the program into an atomic choice. For every composite choice $\lambda$, generate an ASP program $\theta(\lambda)$ by replacing each annotated disjunctive rule $r$ in $\theta$ with the normal rule head($r$) $\cap$ $\lambda(r) \leftarrow$ body($r$). For example, consider the stratified PASP program

```
0.3::a; 0.7::c :- d.
0.6::b; 0.4::d.
a :- b.
```

The composite choices (written as strings, for clarity) are `ab`, `ad`, `cb` and `cd`. The first composite choice produces the ASP program

```
a :- d.    b.    a :- b.
```

That program has a single stable model given by {a, b}.

The second ingredient is a probability distribution over the induced ASP programs. Let $\Lambda$ denote the set of all composite choices of a program. We define a probability distribution Pr on $\lambda \in \Lambda$ as the product of the probabilities annotating each atomic choice in $\lambda$. This distribution is extended to a distribution over ASP programs by $\Pr(\lambda) = \Pr(\theta(\lambda))$. In the previous example, the probability of ab and the induced ASP program is $0.3 \cdot 0.6 = 0.18$.

The final ingredient extends the translation into a multi-valued mapping, and the probability distribution into a lower probability function. Let $\Omega$ denote the set of all interpretations of the Herbrand base of $\theta$. Define $\Gamma$ as the mapping from composite choices $\lambda$ to the respective set of stable models of $\theta(\lambda)$. We assume that $\theta(\lambda) \neq \emptyset$ for each $\lambda$, otherwise $\theta$ is said to be *inconsistent* and has no semantics. The lower and upper probability functions are thus defined by Equation (3) and Equation (4), respectively, and they have an interesting interpretation. The lower probability $\underline{\Pr}(A)$ is the probability that $A$ will be satisfied by *all* stable models of an induced logic program $\theta(\lambda)$ (a logical inference known as *cautious* reasoning). The upper probability $\overline{\Pr}(A)$ is the probability that $A$ will be satisfied by *some* stable model of an induced logic program (a.k.a. *brave* reasoning). Note that if $\omega$ is not a stable model of any induced program then $\underline{\Pr}(\omega) = \overline{\Pr}(\omega) = 0$. For stratified programs, the multi-valued mapping $\Gamma(\lambda)$ always maps to singletons, hence $\underline{\Pr}(A) = \overline{\Pr}(A) = \Pr(A)$, and we recover the standard semantics of probabilistic logic programs [14].

To observe the subtlety of the semantics of PASP programs, consider the program

```
0.3::a;0.7::c.    0.6::b;0.4::d.    a :- b.
```

The respective stable models are $\Gamma(\mathtt{ab}) = \{\mathtt{ab}\}$, $\Gamma(\mathtt{ad}) = \{\mathtt{ad}\}$, $\Gamma(\mathtt{cb}) = \{\mathtt{abc}\}$ and $\Gamma(\mathtt{cd}) = \{\mathtt{cd}\}$. Notice that a is false only for the stable model cd, generated by the composite choice cd. The lower probability of a is thus $\underline{\Pr}(\mathtt{a}) = 1 - \Pr(\mathtt{cd}) = 1 - (0.7 \cdot 0.4) = 0.72 \neq 0.3$.

In [9], the authors suggested adopting a slightly different semantics to enforce that the marginal probabilities derived from the semantics agree with the annotated probability values (e.g., to ensure that $\Pr(\mathtt{a}) = 0.3$ in the example). The same semantics was suggested earlier by Dantsin [10], which considered only logic programs with single (stable) models. While appealing, that proposal might leave programs such as the above with no probability models. It is also equivalent to explicitly including rules in the program that prevent alternative composite choices (e.g., a and c) to be true simultaneously. We do not pursue this strategy in this paper.

We conclude this section with a PASP encoding of the famous Monty Hall's problem, to illustrate how PASP offers an appealing language to specify factored infinitely monotone lower probabilities by means of nondeterminism and probabilistic choices (lines starting with % are comments):

```
% A prize is randomly placed behind 1 of 3 doors
1/3::prize(1); 1/3::prize(2); 1/3::prize(3).
% Each door hides either a prize or a goat
goat(X) :- not prize(X).
% The participant selects door 1
select(1).
% The host will open some door
open(1); open(2); open(3).
% which does not hide the prize
false :- prize(X), open(X), not false.
% and has not been selected by the participant
false :- open(X), select(X), not false.
% The host offers a choice to change
choice(1); choice(2); choice(3).
% to some other door that is not open
false :- choice(X), open(X), not false.
% and has not been selected
false :- choice(X), select(X), not false.
% The participant wins if the initial door
% selected has the prize and she does not take
% the offer to change doors
win_keep :- select(X), prize(X).
% or if she changes to the door with the prize
win_change :- choice(X), prize(X).
```

One interesting modeling strategy in the program above are the rules of the form

```
false :- B1,...,Bn,not false.
```

Such rules create a contradiction whenever $B_1, \ldots, B_n$ are all true, and it thus excludes any interpretation where the $B_i$s all hold. Such constructs are called *integrity constraints* and are quite common in ASP modeling as they rule out infeasible candidate solutions (so much that we usually write them as head-free clauses, viz. :- B1,...,Bn).

Back to the Monty Hall example, the reader can check that $\underline{\Pr}(\mathtt{win\_change}) = \overline{\Pr}(\mathtt{win\_change}) = 2/3$. Thus, the best strategy for the participant is to always change doors. For the particular scenario where the participant *observes* that the host opens door 2, we have that $\underline{\Pr}(\mathtt{win\_change}|\mathtt{open(2)}) = 1/2$ and $\overline{\Pr}(\mathtt{win\_change}|\mathtt{open(2)}) = 1$. The non singleton intervals reflect the incompleteness of our knowledge about the preference of the host in selecting a door to open, which in the program is modeled as a logic disjunction. Despite that, the best course of action is still to change doors.

## 3. Specifying Infinitely Monotone Credal Sets

As can be inferred from the definition of semantics of a PASP program, the induced lower probability function is infinitely monotone. And since the marginalization (or projection) of such functions to a subset of variables remains infinitely monotone, we see that PASP programs can only specify infinitely monotone lower probabilities. We now show that the converse also holds.

To start, consider a simple credal set $\Pr(X = 1) \in [\alpha, \beta]$ for a Bernoulli random variable. It is possible to generate this

credal set using cycles and probabilistic facts, as explored in previous literature [9]; however, a very direct encoding can be adopted by exploiting (annotated) disjunctions and the set's $m$-function characterization:

```
α::m(1); 1−β::m(2); β−α::m(3).
x(1) :- m(1).          x(0) :- m(2).
x(1); x(0) :- m(3).
```

The atoms `x(1)` and `x(0)` represent the assignments $X = 1$ and $X = 0$, respectively. The PASP program induces three logic programs, each containing one of `m(1)`, `m(2)` or `m(3)` with probabilities $\alpha$, $1-\beta$ and $\beta-\alpha$, respectively. The first program contains a single stable model satisfying `x(1)` but not `x(0)`. The second program contains a single stable model satisfying `x(0)` and not `x(1)`. Finally, the third program contains two stable models, each one satisfying either `x(0)` or `x(1)` (but not the other). Thus, by collecting the respective probabilities, we see that $\underline{\Pr}(\texttt{x(1)}) = \alpha$ and $\overline{\Pr}(\texttt{x(1)}) = \beta$, as intended.

The strategy used in this simple example is rather general in the sense that it can be used to specify any infinitely monotone lower probability (and its dominating credal set) by means of an inducing $m$-function as follows. Take a total ordering $A_1, \ldots, A_n$ of focal sets of $m$, and add the annotated disjunction:

```
m(A1)::m(1); ...; m(An)::m(n).
```

Take also an ordering $\omega_1, \omega_2, \ldots$ of the elements in $\Omega$, and for each focal set $A_i = \{\omega_{i_1}, \ldots, \omega_{i_k}\}$, add the rule

```
x(i1); ...; x(ik) :- m(i).
```

The atoms `x(i)` represent the elements $\omega_i \in \Omega$. This straightforward scheme shows that:

**Theorem 1** *Every infinitely monotone lower probability can be specified by a probabilistic answer set program in size proportional to the number of focal sets of its m-function characterization.*

**Proof** Any such lower probability can be characterized uniquely by its corresponding $m$-function. Thus write the corresponding program, as described. It is clear that the program size is proportional to the size of the $m$-function characterization. To show that the program encodes the intended lower probability, observe that the composite choices $\lambda : r \mapsto \texttt{m(i)}$ are in one-to-one correspondence with the focal sets $A_i$, and for each such $\lambda$ the stable models are $\Gamma(\lambda) = \{\{\texttt{m(i)}, \texttt{x(k)}\} : \omega_k \in A_i\}$. Hence, the credal semantics assigns $\underline{\Pr}(A) = \sum_{\lambda:\Gamma(\lambda)\subseteq A} \Pr(\lambda) = \sum_{A_i \subseteq A'} m(A_i)$ to any event $A$, where $A'$ in the last sum maps the interpretations in $A$ to elements of $\Omega$. ■

## 4. Specifying General Credal Sets

As we have noted in the previous section, PASP programs can only capture infinitely-monotone lower probabilities (and their corresponding dominating credal sets). Following an approach initiated by Augustin [2], we extend the expressiveness of the PASP programs by allowing the probability mass functions over composite choices (i.e., the basic belief assignments, in belief theory's jargon) to vary inside a credal set. This idea has already been put forward by Bueno et al. [4] in the context of Markov Decision Processes. In that work, however, the credal sets characterized by such constructs were not discussed, and imprecision was limited to probabilistic facts, hence constrained to intervals. In this section, we provide a more in-depth discussion of credal sets constructed with imprecisely specified PASP programs, as well as more general specification languages.

### 4.1. Probability Intervals

We start with the simplest case of interval-valued annotated disjunctive rules:

```
[ℓ1, u1]::A1; ...; [ℓ1, u1]::Am :- B1, ..., Bn .
```

We assume the intervals of any such construct are *reachable*, that is, $\ell_i + \sum_{j\neq i} u_j \geq 1$ and $u_i + \sum_{j\neq i} \ell_j \leq 1$ for $i = 1, \ldots, m$.

The semantics of programs with such rules is the set of all PASP programs where the probability of annotated disjunctions are selected according to the respective intervals. That is, given a program $\theta$ we generate a (possibly infinite) set of PASP programs by replacing each interval-valued rule $r$ in $\theta$ by an annotated disjunctive rule with weights $\alpha_1, \ldots, \alpha_m$ such that $\alpha_i \in [\ell_i, u_i]$, $i = 1, \ldots, m$, and $\sum_i \alpha_i = 1$. Each such program then defines a probability distribution $\Pr(\lambda)$ over composite choices as the product of associated probabilities, as before. Let $\mathcal{M}(\Lambda)$ denote such set of probabilities. We define a lower probability function $\underline{\Pr}'$ over sets of composite choices $\Lambda' \subseteq \Lambda$ as the lower envelope of those probability distributions:

$$\underline{\Pr}'(\Lambda') = \min \{\Pr(\Lambda') \in \mathcal{M}(\Lambda)\} . \tag{5}$$

Because we assumed that the intervals of annotated disjunctive rules are reachable, the set in the equation above is always non-empty and hence the lower probability is well-defined. Finally, the *extended credal semantics* of a program with interval-valued annotated disjunctive rules is the extension of that lower probability function in Equation (5) by the multi-valued mapping $\Gamma$:

$$\underline{\Pr}(A) = \underline{\Pr}'(\{\lambda \in \Lambda : \Gamma(\lambda) \subseteq A\}) . \tag{6}$$

Thus, we now have two sources of incompleteness, one arising from the imprecision in the specification of the composite probability distribution $\Pr(\lambda)$, and other from

the multi-valued mapping to stable models $\Gamma(\lambda)$. Take the following simple illustrative program.

```
[0.3,0.7]::a; [0.7,0.3]::c.
0.6::b; 0.4::d.
a :- b.
```

The extended credal semantics of the program is the set of PASP programs obtained by varying the probability values for the annotated disjunction in the first line within the given (reachable) intervals. Each such PASP program is acyclic and hence each composite choice induces a single answer set. Hence, we have probability $\theta + 0.6 - \theta \cdot 0.6$ of `a` being true, where $\theta \in [0.3, 0.7]$. For instance, $\underline{\Pr}(\texttt{a}) = 0.72$ and $\overline{\Pr}(\texttt{a}) = 0.88$. Note that a non-sharp probability value is generated in spite of the deterministic semantics (unique stable models) of the induced logic programs.

The lower probability function in Equation (6) is no longer necessarily an infinitely monotone Choquet capacity. In fact, we have that:

**Theorem 2** *Every finitely-generated credal set can be represented by an acyclic and positive probabilistic logic program with a single vacuous interval-valued annotated disjunction and a set of precise annotated disjunction.*

**Proof** Let $\Omega = \{\omega_1, \ldots, \omega_n\}$ be the sample space, and $p_1, \ldots, p_m$ denote the vertices characterizing the credal set. Write the program:

$$[0,1] :: v_1 ; \ldots ; [0,1] :: v_m .$$
$$p_i(\omega_1) :: w_1 ; \ldots ; p_i(\omega_m) :: w_m \ :- \ v_i . \quad [\forall i = 1, \ldots, m]$$

Intuitively, the first rule defines the set of convex combination of the extreme distributions of the credal set. Each remaining rule selects the respective extreme distribution once the corresponding vertex has been defined. The extreme distributions are obtained by placing all probability mass to a single $v_i$ atom in the annotated disjunction, thus effectively "selecting" a vertex of the probability mass function polytope. We obtain the desired credal set by marginalizing out $v_i$.

To obtain the same result using only annotated disjunctions, transform each annotated disjunctive rule into one annotated disjunction $p_i(\omega_1) :: c_1, \ldots, p_i(\omega_n) :: c_n$, and $n$ normal rules of the form $w_j :- v_i, c_j$, for $j = 1, \ldots, n$. ■

It is important to notice that under the extended credal semantics, vacuous annotated disjunctions and (standard) disjunctions have very different effects. For example, the following program specifies a credal set $\Pr(\texttt{x(1)}) \in [\alpha, \beta]$:

$$[0,1] :: \texttt{v(1)} ; [0,1] :: \texttt{v(2)} .$$
$$\alpha :: \texttt{x(1)} ; (1-\alpha) :: \texttt{x(0)} \ :- \ \texttt{v(1)} ,$$
$$\beta :: \texttt{x(1)} ; (1-\beta) :: \texttt{x(0)} \ :- \ \texttt{v(2)} .$$

Replacing the vacuous annotated disjunction with

```
v(1);v(2).
```

produces $\underline{\Pr}(\texttt{x(1)}) = \alpha\beta$.

In closing this section, we note that an interval-valued annotated disjunction induces lower probabilities that are actually two-monotone capacities [11] over composite choices. Given that a multi-valued mapping whose domain is endowed with a two-monotone capacity also generates a two-monotone capacity in its range [2, 21], we have that a program with a single interval-valued annotated disjunction specifies a two-monotone lower probabilities over atoms.

### 4.2. Probabilistic Facts

Probabilistic logic programs are most often defined with *probabilistic facts* in lieu of annotated disjunctions. A probabilistic fact is an expression of the form $\alpha :: A$, where $A$ is a standard atom [14]. It is easily encoded as an annotated disjunction $\alpha :: A; (1-\alpha) :: A'$, where $A'$ is a fresh atom only appearing in this rule, so that allowing probabilistic facts does not increase the modeling power of our language. The converse is also true: an annotated disjunction with sharp probabilities such as $p_1 :: a_1; \ldots ; p_n :: a_n$ can be translated into an equivalent (acyclic) subprogram formed by a set of probabilistic facts $q_1 :: c_1, \ldots, q_n :: c_n$ and (non probabilistic) logic rules $a_i \ :- \ \text{not}\, c_1, \ldots, \text{not}\, c_{i-1}, c_i$, for $i = 1, \ldots, n$ [14]. The probabilities $q_i$ are set to $p_i / \prod_{j<i} q_j = p_i / (1 - \sum_{j<i} p_i)$. For example,

```
0.2::red;0.3::green;0.5::blue.
```

can be translated into

```
0.2::c1.  3/8::c2.  1.0::c3.
red   :- c1.
green :- not c1, c2.
blue  :- not c1, not c2, c3.
```

The latter induces the same (joint) distribution over `red`, `green` and `blue` as the annotated disjunction.

Hence, with sharp probabilities, annotated disjunctions are no more expressive than probabilistic facts. That transformation does not extend easily to the the case of interval-valued probabilities. For example,

```
[0.1,0.3]::red;[0.2,0.4]::green;[0.4,0.6]::blue.
```

is not equivalent to the program

$$[0.1,0.3] :: \texttt{c1} . \quad [2/7,4/9] :: \texttt{c2} . \quad [\alpha_3, \beta_3] :: \texttt{c3} .$$
```
red   :- c1.
green :- not c1, c2.
blue  :- not c1, not c2, c3.
```

Here, the interval $[\alpha_2, \beta_2] = [2/7, 4/9]$ for `c2` is defined as $\alpha_2 = 0.2/(1-\beta_1)$ and $\beta_2 = 0.4/(1-\alpha_1)$, where $[\alpha_1, \beta_1] = [0.1, 0.3]$. These numbers produce correct marginal lower and upper probabilities for `red` and `green`, which are not affected by the choice of values for $\alpha_3$ and $\beta_3$. Hence, if we modify the intervals of `c1` or `c2` (with the rest fixed), we produce the wrong lower probability. Now, $\underline{\Pr}(\texttt{blue}) =$

$(1-\overline{\Pr}(\texttt{c1}))(1-\overline{\Pr}(\texttt{c2}))\underline{\Pr}(\texttt{c3})$, whence $\alpha_3 > 1$. Similarly, we have that $\beta_3 < 1$. Hence, there are not values for $\alpha_i$ and $\beta_i$, $i = 1, 2, 3$, that jointly achieve the desired marginals.

We can however take a different route, by noting that interval-valued annotated disjunctions specify themselves (local) credal sets, hence they can also be reduced to a combination of one vacuous interval-valued annotated disjunction and a set of precise annotated disjunctions, by Theorem 2. These precise annotated disjunctions can then be transformed to probabilistic facts and non-probabilistic normal rules. Moreover, the vacuous interval-valued credal set can be encoded as vacuous probabilistic facts. Therefore, we can convert any interval-valued annotated disjunction to an equivalent acyclic program containing only (vacuous and precise) probabilistic facts and non-probabilistic rules.

To illustrate the translation, consider again that same interval-valued annotated disjunction over atoms `red`, `green` and `blue`. We can represent the same model by enumerating the vertices using probabilistic facts:

```
[0,1]::v1. [0,1]::v2. [0,1]::v3.
v(1) :- v1, v2, v3.
v(2) :- not v1, v2, v3.
v(3) :- v1, not v2, v3.
v(4) :- not v1, not v2, v3.
v(5) :- v1, v2, not v3.
v(6) :- not v1, v2, not v3.
v(6) :- not v2, not v3.

0.1::w1(1);0.3::w2(1);0.6::w3(1).
0.1::w1(2);0.4::w2(2);0.5::w3(2).
0.3::w1(3);0.2::w2(3);0.5::w3(3).
0.2::w1(4);0.2::w2(4);0.6::w3(4).
0.2::w1(5);0.4::w2(5);0.4::w3(5).
0.3::w1(6);0.3::w2(6);0.4::w3(6).

red   :- w1(X), v(X).
green :- not w1(X), w2(X), v(X).
blue  :- not w1(X), not w2(X), w3(X), v(X).
```

The annotated disjunctive rules specifying the vertices can additionally be converted to probabilistic facts (and some rules), as explained. Hence, we have:

**Theorem 3** *Any probabilistic answer set program with interval-valued annotated disjunctions can be converted into an equivalent program containing only interval-valued probabilistic facts and non-probabilistic rules. If the original program is acyclic (resp., nondisjunctive), the resulting program is also acyclic (resp., nondisjunctive).*

Note that unlike the translation of annotated disjunctions into probabilistic facts, the translation stated by the Theorem above is very inefficient, as it requires enumerating all vertices of (the credal set induced by) an interval-valued annotated disjunction.

### 4.3. Relational Programs

The semantics of a PASP program with variables is given by the semantics of its grounding. Thus, for instance, the semantics of the program

```
urn(a). urn(b).
[0.3,0.6]::red(X);[0.4,0.7]::green(X) :- urn(X).
```

is the semantics of the ground program

```
urn(a). urn(b).
[0.3,0.6]::red(a);[0.4,0.7]::green(a) :- urn(a).
[0.3,0.6]::red(b);[0.4,0.7]::green(b) :- urn(b).
```

An alternative interpretation would be to consider non-ground annotated disjunctive rules as specifying a set of grounded annotated disjunctive rules, all which share the same distributions. For instance, we might interpret the program

```
urn(a). urn(b).
[0.3,0.6]::red(X);[0.4,0.7]::green(X) :- urn(X).
```

as intending

```
urn(a). urn(b).
α::red(a);1-α::green(a) :- urn(a).
α::red(b);1-α::green(b) :- urn(b).
```

for some $\alpha \in [0.3, 1]$. That leads to ground programs that cannot be written in the language we specified before. Inference in under such semantics is more challenging, as, for example, upper and lower bounds are not necessarily attained at the local extrema of intervals. We leave the analysis of such alternative semantics as future work.

### 4.4. Linear Inequalities

Even though (vacuous) interval-valued annotated disjunctions suffice to represent any credal set, they can ensure such an expressivity only by enumeration of the extreme distributions. This is often computationally demanding and cause a blown up in program size. To avoid that, we propose *parameterized annotated disjunctive rules*, which are expressions of the form

$$X_1 :: A_1 ; \ldots ; X_m :: A_m \texttt{ :- } B_1, \ldots, B_n \texttt{ .}$$

where $X_i$ are *probability variables* (i.e., strings beginning with upper case letters), and each $B_i$ is either a standard literal or a linear (in)equality involving only the probability variables in the head and numeric constants (i.e., rational numbers). For example,

```
P::a;Q::b;R::c :- P < Q, Q < R, P >= 0.3
```

is a valid parameterized annotated disjunction encoding the credal set of probability distributions over a random variable $X$ taking values on $\{a, b, c\}$, with

$$0.3 \leq \Pr(X = a) < \Pr(X = b) < \Pr(X = c) \leq 1.$$

Here is another example, where we have an urn containing at least twice as many red balls as green balls (and none of any other color):

```
P::red;Q::green :- P >= 2*Q.
```

This is equivalent to:

```
[2/3,1]::red;[0,1/3]::green.
```

The semantics of PASP programs with parameterized annotated disjunctive rules intuitively extends the semantics of programs with interval probabilities. Thus, in Equation (5), we replace the intervals with the more general constraints given by linear probability inequalities. Note that, as it happens with standard annotated disjunctions, the constraints for different annotated disjunctions are taken to denote different probability constraints. For example, the rules

```
P::a;Q::b :- P > Q.
P::a;Q::b :- P <= Q.
```

specify two separately credal sets, one of distributions where $\Pr(\texttt{a}) > \Pr(\texttt{b})$ and other where $\Pr(\texttt{a}) \leq \Pr(\texttt{b})$. The resulting program will induce marginal probabilities $\Pr(\texttt{a})$ that will not satisfy either constraint.

The use of parameterized probabilities (a.k.a. probability variables) extends the use of annotated disjunctions with interval-valued probabilities, but is more expressive. For example, with such constructs, one can specify unconditional open credal sets such as $\Pr(\texttt{a}) > 0$.

As with interval-valued annotated disjunctive rules, the semantics of a non-ground program is the semantics of its grounding. Thus, for example, the program

```
urn(a). urn(b).
P::red(X);Q::green(X) :- P >= 0.3, urn(X).
```

is understood as the ground program

```
urn(a). urn(b).
P::red(a);Q::green(a) :- P >= 0.3, urn(a).
P::red(b);Q::green(b) :- P >= 0.3, urn(b).
```

Note that logical variables are resolved before probability variables are even considered. Thus, we interpret

```
prop(0.8).     P::draw(P) :- prop(P).
```

as the ground program

```
prop(0.8).     P::draw(0.8) :- prop(0.8).
```

In that program, `P` is free to take any value in $[0, 1]$, and `draw(0.8)` is true with probability `P`. While the semantics of such programs are well-defined, they are certainly confusing and should well be avoided.

Because logical variables are grounded by the time we assign a semantics to the program, we might as well allow logical variables to interact with probability variables. For example, we might have a program such as

```
urn(30,70).
P::red;Q::green :- P >= R/(R+O), urn(R,O).
```

whose grounding is

```
urn(30,70).
P::red;Q::green :- P >= 30/(30+30), urn(30,30).
P::red;Q::green :- P >= 30/(30+70), urn(30,70).
P::red;Q::green :- P >= 70/(70+30), urn(70,30).
P::red;Q::green :- P >= 70/(70+70), urn(70,70).
```

Of course, we might run into problems when probability variables are combined with nonnumerical constants, such as for instance if we added the fact `urn(a,b)`. To avoid such complications, we do not pursue this possibility here.

### 4.5. Credal Networks

Any Bayesian network over discrete random variables can be specified as an acyclic probabilistic logic program, with an almost direct translation that associates root nodes with annotated disjunctions and inner nodes with annotated disjunctive rules. For example, consider a simple network with edges $X \rightarrow Y$ and $Z \rightarrow Y$, where $X$ and $Z$ are binary and $Y$ is ternary. The conditional probability values $\Pr(Y = i \mid X = j, Z = k)$ are given by table $T_k(i, j)$ below:

$$T_1 = \begin{pmatrix} 0.3 & 0.9 \\ 0.1 & 0.05 \\ 0.6 & 0.05 \end{pmatrix}, \qquad T_2 = \begin{pmatrix} 0.2 & 0.05 \\ 0.3 & 0.05 \\ 0.5 & 0.9 \end{pmatrix}.$$

That network can be specified by a program with probabilistic facts `x` and `z`, and the annotated disjunctive rules:

```
0.30::y1;0.10::y2;0.60::y3 :- not x, not z.
0.90::y1;0.05::y2;0.05::y3 :- x, not z.
0.20::y1;0.30::y2;0.50::y3 :- not x, z.
0.05::y1;0.05::y2;0.90::y3 :- x, z.
```

The same process can be adapted to translate a separately specified credal network into an acyclic probabilistic program. Root nodes are specified as parameterized annotated disjunctions and inner nodes are specified as parameterized annotated disjunctive rules whose body are the configurations of parent variables.

We can also benefit from logic programming to represent richer types of conditional credal sets. For instance, we can easily specify extensive conditional credal sets. To see this, consider a simple extensive credal network with graph $X \rightarrow Y$ such that $X$ is binary and the conditional credal set of $Y$ is extensively defined by the distributions $T_1$ and $T_2$; in other words, the selection of a conditional probability distribution $\Pr(Y|x)$ constrains the choice of the conditional distribution $\Pr(Y|\neg x)$ to the one in the same table $T_z$. We can specify such a model by augmenting the previous program with a vacuous fact that selects a table:

```
[0,1]::z.
```

Previous work has shown that any acyclic probabilistic logic program with only probabilistic facts and logic rules specifies a Bayesian network whose graph is the dependency

graph of the program and whose conditional probabilities are immediately derived from the facts and rules [6]. This way, acyclic (propositional) programs and binary Bayesian networks specify the same class of models (if we ignore the specification of CPTs).

By a much similar argument, one can show that an acyclic program containing only interval-valued probabilistic facts and non-probabilistic rules can be interpreted as a credal network over the same dependency graph. Since a program with (parameterized) annotated disjunctions can be translated into an equivalent program with only (interval-valued) probabilistic facts and logic rules, we have that:

**Theorem 4** *The semantics of an acyclic probabilistic answer set program is given by a credal network; if only probabilistic facts and nonprobabilistic rules appear, the network structure is the dependency graph of the program.*

**Proof** For simplicity, we consider only programs with interval-valued probabilistic facts; annotated disjunctions can be compiled into probabilistic facts, as discussed. We also assume that no atom appears in more than one probabilistic fact; if that is not the case, then we create a new atom to encode the event "one of the occurrences of the fact is true". Given such a program, we obtain a credal network using essentially the same enconding used for precise probabilistic programs [7], except that we associate root nodes with intervals instead of sharp probabilities. Note that the resulting network has imprecise root nodes and deterministic inner nodes. ■

## 5. Inferential Complexity

We are often interested in computing $\underline{\Pr}(a \mid b)$ for arbitrary atoms $a$ and $b$ in a PASP program with imprecisely specified probabilities.

The complexity of computing the lower or upper probability of an atom in a precise PASP program has been thoroughly investigated elsewhere in terms of program dependency structure (negation, acyclicity, stratification) and the language richness (disjunction, aggregation, variables, etc) [6, 8, 18]. We only note here that the complexity of such an inference in nondisjunctive stratified programs is $PP$-complete, and climbs up to $PP^{NP}$-complete for programs with negation or disjunction (but not both) to $PP^{\Sigma_2^P}$-complete for disjunctive normal programs.

As with other imprecise probability extensions of (classical) probabilistic models, allowing imprecisely specified probabilistic constructs adds to the computational power of the inferential machine, which reflects in higher complexity. In fact, Bueno et al. [4, Theorem 2] showed that the computation of a lower probability in a PASP program with interval-valued probabilistic facts is $NP^{PP}$-hard. The result is intuitive: to calculate the desired probability bound, one must first nondeterministically select an extreme probability value for each probabilistic fact, then run inference it the resulting (precise) probabilistic program. The reason we don't get a higher complexity in the oracle (say, $NP$ with a $PP^{\Sigma_2^P}$ oracle) is Toda's celebrated result [30] that $P^{PP}$ encompasses the entire polynomial complexity hierarchy (thus such "hierarchy" collapses to its first level as an oracle machine).

The following result shows that the situation is unaltered by the adoption of linear probability constraints in rules.

**Theorem 5** *Deciding whether the unconditional lower probability of an atom is above a given threshold is $NP^{PP}$-complete in probabilistic answer set programs with parameterized annotated disjunctions.*

**Proof** Hardness follows from [4]. To show membership, note that inference can be cast as a multilinear program over the probabilities of parameterized annotated disjunctions. Thus, the lower probability is attained at a combination of the vertices of the local polytopes defined by linear probability inequalities. To compute the inference nondeterministically guess a local vertex, for each parameterized annotated disjunction, then solve the resulting inference problem by a call to a $PP^{\Sigma_2^P}$ oracle; the latter is equivalent to polynomially many calls to a $PP$ oracle [30]. ■

A conditional lower probability query $\underline{\Pr}(a|b)$ can be reduced to the computation of (several) unconditional lower probability bounds, by means of the Generalized Bayes Rule (GBR) [33]. According to the rule, $\underline{\Pr}(a|b)$ is the unique solution $\mu$ of:

$$\min_{\Pr} \Pr(a,b) - \mu \Pr(b) = 0 \Leftrightarrow \min_{\Pr} \Pr(a,b) + \mu \Pr(\neg b) = \mu.$$

To compute the left hand side of the second equation for a fixed $\mu$, we extend the probabilistic program with the rules

$$q :- a, b \quad \text{and} \quad \mu :: q :- \mathsf{not}\, b,$$

where $q$ is a fresh atom. Then $\underline{\Pr}(q) = \mu$ iff $\Pr(a,b) + \mu \Pr(\neg b) = \mu$. Now to find the desired conditional lower probability we only need to perform a binary search for $\mu \in [0,1]$ that solves the GBR's equation. We thus get the following corollary:

**Corollary 6** *Deciding the conditional lower probability of an atom in a probabilistic answer set program with parameterized annotated disjunctions is $NP^{PP}$-complete.*

## 6. Decision Making

One use of interval-valued annotated disjunctions is to encode decisions that an agent can make. For example, in

the Monty-Hall encoding, we might specify whether the participant accepts or not the offer to change doors as

```
[0,1]::change;[0,1]::keep.
```

Then we can re-state the outcomes based on that decision:

```
win_keep :- select(X), prize(X), keep.
win_change :- choice(X), prize(X), change.
```

Computing e.g. $\underline{\Pr}(\texttt{win\_keep})$ and $\overline{\Pr}(\texttt{win\_keep})$ thus gives us the minimum and maximum probability of a strategy: selecting a door and deciding to change or not.

Van den Broeck et al. [13] extended Problog's syntax to cope with such decision making situations. They introduced *decision facts*, denoted as ? :: $A$ where $A$ is a fact, to represent an agent's decisions. They also introduced *utility attributes*, specified by the special two-place predicate $\texttt{util}(A, U(A))$, which maps an atom $A$ to a real value $U(A)$. The objective is to find an optimal strategy, that is, a selection of 0/1 probabilities to decision facts that maximizes the sum of the utilities of satisfied atoms.

Here is an illustrative example from [13] of a decision problem encoded in DTProblog:

```
?::umbrella. ?::raincoat.  0.3::rainy. 0.5::windy.

broken :- umbrella, rainy, windy.
dry :- umbrella, not broken.
dry :- raincoat.
dry :- not rainy.
util(umbrella,-2). util(raincoat,-20).
util(dry,60). util(broken,-40).
```

The optimal strategy for the program is $\sigma(\texttt{umbrella}) \mapsto 1$ and $\sigma(\texttt{raincoat}) \mapsto 0$, which implies $\Pr(\texttt{dry}) = 0.85$, $\Pr(\texttt{broken}) = 0.15$, $\Pr(\texttt{umbrella}) = 1$, $\Pr(\texttt{raincoat}) = 0$, and hence in the expected utility $(-2) \cdot 1 + (-40) \cdot 0.15 + (-20) \cdot 0 + 60 \cdot 0.85 = 43$.

We can represent DTProblog programs as stratified PASP programs by transforming decision facts into vacuous facts and by encoding the additive utility function as a probability model, as described in [19]. To perform the converse, translate each interval-valued annotated disjunction into a series of decision facts and specify a utility as an indicator function of the query atom. It thus follows that:

**Theorem 7** *The computation of a maximum expected utility in a DTProblog program can be efficiently reduced to an unconditional inference in a stratified PASP program with interval-valued annotated disjunctions, and vice-versa.*

Conditional inferences in imprecise PASP programs can also be reduced to the (several) computation(s) of maximum expected utility of DTProblog programs by encoding the binary search of GBR as discussed in Section 5. Combined, these arguments show that:

**Theorem 8** *The computation of maximum expected utility for DTProblog programs is $NP^{PP}$-complete.*

**Proof** Membership follows from Theorem 7. Hardness follows as the inferential complexity of DTProblog is $NP^{PP}$-complete, as this is the complexity of inference of PASP programs with imprecise probabilistic facts [4]. ■

To our knowledge, the complexity of DTProblog inference has not been obtained previously.

## 7. Conclusion

We have discussed extensions of probabilistic answer set programming languages that allow for a rich class of imprecise probability models. In particular, we have proposed a semantics for programs whose probabilistic choices are annotated either with intervals or with linear inequality constraints over probability values.

In many ways, this work is preliminary. For instance, one important point that we have left out of this discussion is how to effectively perform inference with imprecisely-specified probabilistic answer set programs. In fact, we have implemented the probabilistic answer set programs with interval-valued probabilistic facts in an open-source package, available at http://kamel.ime.usp.br/dpasp/. Currently, unconditional inferences are performed by vertex enumeration, and interval-valued annotated disjunctive rules are not supported. Future work must devise more efficient algorithms that allow for faster inference, and that support also linear probability constraints. One possibility is to combine ideas from knowledge compilation [9] and inference in credal networks [1].

There are also theoretical questions about the complexity of other types of inferences (e.g., maximum a posteriori) with imprecise programs, as well as of lower/upper probability queries in programs with richer constructs (variables, aggregates, etc), as has been explored for the case of programs with sharp probabilities [18]. We hope this paper instigates research in that direction.

## Acknowledgments

The first author is partially supported by São Paulo Research Agency grant #2022/02937-9 and CNPq grant #305136/2022-4. The second author is partially supported by CNPq grant #305753/2022-3. Both authors received generous support from the C4AI (supported by FAPESP grant 2019/07665-4 and IBM corporation) and CAPES Finance Code 001.

## Author Contributions

Both authors contributed with ideas, discussion and revision of the text; results and text were produced primarily by the first author.

## References


[1] Alessandro Antonucci, Cassio Polpo de Campos, David Huber, and Marco Zaffalon. Approximate credal network updating by linear programming with applications to decision making. *International Journal of Approximate Reasoning*, 58:25–38, 2015.

[2] Thomas Augustin. Generalized basic probability assignments. *International Journal of General Systems*, 34(4):451–463, 2005.

[3] Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(1):57–144, 2009.

[4] Thiago P. Bueno, Denis D. Mauá, Leliane N. Barros, and Fabio G. Cozman. Modeling Markov decision processes with imprecise probabilities using probabilistic logic programming. In *Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications*, pages 49–60, 2017.

[5] Fabio Gagliardi Cozman. Credal networks. *Artificial intelligence*, 120(2):199–233, 2000.

[6] Fabio Gagliardi Cozman and Denis Deratani Mauá. Probabilistic graphical models specified by probabilistic logic programs: Semantics and complexity. In *Proceedings of the 8th International Conference on Probabilistic Graphical Models*, pages 110–122, 2016.

[7] Fabio Gagliardi Cozman and Denis Deratani Mauá. On the complexity of propositional and relational credal networks. *International Journal of Approximate Reasoning*, 83:298–319, 2017.

[8] Fabio Gagliardi Cozman and Denis Deratani Mauá. On the semantics and complexity of probabilistic logic programs. *Journal of Artificial Intelligence Research*, 60:221–262, 2017.

[9] Fabio Gagliardi Cozman and Denis Deratani Mauá. The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. *International Journal of Approximate Reasoning*, 125: 218–239, 2020.

[10] Eugene Dantsin. Probabilistic logic programs and their semantics. In *Proceedings of the 1st Russian Conference on Logic Programming*, pages 152–164, 1992.

[11] Luis M. de Campos, Juan F. Huete, and Serafín Moral. Probability intervals: a tool for uncertain reasoning. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2:167–196, 1994.

[12] Eduardo Menezes De Morais and Marcelo Finger. Probabilistic answer set programming. In *Proceedings of the 2013 Brazilian Conference on Intelligent Systems*, pages 150–156, 2013.

[13] Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, and Luc De Raedt. DTProbLog: A decision-theoretic probabilistic prolog. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1217–1222, 2010.

[14] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, 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, 2015.

[15] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning (SLAIML). Springer Cham, 2013.

[16] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, 1988.

[17] Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning*, pages 145–154, 2016.

[18] Denis Deratani Mauá and Fabio Gagliardi Cozman. Complexity results for probabilistic answer set programming. *International Journal of Approximate Reasoning*, 118:133–154, 2020.

[19] Denis Deratani Mauá, Cassio Polpo de Campos, and Marco Zaffalon. The complexity of approximately solving influence diagrams. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 604–613, 2012.

[20] Enrique Miranda and Sébastien Destercke. Extreme points of the credal sets generated by comparative probabilities. *Journal of Mathematical Psychology*, 64–65:44–57, 2015.

[21] Enrique Miranda, Gert de Cooman, and Inés Couso. Lower previsions induced by multi-valued mappings. *Journal of Statistical Planning and Inference*, 133(1): 173–197, 2005.

[22] Hung T. Nguyen. On random sets and belief functions. *Journal of Mathematical Analysis and Applications*, 65(3):531–542, 1978.

[23] Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[24] David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94:7–56, 1997.

[25] David Poole. Abducing through negation as failure: Stable models within the independent choice logic. *Journal of Logic Programming*, 44(1–3):5–35, 2000.

[26] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2462–2467, 2007.

[27] Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*. River Publishers, 2018.

[28] Tiasuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming*, pages 715–729, 1995.

[29] Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[30] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5): 865–877, 1991.

[31] Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe. Logic programs with annotated disjunctions. In *Proceedings of the 20th International Conference on Logic Programming*, pages 431–445, 2004.

[32] Joost Vennekens, Marc Denecker, and Maurice 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.

[33] Peter Walley. *Statistical Reasoning with Imprecise Probabilities*. Chapman and Hall, 1991.

[34] Larry A. Wasserman and Joseph B. Kadane. Bayes' theorem for choquet capacities. *The Annals of Statistics*, 18(3):1328–1339, 1990.