Statistical Statements in Probabilistic Logic Programming

Damiano Azzolini[1][0000−0002−7133−2673], Elena Bellodi[2][0000−0002−3717−3779], and Fabrizio Riguzzi[1][0000−0003−1654−9703]

[1] Dipartimento di Matematica e Informatica – Università di Ferrara
Via Saragat 1, 44122, Ferrara, Italy
{damiano.azzolini,fabrizio.riguzzi}@unife.it
[2] Dipartimento di Ingegneria – Università di Ferrara
Via Saragat 1, 44122, Ferrara, Italy
elena.bellodi@unfe.it

Abstract. Probabilistic Logic Programs under the distribution semantics (PLPDS) do not allow statistical probabilistic statements of the form "90% of birds fly", which were defined "Type 1" statements by Halpern. In this paper, we add this kind of statements to PLPDS and introduce the PASTA ("Probabilistic Answer set programming for STAtistical probabilities") language. We translate programs in our new formalism into probabilistic answer set programs under the credal semantics. This approach differs from previous proposals, such as the one based on "probabilistic conditionals" as, instead of choosing a single model by making the maximum entropy assumption, we take into consideration all models and we assign probability intervals to queries. In this way we refrain from making assumptions and we obtain a more neutral framework. We also propose an inference algorithm and compare it with an existing solver for probabilistic answer set programs on a number of programs of increasing size, showing that our solution is faster and can deal with larger instances.

Keywords: Probabilistic Logic Programming, Statistical Statements, Statistical Relational Artificial Intelligence

1 Introduction

Probabilistic Logic Programming (PLP) [19] extends Logic Programming (LP) by considering various probabilistic constructs. ProbLog [8] is an example of a PLP language based on the distribution semantics (PLPDS) [20]. This semantics assumes that every program has a two-valued well-founded model [24].

In giving a semantics to First-Order knowledge bases, Halpern [13] distinguishes statistical statements from statements about degrees of belief, and presents two examples: "the probability that a randomly chosen bird flies is 0.9" and "the probability that Tweety (a particular bird) flies is 0.9". The first statement captures statistical information about the world while the second captures

a degree of belief. The first type of statement is called "Type 1" while the latter "Type 2". The first statement can be read as "90% of the population of birds flies".

The distribution semantics allows statements stating that, if a specific x is a bird, then x flies with probability 0.9 (or it does not with probability 0.1). In fact, the semantics of general rules of the form `0.9::flies(X) :- bird(X).` is given by the set of its ground instantiations of the form `0.9::flies(x) :- bird(x).`, which has the just described meaning. In this paper, we aim at adding to PLPDS the possibility of expressing "Type 1" statements, exploiting for this purpose Probabilistic Answer Set Programming.

Answer Set Programming (ASP) [5] is a powerful rule-based language for knowledge representation and reasoning. An extension to ASP that manages uncertain data is Probabilistic Answer Set Programming (PASP). The credal semantics [6] assigns a probability range to every query to probabilistic answer set programs - instead of a sharp value as in PLPDS - where lower and upper probability bounds are computed by analyzing the stable models of every world.

"Type 1" statements are called "probabilistic conditionals" in [15], where they are given a semantics in terms of the principle of maximum entropy: the unique model with maximum entropy is chosen. Instead of selecting only one model, we keep all models at the cost of inferring a probability interval instead of a sharp probability. We think this is of interest because it avoids making the rather strong maximum entropy assumption.

We propose a new language, called PASTA for "Probabilistic Answer set programming for STAtistical probabilities", where we exploit the credal semantics to take into account "Type 1" statements in PLPDS. In particular, probabilistic conditionals are converted into an ASP rule plus two constraints: the rule characterizes the elements of the domain while the constraints inject the statistical information on the possible stable models of every world. To perform exact inference under this semantics we developed an algorithm, taking the same name of the language, which returns lower and upper bounds for the probability of queries, and compared it with PASOCS [23]. The results show that, if we preprocess the input program into a form that allows reasoning about its structure, it is possible to obtain better performance on every program we tested.

The paper is structured as follows: in Section 2 we review the basic knowledge relative to ASP, PLPDS, the credal semantics, and probabilistic conditionals. In Section 3 we describe the PASTA language. In Section 4 we introduce an algorithm to perform exact inference on PASTA programs, that is experimentally tested in Section 5. Section 6 surveys related work and Section 7 concludes the paper.

2 Background

2.1 Answer Set Programming

We expect the reader to be familiar with the basic concepts of Logic Programming and First-Order Logic. We consider here also *aggregate atoms* [1] of the

form $g_0 \diamond_0 \#f\{e_1; \ldots; e_n\} \diamond_1 g_1$, where f is an aggregate function symbol, $\diamond_0$ and $\diamond_1$ are arithmetic comparison operators, and g_0 and g_1 are constants or variables called *guards*; each e_i is an expression of the form $t_1, \ldots, t_l : F$, where F is a conjunction of literals and $t_1, \ldots, t_l$, with $l > 0$, are terms whose variables appear in F. $g_0 \diamond_0$ or $g_1 \diamond_1$ or even both can be omitted. Moreover, $\diamond_0$ and $\diamond_1$ can be omitted as well and, if omitted, are considered equivalent to $\leq$. A *disjunctive rule* (or simply *rule*) is an expression of the form

$$H_1 \vee \cdots \vee H_m \leftarrow B_1, \ldots, B_n$$

where each H_i is an atom and each B_i is a literal. $H_1 \vee \cdots \vee H_m$ is the *head* of the rule and $B_1, \ldots, B_n$ is the body. We will usually replace $\vee$ with `;` and $\leftarrow$ with `:-` when describing actual code. We consider only *safe* rules, where all variables occur in a positive literal in the body. If $m = 0$ and $n > 0$, the rule is an *integrity constraint*. Facts can also be defined through a range with the notation `f(a..b)`, where both `a` and `b` are integers. A rule is *ground* when it does not contain variables. A *program*, also called *knowledge base*, is a finite set of rules. Given an answer set program $\mathcal{P}$, we define its *Herbrand base* (denoted with $B_\mathcal{P}$) as the set of all ground atoms that can be constructed using the symbols in the program. An *interpretation* I for $\mathcal{P}$ is a set such that $I \subset B_\mathcal{P}$. An interpretation I *satisfies* a ground rule if at least one head atom is true in I when the body is true in I. If an interpretation satisfies all the groundings of all the rules of a program it is called a *model*. Given a ground program $\mathcal{P}_g$ and an interpretation I we call *reduct* [10] of $\mathcal{P}_g$ with respect to I the program obtained by removing from $\mathcal{P}_g$ the rules in which a literal in the body is false in I. An interpretation I is an *answer set* (also called *stable model*) for $\mathcal{P}$ if I is a minimal model (under set inclusion) of the reduct of $\mathcal{P}_g$. We denote with $AS(\mathcal{P})$ the set of all the answer sets of a program $\mathcal{P}$. Sometimes, not all the elements of an answer set are needed, so we can project the computed solution into a set of atoms. That is, we would like to compute the *projective solutions* [12] given a set of ground atoms V, represented by the set $AS_V(\mathcal{P}) = \{A \cap V \mid A \in AS(\mathcal{P})\}$. An atom a is a *brave consequence* of a program $\mathcal{P}$ if $\exists A \in AS(\mathcal{P})$ such that $a \in A$. We denote the set containing all the brave consequences with $BC(\mathcal{P})$. Similarly, a is a *cautious consequence* if $\forall A \in AS(\mathcal{P}),\ a \in A$, and we denote the set containing all the cautious consequences with $CC(\mathcal{P})$.

Example 1 (Bird). Consider the following answer set program $\mathcal{P}$:

```
bird(1..4).
fly(X) ; not_fly(X):- bird(X).
:- #count{X:fly(X),bird(X)} = FB,
   #count{X:bird(X)} = B, 10*FB < 6*B.
```

The first line states that there are 4 birds, indexed with 1, 2, 3, and 4. The disjunctive rule states that a bird `X` can fly or not fly. In the constraint, the first aggregate counts the flying birds and assigns this value to `FB`, while the second aggregate counts the birds and assigns the result to `B`. Overall, the constraint imposes that at least 60% of the birds fly (we converted the values into

integers since ASP cannot easily manage floating point numbers). This program has 5 answer sets, $BC(\mathcal{P})$ = {b(1) b(2) b(3) b(4) f(1) nf(1) f(2) nf(2) f(3) nf(3) f(4) nf(4)}, $CC(\mathcal{P})$ = {b(1) b(2) b(3) b(4)}, and $AS_V(\mathcal{P})$ = {{b(1) b(2) b(3) b(4)}} where b/1 stands for bird/1, f/1 for fly/1, nf/1 for not_fly/1 and V = {b(1),b(2),b(3),b(4)}.

2.2 Probabilistic Logic Programming

In LP, a large body of work has appeared for allowing probabilistic reasoning. One of the most widespread approaches is the distribution semantics (DS) [20] according to which a probabilistic logic program defines a probability distribution over normal logic programs called *worlds*. The DS underlies many languages such as ProbLog [8]. Following the ProbLog syntax, probabilistic facts take the form $\Pi :: f.$ where f is a fact and $\Pi \in]0, 1]$. For example, with 0.9::fly(tweety). we are stating that the probability that tweety flies is 0.9, i.e., we believe in the truth of fly(tweety) with probability 0.9. This is a"Type 2" statement.

An *atomic choice* indicates whether a grounding $f\theta$, where θ is a substitution, for a probabilistic fact $\Pi :: f$ is selected for a world or not, and it is represented with the triple (f, θ, k) where k can be 1 (fact selected) or 0 (fact not selected). A *composite choice* is a consistent set of atomic choices, i.e., only one choice can be made for a single ground probabilistic fact. The probability of a composite choice κ can be computed with the formula:

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

If a composite choice contains one atomic choice for every grounding of each probabilistic fact, it is called a *total* composite choice or *selection*, and it is usually indicated with σ. Every selection identifies a normal logic program w called *world* composed of the rules of the program and the probabilistic facts that correspond to atomic choices with $k = 1$. Finally, the probability of a *query* q (a ground literal or a conjunction of ground literals) is computed as the sum of the probabilities of the worlds where the query is true:

$$P(q) = \sum_{w \models q} P(w) \tag{2}$$

where $P(w)$ is given by the probability of the corresponding selection (computed with Equation 1).

2.3 Credal Semantics

The DS considers only programs where each world has a two-valued well-founded model [24]. However, in the case of answer set programs, this often does not hold. When logic programs are not stratified, they may have none or several stable

models, in which case the well-founded model is not two-valued. If the program has multiple stable models, there are various available semantics: here we focus on the *credal* semantics [6,7]. Under this semantics, every query q is described by a lower and an upper probability, denoted respectively with $\underline{P}(q)$ and $\overline{P}(q)$, with the intuitive meaning that $P(q)$ lies in the range $[\underline{P}(q), \overline{P}(q)]$. If every world has exactly one stable model, $\overline{P}(q) = \underline{P}(q)$ and the credal semantics coincides with the DS. A world w contributes to the upper probability if the query is true in at least one of its stable models and to the lower probability if the query is true in all its stable models. In formulas,

$$\overline{P}(q) = \sum_{w_i | \exists m \in AS(w_i),\ m \models q} P(w_i),\ \underline{P}(q) = \sum_{w_i | \forall m \in AS(w_i),\ m \models q} P(w_i) \tag{3}$$

[6] also suggested an algorithm to compute the probability of q given evidence e (conditional probability). In this case, the upper conditional probability is given by

$$\overline{P}(q \mid e) = \frac{\overline{P}(q, e)}{\overline{P}(q, e) + \underline{P}(\neg q, e)} \tag{4}$$

If $\overline{P}(q, e) + \underline{P}(\neg q, e) = 0$ and $\overline{P}(\neg q, e) > 0$, $\overline{P}(q \mid e) = 0$. If both $\overline{P}(q, e)$ and $\overline{P}(\neg q, e)$ are 0, this value is undefined. The formula for the lower conditional probability is

$$\underline{P}(q \mid e) = \frac{\underline{P}(q, e)}{\underline{P}(q, e) + \overline{P}(\neg q, e)} \tag{5}$$

If $\underline{P}(q, e) + \overline{P}(\neg q, e) = 0$ and $\overline{P}(q, e) > 0$, $\underline{P}(q \mid e) = 1$. As before, if both $\overline{P}(q, e)$ and $\overline{P}(\neg q, e)$ are 0, this value is undefined.

2.4 Probabilistic Conditionals

Following [15], a probabilistic conditional is a formula of the form $K = (C \mid A)[\Pi]$ where C and A are First-Order formulas and $\Pi \in [0, 1]$. The intuitive meaning is: the number of individuals that satisfy C is $100 \cdot \Pi$ percent of the individuals that satisfy A.

Example 2 (Bird conditional.). Consider the following example, inspired by [25]:

```
bird(1)
(fly(X) | bird(X))[0.6]
```

The second statement says that, out of all the birds, 60% fly.

In this setting, [21] define a possible world w as an interpretation. Let Ω be the set of all possible worlds. A *probabilistic interpretation* P is a probability distribution over Ω, i.e., a function $P : \Omega \to [0, 1]$. Given a conjunction of ground literals q, $P(q) = \sum_{w \models q} P(w)$. The aggregating semantics states that a probability distribution P over interpretations is a *model of a First-Order*

formula if and only if $w \not\models F \implies P(w) = 0 \; \forall w$, and is a *model of a conditional* $K = (C \mid A)[\Pi]$ if and only if

$$\frac{\sum_{(C_i|A_i)\in G(K)} P(A_i, C_i)}{\sum_{(C_i|A_i)\in G(K)} P(A_i)} = \Pi \tag{6}$$

where $G(K)$ is the set containing all the ground instances of a conditional K. A probabilistic interpretation is a model for a knowledge base if it models all the formulas and all the conditionals. According to [18,25], the semantics of a knowledge base composed of probabilistic conditionals is given by the model with the highest entropy. The maximum entropy (MaxEnt) distribution for a knowledge base $\mathcal{K}$ is defined as:

$$P^{MaxEnt} = \arg\max_{P \models \mathcal{K}} - \sum_{w_i} P(w_i) \cdot log(P(w_i))$$

With this formulation, it is possible to assign a sharp probability value to every query. In this paper, we follow a different approach and we consider probabilistic conditionals as statistical "Type 1" statements [13], interpreting them under the credal semantics.

3 Probabilistic Answer set programming for STAtistical probabilities (PASTA)

A probabilistic conditional expresses statistical information about the world of interest, but we would like to avoid selecting a model making the maximum entropy assumption. We would rather consider all possible models and derive lower and upper bounds on the probability of queries using the credal semantics. Here, we consider "Type 1"/probabilistic conditionals of the form

$$(C \mid A)[\Pi_l, \Pi_u].$$

with a lower (Π_l) and an upper (Π_u) bound, with the intuitive meaning that the fraction of As that are also Cs is between Π_l and Π_u. Note that Π_l and Π_u can be vacuous, i.e., they can be respectively 0 and 1. We follow an approach based on the DS, so here worlds are ground programs. The meaning of the statement above is that the models of a world where the constraint

$$\Pi_l \leq \frac{\#count\{\mathbf{X} : C(\mathbf{X}), A(\mathbf{X})\}}{\#count\{\mathbf{X} : A(\mathbf{X})\}} \leq \Pi_u \tag{7}$$

does not hold should be excluded, where $\mathbf{X}$ is the vector of variables appearing in C and A.

We consider a program as being composed of regular rules, probabilistic facts, and conditionals of the previously described form, and we assign a semantics to it by translating it into a probabilistic answer set program. We call this language PASTA (Probabilistic Answer set programming for STAtistical probabilities).

Probabilistic facts and rules appear unmodified in the probabilistic answer set program. The conditional $(C \mid A)[\Pi_l, \Pi_u]$ is transformed into three answer set rules. The first is a disjunctive rule of the form `C;not_C:-A`. We require this rule to be safe. Then, we introduce two integrity constraints that mimic Equation 7 through aggregates: we count all the ground atoms that satisfy A (call this value ND) and A and C (call this value NN) and we impose that NN must be greater than or equal to $100 \cdot \Pi_l$ percent of ND and smaller than or equal to $100 \cdot \Pi_u$ percent of ND. The constraints are not generated if the bounds are vacuous. The conditional `(fly(X) | bird(X))[0.6]` of Example 2 is transformed into the rule and the constraint shown in Example 1. Finally, the probability interval of a query from a PASTA program is the probability interval of the query computed from the transformed probabilistic answer set program.

Example 3 (Bird probabilistic). Consider the program

```
0.4::bird(1..4).
(fly(X)|bird(X))[0.6].
```

This program is transformed into a probabilistic answer set program including four probabilistic facts, the rule, and the constraint from Example 1. There is only one constraint since the upper bound is vacuous. Consider the query $q =$ `fly(1)`. There are $2^4 = 16$ possible worlds. The query is false if `bird(1)` is false, so we can consider only $2^3 = 8$ worlds. There is 1 world with 4 birds, and it has 5 models. The query is true only in 4 of them, so we have a contribution of 0.4^4 to the upper probability. There are 3 worlds with 3 birds: these have 4 models each and the query is true in only three of them, so we have a contribution of $3 \cdot (0.4^3 \cdot (1 - 0.4))$ to the upper probability. There are 3 worlds with 2 birds: these have only one model and the query is true in it, so we have a contribution to both lower and upper probabilities of $3 \cdot (0.4^2 \cdot (1-0.4)^2)$. Finally, there is only 1 world with 1 bird, it has only 1 model and the query is true in it, so we have a contribution to both lower and upper probabilities of $0.4 \cdot (1-0.4)^3$. Overall, for the query `fly(1)` we get 0.2592 for the lower and 0.4 for the upper probability, so the probability lies in the range $[0.2592, 0.4]$. Similarly, by applying Formulas 4 and 5, the probability of the same query given evidence $e =$ `fly(2)` is in the range $[0.144, 0.44247]$ since $\underline{P}(q, e) = 0.0576$, $\overline{P}(q, e) = 0.16$, $\underline{P}(\neg q, e) = 0.2016$, and $\overline{P}(\neg q, e) = 0.3424$.

4 Inference in PASTA

By rewriting probabilistic conditionals as ASP rules, computing the probability of a query requires performing inference in PASP. To the best of our knowledge, the only system that allows (exact) inference in probabilistic answer set programs with aggregates is PASOCS [23], an implementation of the algorithm presented in [6]. The algorithm computes the probability of a query by generating all possible worlds (2^n, where n is the number of ground probabilistic facts in the program). For each world, it computes the brave and cautious consequences

(there is no need to compute all the answer sets). If the query is present in the brave consequences of a world, that world contributes to the upper probability. If the query is also present in the cautious consequences, that world also contributes to the lower probability. Despite its simplicity, this algorithm relies on the generation of all the possible worlds and does not take advantage of the structure of a program. For example, in Example 3, with query `fly(1)`, the probabilistic fact `bird(1)` must be true to get a contribution to the lower or upper probability, and so we can avoid generating the worlds where this fact is not present. Moreover, for both conditional and unconditional queries, we do not need to generate all the possible models for every world, we just need to check whether there is at least one model that satisfies the required constraints. To accommodate these ideas, we propose Algorithm 1, that we call PASTA like the language.

Consider first the problem of computing the probability of a query q (without evidence). We generate a non-probabilistic answer set program as follows. Every certain rule is kept unchanged. Every conditional is converted into three ASP rules as described in Section 3. Every ground probabilistic fact of the form `P::f` is converted into two rules of the form `f(P1):- f. not_f(1-P1):- not f.` where `P1` is `P` $\cdot 10^n$ (since ASP cannot manage floating point numbers). The atom `f` is then defined by a rule of the form `0{f}1.` Function ConvertProbFactsAndConditionals performs these conversions. Let us call the resulting program $PASP_p$. We then add to $PASP_p$ a constraint (line 4) imposing that the query must be true, represented with `:- not query.` (for Example 3 it becomes `:- not fly(1).`). We are not interested in all possible solutions, but only in the cautious consequences projected over the ground probabilistic facts, since we want to extract the probabilistic facts that are true in every answer set. These will constitute the *minimal* set of probabilistic facts. Function ComputeMinimalSet computes this set. These facts can be set to true since they are always present in the answer sets when the query is true, and so when there is a contribution to the probabilities. In the worst case, the resulting set will be empty. If we consider Example 3 and query `fly(1)`, the only atom (already converted as described before with $n = 3$) in this set will be `bird(1,400)`, so the corresponding probabilistic fact must be always true. After this step, we add to $PASP_p$ one integrity constraint for every element in the minimal set of probabilistic facts, to set them to true. Note that now $PASP_p$ does not contain the constraint imposed on the query in the previous step. For Example 3 and query `fly(1)`, we add `:- not bird(1,400).` to the program (line 9). Moreover, we add two more rules that indicate whether a model contains or not the query (line 11). For Example 3 and query `fly(1)` these are: `q:- fly(1). nq:- not fly(1).` Finally, we project the answer sets [12] to the probabilistic facts and atoms `q` and `nq`, since we need to consider only the truth values of the probabilistic facts to compute the probability of a world (line 13). The probabilistic facts present in the projected answer sets identify a world. Given an answer set, its probability (the probability of the world it represents) is given by the product of the probabilities of the probabilistic facts in it. Function ComputeContribu-

TION (line 18) computes the probability of every world and counts the models, the models where the query is true, the models where the query is false, the models where the query and evidence are true, and the models where the query is false and the evidence is true. For a query without evidence, the number of models where the query is true and the number of models where the query is false will only be either 0 or 1. To get the lower and upper probabilities, we apply Formulas 3. If we consider again Example 3 with query `fly(1)`, two of the possible projective solutions are:

```
b(1,400) b(2,400) b(3,400) b(4,400) nq
b(1,400) b(2,400) b(3,400) b(4,400) q
```

where, for the sake of brevity, `b/2` stands for `bird/2`. These two solutions show that the world with 4 birds has at least one model where the query is true and at least one model where the query is false, so it only contributes to the upper probability with $0.4 \cdot 0.4 \cdot 0.4 \cdot 0.4 = 0.0256$. Here, we also see the improvement given by computing the projective solutions: we only need to know whether the query is true or false in some models of a world, and not the exact number of models in which the query is true. For example, as shown in Example 1, the world with 4 birds has 5 models: 4 where the query is true and 1 where the query is false. However, to compute the probability bounds, it is not necessary to know the exact number: at most two stable models (one with the query true and one with the query false) for each world are needed instead of five. A difference with [23] is that PASOCS computes both brave and cautious consequences for every world, while PASTA computes projective solutions only once.

Consider now a conditional query. As before, we need to identify the minimal subset of probabilistic facts. However, we now add a constraint forcing the evidence (`ev`) to true instead of the query (line 6). We then add two more rules of the form `e:- ev.` and `ne:- not ev.` (line 15) and project the solutions also on the `e` and `ne` atoms (line 16). Finally, we analyse the resulting answer sets to compute the values that contribute to the lower (lp) and upper (up) probability, as described in Formulas 4 and 5.

5 Experiments

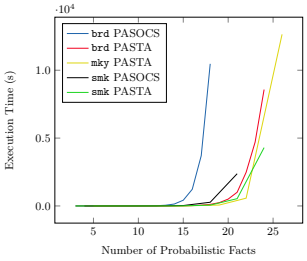
We implemented Algorithm 1 with Python3 using `clingo` [11] to compute answer sets[3]. We performed a series of experiments to compare PASTA with PASOCS [23]. For PASOCS, we use the single threaded mode and select exact inference. For PASTA, the execution time includes both the computation of the minimal set of probabilistic facts and the computation of the projective solutions. Usually, the time required for the first operation is negligible with respect to the computation of the probability. We selected three different programs described in [25]. The first program, `brd`, is `{(fly(X) | bird(X))[0.8,1],0.1::fly(1)}` with an increasing number of probabilistic facts `bird/1` with an associated probability of 0.5. The goal is to compute the probability of the query `fly(1)`.

[3] Source code and programs available at: https://github.com/damianoazzolini/pasta

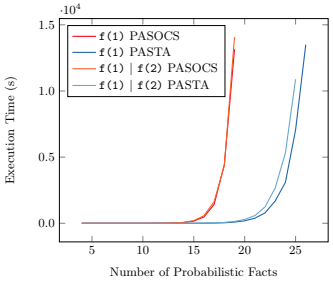
Algorithm 1 Function ComputeProbabilityBounds: computation of the probability bounds of a query *query* given evidence *ev* in a PASTA program $\mathcal{P}$.

function ComputeProbabilityBounds($query, ev, \mathcal{P}$)
  $probFacts, PASP_p \leftarrow$ ConvertProbFactsAndConditionals($\mathcal{P}$)
  if *ev is undefined* **then**
    $minSet \leftarrow$ ComputeMinimalSet($PASP_p \cup \{$:- *not query*.$\}$)
  else
    $minSet \leftarrow$ ComputeMinimalSet($PASP_p \cup \{$:- *not ev*.$\}$)
  end if
  for all $a \in minSet$ **do** ▷ a represents a probabilistic fact
    $PASP_p \leftarrow PASP_p \cup \{$:- *not a*.$\}$
  end for
  $PASP_p^q \leftarrow PASP_p \cup \{q$:- *query*., nq:- *not query*.$\}$
  if *ev is undefined* **then**
    $AS \leftarrow$ ProjectSolutions($PASP_p^q, probFacts, q \cup nq$)
  else
    $PASP_p^{qe} \leftarrow PASP_p^q \cup \{e$:- *ev*., ne:- *not ev*.$\}$
    $AS \leftarrow$ ProjectSolutions($PASP_p^{qe}, probFacts, q \cup nq \cup e \cup ne$)
  end if
  $worldsList \leftarrow$ ComputeContribution(AS)
  $lp \leftarrow 0,\ up \leftarrow 0$
  for all $w \in worldsList$ **do** ▷ Loop through answer sets
    if *ev is undefined* **then**
      if $w.modelQueryCounter > 0$ **then**
        $up \leftarrow up + P(w)$
        if $w.modelNotQueryCounter == 0$ **then**
          $lp \leftarrow lp + P(w)$
        end if
      end if
    else
      $up_{qe} \leftarrow 0,\ lp_{qe} \leftarrow 0,\ up_{nqe} \leftarrow 0,\ lp_{nqe} \leftarrow 0$
      if $w.modelQueryEvCounter > 0$ **then**
        $up_{qe} \leftarrow up_{qe} + P(w)$
        if $w.modelQueryEvCounter = w.models$ **then**
          $lp_{qe} \leftarrow lp_{qe} + P(w)$
        end if
      end if
      if $w.modelNotQueryEvCounter > 0$ **then**
        $up_{nqe} \leftarrow up_{nqe} + P(w)$
        if $w.modelNotQueryEvCounter = w.models$ **then**
          $lp_{nqe} \leftarrow lp_{nqe} + P(w)$
        end if
      end if
    end if
  end for
  if *ev is not undefined* **then**
    if $up_{qe} + lp_{nqe} == 0$ *and* $up_{nqe} > 0$ **then**
      $lp \leftarrow 0,\ up \leftarrow 0$
    else if $lp_{qe} + up_{nqe} == 0$ *and* $up_{qe} > 0$ **then**
      $lp \leftarrow 1,\ up \leftarrow 1$
    else
      $lp \leftarrow \frac{lp_{qe}}{lp_{qe}+up_{nqe}},\ up \leftarrow \frac{up_{qe}}{up_{qe}+lp_{nqe}}$
    end if
  end if
  return lp, up
end function

The second program, `mky`, represents the pair of conditionals `{(f(X) | h(X))[0.2,1],(f(X,Y) | h(Y),r(X,Y))[0.9,1]}`, with an increasing number of probabilistic facts `h/1` and `r/2`, both with an associated probability of 0.5. The distribution of the facts `r/2` follows a Barabási-Albert model, i.e., a graph, generated



(a) Inference times for the `brd`, `mky`, and `smk` experiments.



(b) Inference times for the `bird` experiments.

Fig. 1: Results for the experiments.

with the Python `networkx` package with parameter m_0 (representing the number of edges to attach from a new node to existing nodes) set to 3 and an increasing number of nodes. We randomly selected half of the total number of nodes to generate the `h/1` facts. The query is `f(0),f(0,1)`. The third program, `smk`, represents the conditional `{(smokes(Y) | smokes(X),friend(X,Y))[0.4,1]}` with an increasing number of probabilistic facts `friend/2` with an associated probability of 0.5, following the Barabási-Albert model. The query is `smokes(I)`, where `I` is a random node. For both `mky` and `smk`, the results are averaged over 10 different programs, to make them more representative since the graph generation is not deterministic, and thus some instances can be easier to query. For all the three programs, the minimal set of probabilistic facts is empty, so the PASOCS and PASTA work on the same set of worlds. Inference times are shown in Figure 1a. PASOCS on `mky` returned an internal error of the solver while parsing the program. In a second experiment, `bird`, we modify the `brd` program by removing `0.1::fly(1)`, and we ask two queries: `fly(1)`, and `fly(1)` given that `fly(2)` has been observed. For these two experiments, the minimal set of probabilistic facts contains `bird(1)`. Results are shown in Figure 1b. Overall, with our solution we can manage a larger number of probabilistic facts. Moreover, the introduction of the minimal set of probabilistic facts gives a substantial improvement, as shown in Figure 1b. However, both PASOCS and PASTA rely on the generation of all the worlds, which increase in an exponential way.

6 Related Work

There are several PASP systems such as P-log [4], LPMLN [16], PrASP [17], and SMProbLog [22]: these aim at finding sharp probability values. We compare PASTA only with PASOCS [23] since, to the best of our knowledge, it is the only system that performs inference on probabilistic answer set programs with aggregates under the credal semantics. The solver proposed in [9] allows counting the answer sets of a given program, so, in principle, may be applied to perform inference in PASTA programs, however, aggregates are not allowed. The solution

proposed in [2] adopts ASP techniques to perform inference in probabilistic logic programs but it is still focused on the computation of a sharp probability value. Statistical statements are considered also by [14] where a semantics is given by resorting to cross entropy minimization. Similarly to the case of [25], we differ because we do not select a specific model but we consider all the models consistent with the statements and we compare lower and upper bounds.

7 Conclusions

In this paper, we considered probabilistic conditionals as statistical statements - "Type 1" statements according to Halpern's definition - and interpreted them under the credal semantics of probabilistic answer set programs. Our approach, called PASTA, includes both a language and an inference algorithm: the language is given a semantics by converting a probabilistic conditional into three ASP rules, one corresponding to the possible combinations of facts, and two constraints, one for the lower and one for the upper bound; the algorithm computes lower and upper probability values for conditional queries. On various programs, PASTA is able to handle a larger number of probabilistic facts than the state of the art solver for probabilistic answer set programs under the credal semantics. As future work, we plan to introduce abductive reasoning in this framework [3].

Acknowledgements This research was partly supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No. 952215. Damiano Azzolini was supported by IndAM - GNCS Project with code CUP_E55F22000270001.

References

1. Alviano, M., Faber, W.: Aggregates in answer set programming. KI-Künstliche Intelligenz 32(2), 119–124 (2018)
2. Aziz, R.A., Chu, G., Muise, C.J., Stuckey, P.J.: Stable model counting and its application in probabilistic logic programming. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. pp. 3468–3474. AAAI Press (2015)
3. Azzolini, D., Bellodi, E., Ferilli, S., Riguzzi, F., Zese, R.: Abduction with probabilistic logic programming under the distribution semantics. International Journal of Approximate Reasoning 142, 41–63 (2022)
4. Baral, C., Gelfond, M., Rushton, N.: Probabilistic reasoning with answer sets. Theor. Pract. Log. Prog. 9(1), 57–144 (2009)
5. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)
6. Cozman, F.G., Mauá, D.D.: On the semantics and complexity of probabilistic logic programs. J. Artif. Intell. Res. 60, 221–262 (2017)
7. Cozman, F.G., Mauá, D.D.: The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. Int. J. Approx. Reason. 125, 218–239 (2020)

8. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Veloso, M.M. (ed.) IJCAI 2007. vol. 7, pp. 2462–2467. AAAI Press/IJCAI (2007)
9. Eiter, T., Hecher, M., Kiesel, R.: Treewidth-aware cycle breaking for algebraic answer set counting. In: Bienvenu, M., Lakemeyer, G., Erdem, E. (eds.) Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021. pp. 269–279 (2021)
10. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: European Workshop on Logics in Artificial Intelligence. pp. 200–212. Springer (2004)
11. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot asp solving with clingo. Theory and Practice of Logic Programming 19(1), 27–82 (2019)
12. Gebser, M., Kaufmann, B., Schaub, T.: Solution enumeration for projected boolean search problems. In: van Hoeve, W.J., Hooker, J. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 71–86. Springer-Verlag (2009)
13. Halpern, J.Y.: An analysis of first-order logics of probability. Artif. Intell. 46(3), 311–350 (1990)
14. Jaeger, M.: Probabilistic reasoning in terminological logics. In: Doyle, J., Sandewall, E., Torasso, P. (eds.) 4th International Conference on Principles of Knowledge Representation and Reasoning. pp. 305–316. Morgan Kaufmann (1994)
15. Kern-Isberner, G., Thimm, M.: Novel semantical approaches to relational probabilistic conditionals. In: Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning. pp. 382–392. AAAI Press (2010)
16. Lee, J., Wang, Y.: A probabilistic extension of the stable model semantics. In: AAAI Spring Symposia (2015)
17. Nickles, M.: A tool for probabilistic reasoning based on logic programming and first-order theories under stable model semantics. In: Michael, L., Kakas, A. (eds.) Logics in Artificial Intelligence. pp. 369–384. Springer International Publishing, Cham (2016)
18. Paris, J.B.: The Uncertain Reasoner's Companion: A Mathematical Perspective. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (1995)
19. Riguzzi, F.: Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning. River Publishers, Gistrup, Denmark (2018)
20. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) ICLP 1995. pp. 715–729. MIT Press (1995)
21. Thimm, M., Kern-Isberner, G.: On probabilistic inference in relational conditional logics. Logic Journal of the IGPL 20(5), 872–908 (03 2012)
22. Totis, P., Kimmig, A., Raedt, L.D.: Smproblog: Stable model semantics in problog and its applications in argumentation. ArXiv abs/2110.01990 (2021)
23. Tuckey, D., Russo, A., Broda, K.: Pasocs: A parallel approximate solver for probabilistic logic programs under the credal semantics. ArXiv abs/2105.10908 (2021)
24. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
25. Wilhelm, M., Kern-Isberner, G., Finthammer, M., Beierle, C.: Integrating typed model counting into first-order maximum entropy computations and the connection to markov logic networks. In: Barták, R., Brawner, K.W. (eds.) Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference. pp. 494–499. AAAI Press (2019)