SMProbLog: Stable Model Semantics in ProbLog and its Applications in Argumentation

Pietro Totis, Angelika Kimmig, Luc De Raedt

KU Leuven, Dept. of Computer Science; Leuven.AI, B-3000 Leuven, Belgium
firstname.lastname@cs.kuleuven.be

Abstract

We introduce SMProbLog, a generalization of the probabilistic logic programming language ProbLog. A ProbLog program defines a distribution over logic programs by specifying for each clause the probability that it belongs to a randomly sampled program, and these probabilities are mutually independent. The semantics of ProbLog is given by the success probability of a query, which corresponds to the probability that the query succeeds in a randomly sampled program. It is well-defined when each random sample uniquely determines the truth values of all logical atoms. Argumentation problems, however, represent an interesting practical application where this is not always the case. SMProbLog generalizes the semantics of ProbLog to the setting where multiple truth assignments are possible for a randomly sampled program, and implements the corresponding algorithms for both inference and learning tasks. We then show how this novel framework can be used to reason about probabilistic argumentation problems. Therefore, the key contribution of this paper are: a more general semantics for ProbLog programs, its implementation into a probabilistic programming framework for both inference and parameter learning, and a novel approach to probabilistic argumentation problems based on such framework.

1 Introduction

Designing systems that are able to argue and persuade is a very relevant and challenging task in the development of artificial intelligence. The goal of abstract argumentation (Dung 1995) is to compute the set of acceptable arguments from a framework describing an argumentation process. In real-world scenarios the information available is often incomplete or questionable, therefore the ability to take into account uncertainty is fundamental in such situations. Consider for instance the following example of argumentative microtext adapted from (Stede et al. 2016):

Example 1 *Yes, it's annoying and cumbersome to separate your rubbish properly all the time (a_1), but small gestures become natural by daily repetition (a_2). Three different bin bags stink away in the kitchen and have to be sorted into different wheelie bins (a_3). But still Germany produces way too much rubbish (a_4) and too many resources are lost when what actually should be separated and recycled is burnt (a_5). We Berliners should take the chance and become pioneers in waste separation! (a_6)*

In the epistemic approach (Hunter 2013) a probabilistic argument graph describes degrees of belief associated to arguments a_i, e.g. "a_1 is believed with probability 0.7" and their relations, for example a_1 attacks a_2, a_3 supports a_1...We want to reason over these elements in order to answer queries like "What is the likelihood of accepting argument a_6?". This is a typical probabilistic logic programming (PLP) task: PLP frameworks and languages are designed to provide powerful general-purpose tools for modeling of and reasoning about structured, uncertain domains, e.g. PRISM (Sato and Kameya 2008), ICL (Poole 2008), ProbLog (De Raedt, Kimmig, and Toivonen 2007) or LPAD/CP-logic (Vennekens, Denecker, and Bruynooghe 2009). When considering these two domains, the question is natural: can PLP effectively model argumentation processes and reason over its intrinsic uncertainty?

This question has been only partially investigated so far: of the two main interpretations of probabilities in argument graphs (Hunter 2013), namely the *constellations* approach and the *epistemic* approach, only the former has been studied in the context of PLP (Mantadelis and Bistarelli 2020). In fact, probabilistic argumentation systems propose different combinations of argumentation frameworks, probability interpretations and reasoning systems, tailored to manipulating probabilities in argumentation. As we summarize in Table 1, some do not follow the epistemic approach (Li, Oren, and Norman 2011; Kido and Okamoto 2017; Mantadelis and Bistarelli 2020), others do not consider PLP or a Bayesian view (i.e. causal influence between random variables defining a single probability distribution) of probabilities (Hunter, Polberg, and Thimm 2020). This paper fills this gap and shows how PLP can be used to reason about an epistemic argumentation graph. Approaching argumentation from PLP raises also the question: can we answer queries of the kind "How does my belief in a_1 change if a_5 is accepted?" or perform typical Bayesian tasks as learning the beliefs of an agent given a set of observations of accepted arguments?

We propose a new PLP framework where this is possible, since we argue that existing systems are limited in what problems they can model and reason about. In fact, traditional PLP frameworks do not consider as valid inputs the programs modelling cyclic relations involving negations. This is a pattern often found in argument graphs, where reciprocal attacks are common. For example, we will encode

Framework	Epistemic	PLP	Bayesian	EM Learning	Implementation
(Li, Oren, and Norman 2011)	✗	✗	✗	✗	✓
(Kido and Okamoto 2017)	✗	✗	✓	✗	✗
(Mantadelis and Bistarelli 2020)	✗	✓	✓	✗	✓
(Hunter, Polberg, and Thimm 2020)	✓	✗	✗	✗	✗
SMProbLog	✓	✓	✓	✓	✓

Table 1: Argumentation frameworks comparison.

this pattern, i.e. accepting a_1 inhibits my belief in a_2 and vice versa, with the logic rules $\neg a_1 \leftarrow a_2$. $\neg a_2 \leftarrow a_1$. A model containing such rules would not be a valid input for traditional PLP frameworks. For this reason, in sections 3 and 4 we propose a PLP system, SMProbLog, based on a new semantics, where it is possible to reason over such models. In Section 5 we show how this framework can model and reason over probabilistic argumentation problems, and how to apply typical PLP techniques in this setting.

The key contributions of the paper are: 1) we define a novel semantics for PLP based on stable model semantics, 2) we develop an implementation of a PLP framework, SMProbLog, derived from ProbLog2 (Dries et al. 2015) where inference and learning tasks can be performed under the new semantics, 3) we show an application of SMProbLog to encode and solve probabilistic argumentation problems in a novel reasoning framework based on a Bayesian interpretation and manipulation of probabilities.

2 Related work

In this paper we present an extension of the ProbLog system and semantics (De Raedt, Kimmig, and Toivonen 2007). ProbLog is a probabilistic language extending Prolog, where facts and clauses are annotated with (mutually independent) probabilities. Probabilistic logic programs based on distribution semantics can be viewed as a "programming language" generalization of Bayesian Networks (Sato 1995). Many PLP frameworks, e.g. ProbLog, ICL, PRISM and CP-Logic, do not support general *normal* logic programs (cfr. Section 3). For instance ICL uses acyclic logic programs, while ProbLog's and CP-Logic's valid inputs are programs where each possible world corresponds to a two-valued unique well-founded model (Dries et al. 2015). When general normal logic programs are concerned, well-founded semantics (Gelder, Ross, and Schlipf 1991) is a three-valued semantics, that is, logical atoms can be true, false or undefined. On the other hand, stable model semantics (Gelfond and Lifschitz 1988) defines two-valued models, but a logic program can have more than one stable model. Both semantics have been considered to extend traditional PLP framework's semantics: Hadjichristodoulou and Warren (2012) presents an extension of PRISM based on three-valued well-founded models. On the other hand, the application of stable model semantics in PLP is represented by probabilistic answer set programming (ASP) (Cozman and Mauá 2017, 2020). With SMProbLog, we consider stable model semantics in the context of the distribution semantics, that is, we extend a probability distribution over the probabilistic choices in the program to a probability distribution over stable models of the full program, whereas probabilistic ASP directly defines the latter; cf. also Example 4.

In this paper we will focus on computing exact probabilities as in Baral, Gelfond, and Rushton (2009) and Lee and Wang (2016) rather than defining an interval of probability for each atom as in *credal semantics* (Lukasiewicz 1998).

We will consider the application of this semantics to probabilistic argumentation problems. This is a novel approach as previous work considers different semantics and reasoning techniques for probabilistic argument graphs. An *argument graph* (Dung 1995) is a pair (A, R) where A is a set of arguments and $R \subseteq A \times A$ is a binary (attack) relation over A. A *probabilistic argument graph* (Li, Oren, and Norman 2011) is a tuple (A, R, P_A, P_R) where: (A, R) is an argument graph and P_A and P_R are functions: $P_A : A \rightarrow [0, 1]$ and $P_R : A \times A \rightarrow [0, 1]$. The *constellations* approach (Hunter 2013) interprets the probabilities as uncertainty over the structure of the graph and models a probability distribution over subgraphs. The admissible arguments are defined by the classical extension-based semantics (Dung 1995). Mantadelis and Bistarelli (2020) presents an encoding of a constellations approach in ProbLog. The *epistemic* approach interprets the probabilities as a direct measure of an agent's belief in the arguments. Previous work focused on reasoning about the properties of families of probability distributions that are consistent with the argument graph and additional constraints (*epistemic graphs*) (Hunter, Polberg, and Thimm 2020). Our approach combines both views into a novel reasoning framework. From the epistemic approach we adopt the view of probabilities as a measure of the degree of belief of an agent, but consider a single probability distribution in a Bayesian style rather than families of distributions. Such distribution, as in the constellation approach, is determined by the probability that an argument is true (accepted) in a possible world, defined by the distribution semantics. However, in order to determine whether an argument is true or not, we rely on the stable model semantics for logic programs rather than the classical argumentation semantics. Finally, Kido and Okamoto (2017) applies Bayesian reasoning techniques for statistically estimating the existence of an attack relation between arguments, but does not manipulate degrees of belief directly connected with acceptability.

3 Probabilistic Semantics

A *normal* logic program $\mathcal{L}$ is a set of rules of the form $h \leftarrow b_1, \ldots, b_n$. We use the standard terminology about atoms, terms, predicates and literals. The *head* h is an atom and the *body* $b_1, \ldots, b_n$ is a logical conjunction of liter-

als. We denote with $\neg a$ the logical negation of an atom a, and with $\sim a$ the negation as failure of a. Rules with empty body are called facts. We model uncertainty in logic programs by annotating facts with probabilities: the probabilities of facts are mutually independent and each *probabilistic fact*, i.e. $p :: f.$, corresponds to an atomic choice, that is, a choice between including f in the program (with probability p) or discarding it $(1 - p)$. Probabilistic logic programs are queried for the likelihood of atoms, which is commonly defined by the *distribution semantics* (Sato 1995). A *total choice* is a combination of atomic choices over all probabilistic facts, that is, a subset of the set of all facts f. We denote the set of all total choices of $\mathcal{L}$ with $\Omega(\mathcal{L})$. The probabilities of facts define a probability distribution over all (2^n with n probabilistic facts) total choices defining the possible non-probabilistic *subprograms*, also called *possible worlds*, obtained from $\mathcal{L}$ by including or discarding a fact according to the corresponding atomic choice. The probability of success of a query is the sum of the probability of each possible world where the query is true.

As the distribution semantics relies on a one-to-one mapping between total choices and models, many PLP frameworks, e.g. ProbLog, PRISM and CP-Logic, restrict the class of valid inputs to programs where each total choice corresponds to a two-valued well founded model (Gelder, Ross, and Schlipf 1991). As we will show, this limitation prevents the encoding of a variety of causal relations as far as argumentation is concerned. For this reason we focus on more general semantics to define models and probabilities based on total choices. We consider *stable model semantics* (Gelfond and Lifschitz 1988): given a normal logic program $\mathcal{L}$ and a set S of atoms interpreted as $true$, called *candidate model*, stable model semantics is defined in terms of the *reduct* of a program $\mathcal{L}$ w.r.t. S. The *reduct* of P w.r.t. S, $\mathcal{L}^S$, is defined as follows: 1) for all $r \in \mathcal{L}$ remove r from $\mathcal{L}$ if $a \in S$ is negated in the body of r; 2) for all $r \in \mathcal{L}$ remove from the body of r all $\neg a, a \notin S$. If S is a minimal model for $\mathcal{L}^S$ then S is a *stable model* (*answer set*) for $\mathcal{L}$.

Using stable models allows us to relax the standard PLP assumption of exactly one model per total choice.

Definition 1 *A valid* SM*ProbLog program is a probabilistic normal logic program where each total choice leads to* at least *one (two-valued) stable model.*

We note that this definition excludes the case where a total choice has *no* stable model, i.e., results in an inconsistent program, and thus a loss of probability mass. Our focus here is on handling non-probabilistic choices introduced by the logic component, as illustrated in the following example.

Example 2 *Consider the following normal logic program:* $0.5 :: a.\ \ 0.5 :: b.\ \ c \leftarrow a.\ \ d \leftarrow b.\ \ c \leftarrow \sim d.\ \ d \leftarrow \sim c.$ *There are four possible worlds:* $\omega_1 = \{a, b\}$, $\omega_2 = \{a\}$, $\omega_3 = \{b\}$, $\omega_4 = \{\}$, *with* $P(\omega_i) = 0.25$. *While the first three correspond to one stable model, i.e.* $M(\omega_1) = \{\{a, b, c, d\}\}$, $M(\omega_2) = \{\{a, c\}\}$, $M(\omega_3) = \{\{b, d\}\}$, ω_4 *has two stable models:* $M(\omega_4) = \{\{c\}, \{d\}\}$.

We follow the principle of maximum entropy and choose to uniformly distribute the probability of a total choice across all the corresponding stable models. Consider for instance Example 2: while the probabilistic choices characterize the four possible worlds, the logic encoded in the rules prescribes in ω_4 a choice between c and d. Since this choice is not related to beliefs but rather logical consistency, we assume that all stable models are equally probable for a given total choice. More formally, each total choice ω corresponds to: (1) a probability $P(\omega)$ as defined by distributions semantics and (2) a set $M(\omega)$ of stable models:

Definition 2 *Given a probabilistic normal logic program $\mathcal{L}$, the probability $P(\omega)$ of a total choice ω is:*

$$P(\omega) = \prod_{(f:p)\in\mathcal{L}, f\in\omega} p \cdot \prod_{(f:p)\in\mathcal{L}, f\notin\omega} 1 - p$$

that is, the product of the probabilities of the facts for being included/excluded from ω.

Definition 3 *Given a probabilistic normal logic program $\mathcal{L}$ and a corresponding total choice ω, $M(\omega)$ is the set of stable models of the subprogram induced by ω on $\mathcal{L}$.*

The probability of a stable model $S_\omega \in M(\omega)$ is the probability of the corresponding total choice ω normalized w.r.t. the number of the stable models for that possible world:

Definition 4 *Given a probabilistic normal logic program $\mathcal{L}$ and a total choice ω, $\forall S_\omega \in M(\omega)$: $\hat{P}(S_\omega) = \frac{P(\omega)}{|M(\omega)|}$*

The probability of an atom a is the sum of the probabilities of the models where a is true:

Definition 5 *Given a probabilistic normal logic program $\mathcal{L}$, the probability $\mathbb{P}(a)$ of success of querying a in $\mathcal{L}$ is:*

$$\mathbb{P}(a) = \sum_{a\in S_\omega, S_\omega\in M(\omega), \omega\in\Omega(\mathcal{L})} \hat{P}(S_\omega)$$

Our approach generalizes traditional PLP frameworks: when a program defines total choices corresponding to exactly one two-valued well-founded model, then the semantics agree on both models and probability. As for the model, if a normal logic program has a total two-valued well-founded model, then the model is the unique stable model (Gelder, Ross, and Schlipf 1991). Thus, in this case $|M(\omega)| = 1$ for each total choice ω, from which it follows that $\hat{P}(S_\omega) = P(\omega)$ for the single $S_\omega \in M(\omega)$ ($M(\omega) = \{S_\omega\}$). This means that the probability of the model is the probability of the corresponding total choice as in (probabilistic) two-valued well founded semantics.

Example 3 *We complete Example 2: models $\{a, b, c, d\}$, $\{a, c\}$, $\{b, d\}$ are the unique model for, respectively, $\omega_1, \omega_2, \omega_3$ hence $\hat{P}(\{a, b, c, d\}) = \hat{P}(\{a, c\}) = \hat{P}(\{b, d\}) = P(\omega_1) = P(\omega_2) = P(\omega_3) = 0.25$. The likelihood of ω_4 is uniformly distributed over the two stable models $\{c\}$ and $\{d\}$, whose probability is thus $\frac{P(\omega_4)}{|M(\omega_4)|} = \frac{0.25}{2} = 0.125$. Therefore $\mathbb{P}(a) = \mathbb{P}(b) = 0.5$, $\mathbb{P}(c) = \mathbb{P}(d) = 0.625$.*

The key difference between SMProbLog and probabilistic ASP frameworks is that our approach is based on the distribution semantics, that is, we consistently extend a probability distribution over total choices to a probability distribution over models of the logic program, whereas probabilistic

ASP languages such as P-Log (Baral, Gelfond, and Rushton 2009) and LP$^{\text{MLN}}$ Lee and Wang (2016) use the probability labels to directly define a (globally normalized) probability distribution over the models of a (derived) program. Consider the following example:

Example 4 *In Example 2, if we associate to each stable model S_ω the corresponding $P(\omega)$, the sum of the probabilities of the stable models is $5 \cdot 0.25 = 1.25$ and a normalization w.r.t. all answer sets leads to a probability $\hat{P}'(S_\omega) = 0.25/1.25 = 0.2$ for all S_ω. Then $\mathbb{P}(a) = \hat{P}'(S_{\omega_1}) + \hat{P}'(S_{\omega_2}) = 0.4$. This means that the marginal probability of a in the joint distribution is lower than the prior 0.5 despite no epistemic influence is defined on a. In our approach, the marginal probability of that fact under the final distribution equals the fact's label, and is thus directly interpretable.*

4 Inference and learning

In this section we present the algorithms implemented in SMProbLog to perform inference and learning tasks. We reduce the probabilistic inference task to a *weighted model counting problem* (WMC) (Cadoli and Donini 1997). The task of model counting is #P-complete in general, therefore we follow the approach of Fierens et al. (2011) of transforming the logic program into a representation where this task becomes tractable. In this setting, the original grounding procedure is no longer applicable, since it uses the presence of cycles through negation as a sufficient condition to reject programs for which the two-valued well-founded models semantics is not well-defined. We describe the novel normalization algorithm and show that under this semantics ProbLog2's expectation-maximization algorithm (EM learning) for Bayesian learning over a set of evidence (Gutmann, Thon, and De Raedt 2011) is correct.

Inference

The inference task is defined as follows:

Definition 6 *The Inference task:* ***Given***

- *A ProbLog program $\mathcal{L}$: let G be the set of all ground (probabilistic and derived) atoms of $\mathcal{L}$.*
- *A set $E \subseteq G$ of observed atoms (evidence), along with a vector e of corresponding observed truth values ($E = e$).*
- *A set $Q \subseteq G$ of atoms of interest (queries).*

Find *the marginal distribution of every query atom given the evidence, i.e. computing $P(q \mid E = e)$ for each $q \in Q$.*

Compilation. In SMProbLog the reduction to WMC is based on the pipeline described in Fierens et al. (2011), with the adaptations proposed in Aziz et al. (2015), where the CNF conversion of $\mathcal{L}$ is replaced with an encoding of the rules, variables and the corresponding strongly connected components as the input for the knowledge compiler. $\mathcal{L}$ is in fact transformed by means of knowledge compilation into a *smooth d-DNNF* formula (deterministic Decomposable Negation Normal Form) (Darwiche 2004).

Definition 7 *A* NNF *is a rooted directed acyclic graph in which each leaf node is labeled with a literal and each internal node is labeled with a disjunction or conjunction. A smooth* d-DNNF *is an NNF with the following properties:*

- *Deterministic: for all disjunctive nodes the children represent formulas pairwise inconsistent.*
- *Decomposable: the subtrees rooted in two children of a conjunction node do not have atoms in common.*
- *Smooth: all children of a disjunction node use the same set of atoms.*

On d-DNNFs the task of model counting becomes tractable. The d-DNNF is further transformed into an equivalent arithmetic circuit, by replacing conjunctions and disjunctions respectively with multiplication and summation nodes, and by replacing leaves with the weight of the corresponding literals. The arithmetic circuit thus allows us to efficiently perform the task of WMC. In this pipeline, we replace the grounding step with a bottom-up grounding procedure to handle normal logic programs, and use the absence of cycles through negations as a condition to reduce to ProbLog's evaluation algorithm, where SMProbLog's semantics is equivalent as discussed in Section 3. If this is not the case, we evaluate the output of the pipeline as follows.

Enumeration. The new semantics requires us to know for each model the corresponding normalization constant. We derive from the d-DNNF an equivalent circuit where each leaf for a literal l is replaced by a list of (partial) models $[\{l\}]$, disjunctions are replaced by the union of the children and conjunctions correspond to the cartesian product of the children. Traversing bottom-up such circuit returns the list of models, from which we build a map from total choices to the corresponding (number of) models. In fact, for each model the truth value of the literals corresponding to probabilistic facts defines the corresponding total choice. This is a computationally expensive step since it entails enumerating all possible models.

Evaluation. To evaluate $P(q \mid E = e)$, the weights of the leaves in the arithmetic circuit are instantiated according to the query and the given evidence. The weight of the root is then the sum of the weights of the models where q is true and evidence is satisfied. Since WMC can be performed efficiently on the circuit, we compute first the unnormalized weight of the root w^*, and then correct it. We know that for each total choice ω s.t. $n = |M(\omega)| > 1$ and $\omega \models E = e$, we are overcounting the weight of its models by $w_{M(\omega)} = \frac{P(\omega)\cdot(n-1)}{n}$, hence we remove $w_{M(\omega)}$ from w^* for each $S_\omega \in M(\omega)$ such that $S_\omega \models q \wedge E = e$ (Algorithm 2). This correction increases the complexity of the evaluation step, as we need to iterate over the total choices with multiple models to retrieve and apply the different normalization constants. Normalizing weights while traversing the circuit is possible but less efficient. In fact, in the d-DNNF different total choices can correspond to the same node, e.g. when conjoining a literal and a disjunction node, therefore the weight of such node has to be normalized with (potentially) different constants for its children. At the same time normalization cannot be applied on the children level,

because they might be a subtree for different total choices. Therefore, their weights cannot be aggregated in a single value until a total choice is defined, which is a similar process to the enumeration step on the d-DNNF. In Section 6 we show that exploiting the efficiency of the (unnormalized) WMC task results in the evaluation step being significantly faster than the enumeration step.

Algorithm 2: Evaluation step schema

Output: $P(q \mid E = e)$
$w^* \leftarrow unnormalized(root(\mathcal{C}))$
for ω **in** $\Omega(\mathcal{L})$ **s.t.** $|M(\omega)| > 1, \omega \models E = e$ **do**
  for $S_\omega \in M(\omega)$ **s.t.** $S_\omega \models q \wedge E = e$ **do**
    $w^* = w^* - \frac{P(\omega)\cdot(n-1)}{n}$
return w^*

Learning by Expectation Maximization

In the learning from interpretation setting, given a ProbLog program $\mathcal{L}$ where some probabilities are unknown (parameters), and a set of interpretations for $\mathcal{L}$, the goal is to estimate the value of the parameters such that the predicted values maximize the likelihood of the given interpretations:

Definition 8 *Max-Likelihood Parameter Estimation:* ***Given***

- *A ProbLog program $\mathcal{L}(\mathbf{p}) = \mathcal{F} \cup \mathcal{BK}$ where $\mathcal{F}$ contains probabilistic facts and $\mathcal{BK}$ contains background knowledge. $\mathbf{p} = \langle p_1, ..., p_N \rangle$ is a set of unknown parameters attached to probabilistic facts.*
- *A set $\mathcal{I}$ of (partial) interpretations $\{I_1, ..., I_M\}$ as training examples.*

Find *the maximum likelihood probabilities $\widehat{\mathbf{p}} = \langle \widehat{p_1}, ..., \widehat{p_N} \rangle$ for all interpretations in $\mathcal{I}$. Formally,*

$$\widehat{\mathbf{p}} = \arg\max_{\mathbf{p}} P(\mathcal{I}|\mathcal{L}(\mathbf{p})) = \arg\max_{\mathbf{p}} \prod_{m=1}^{M} P_s(I_m|\mathcal{L}(\mathbf{p}))$$

Learning from interpretations of parameters in a ProbLog program is implemented in a likelihood maximization setting such that the parameters are iteratively updated.

In total observability each interpretation $I_j \in \mathcal{I}$ observes the truth value of each atom and probabilistic fact of $\mathcal{L}$. This case reduces to counting the number of true occurrences of each of probabilistic fact in the interpretations $\mathcal{I}$ (Gutmann, Thon, and De Raedt 2011). This is correct for SMProbLog because the observation of a probabilistic fact is independent of the rest of the program for each interpretation.

In the partially observable case, where each interpretation I_j observes the truth value of a subset of $\mathcal{L}$, the parameter update iteration relies on probabilistic inference to update the likelihood of a fact given an interpretation I_m. At each iteration k the parameters from the previous iteration $k-1$ are used in $\mathcal{L}$ to compute the conditional expectation of the parameter given the interpretations, until convergence. This means that the normalization w.r.t. multiple stable models is incorporated in the conditional probabilities computed by the inference task, therefore also the parameter estimate is normalized w.r.t. the different stable models that may correspond to a given partial observation.

5 Modelling Probabilistic Argumentation

In this section we show how PLP modelling techniques can be applied to encode causal relations between beliefs in arguments. We show how causal relations between arguments often lead to models that require the generalized semantics presented in the previous section. The flexibility of PLP not only captures basic probabilistic argument graph, but also many other features from the argumentation literature:

- Support relations.
- Quantitative evaluations of attacks/supports.
- Attack (and supports) from sets of arguments.
- Distinctions between proponents of arguments.

which we also combine with features from PLP like first-order knowledge and querying conditional probabilities. Finally, we show an intuitive method to model belief in arguments and their relations, by means of PLP and ProbLog's syntactical features, to provide an implementation of our approach. Let us consider Example 1 to describe our modelling approach. We model arguments (a_i), attacks (relation R^-) $\{(a_1, a_6), (a_4, a_1)\}$,$(a_1, a_2), (a_2, a_1)\}$, supports (relation R^+) $\{(a_5, a_4), (a_3, a_1)\}$ and distinguish between arguments coming from a proponent ($\{a_2, a_4, a_5, a_6\}$) and those introduced by an opponent ($\{a_1, a_3\}$). We also define the following probability functions to derive an example of a probabilistic argumentation problem: $P_A(a_1) = 0.4$, $P_A(a_2) = 0.8$, $P_A(a_3) = 0.3$, $P_A(a_4) = 0.7$, $P_A(a_5) = 0.6$, $P_A(a_6) = 0.7$, $P_{R^-}(a_1, a_6) = 0.6$, $P_{R^-}(a_2, a_1) = 0.8$, $P_{R^-}(a_1, a_2) = 0.7$, $P_{R^-}(a_4, a_1) = 0.3$, $P_{R^+}(a_5, a_4) = 0.6$, $P_{R^+}(a_3, a_1) = 0.5$. Note that we already consider probabilistic argument graphs where a (probabilistic) support relation is also included.

We define the semantics of the probabilities in an argument graph by mapping each element of the graph to a fact, rule or probability in a PLP model. We model a probabilistic argumentation graph (with supports) by mapping:

1. $P_A(a) = p$ to the fact $p :: base\_arg(a)$ and the rule $arg(a) \leftarrow base\_arg(a)$.
2. attacks $P_{R^-}(a, b) = p$ to rules $p :: \neg arg(b) \leftarrow arg(a)$.
3. supports $P_{R^+}(a, b) = p$ to rules $p :: arg(b) \leftarrow arg(a)$.

Arguments thus map to predicates of the form $arg(a)$, which we will query for their truth value. The semantics of P_A is given by the probabilistic facts of the form $p :: base\_arg(a)$: this is a prior belief from which the truth of the corresponding argument can be inferred. It is independent of the other epistemic information, therefore represents an unconditioned bias towards $arg(a)$. The edges of an argument graph model the influence between the arguments: we encode them as *causal relations* between the acceptance of two arguments. That is, a support $arg(b) \leftarrow arg(a)$. reads as "accepting a causes believing in (accepting) b". Viceversa, $\neg arg(b) \leftarrow arg(a)$. reads as "accepting a causes not believing in (rejecting) b". The semantics of P_{R^+} and P_{R^-} is given by the mapping of the probability of the relation as the probability of the rule. For example a probability $P_{R^+}(a_5, a_4) = 0.6$ is mapped to $0.6 :: arg(a_4) \leftarrow arg(a_5)$, which describes how believing in a_5 causes a (fine-grained)

increase in the belief in a_4. Similarly, attacks are encoded as supports for counterarguments, resulting in an inhibition of the belief in the original argument, e.g. $0.3 :: \neg arg(a_1) \leftarrow arg(a_4)$. This mapping relies on three features of PLP (and ProbLog): first, rule heads can be annotated as syntactic sugar: $p :: h \leftarrow b_1, \ldots, b_n.$ is equivalent to a new fact $p :: f.$ plus $h \leftarrow f, b_1, \ldots, b_n$. This gives a quantitative evaluation of how likely believing in the body causes believing the head as well. Second, attacks define an *inhibition effect* (Meert and Vennekens 2014) on a belief by means of a language feature for probabilistic logic frameworks called *negation in the head* (Vennekens 2013). Negation in the head gives an interpretation of the negation of heads from Answer Set Programming in the context of epistemic reasoning and logic theories defining causal mechanisms. We describe such interpretation by means of ProbLog's syntax: each rule $\neg h \leftarrow b_1, \ldots, b_n.$ is interpreted by replacing all heads h in the program with a new atom h_{pos} and all heads $\neg h$ with a new atom h_{neg}, and the rule $h \leftarrow h_{pos}, {\sim}h_{neg}.$ is added. This last rule thus defines that h can be inferred from any of the causes of h (making h_{pos} true) only if there is no cause for believing in h_{neg}, the opposite of h. Finally, the probability of an atom which appears as the head of multiple rules is computed by means of the typical *noisy-or* effect (Meert and Vennekens 2014).

Example 5 *We show the rewriting for* $P_A(a_1) = 0.4$*,* $P_A(a_2) = 0.8$ *and* $P_{R^-}(a_1, a_2) = 0.7$*:*

$0.4 :: base\_arg(a_1).$ $0.8 :: base\_arg(a_2).$ $0.7 :: f.$
$arg(a_6) \leftarrow base\_arg(a_2).$ $arg(a_1) \leftarrow base\_arg(a_1).$
$arg(a_2) \leftarrow arg_{pos}(a_2), {\sim}arg_{neg}(a_2).$
$arg_{neg}(a_2) \leftarrow f, arg(a_1).$

These features combined provide the semantics for an argument graph where the belief of an argument a is defined by its marginal probability in the modelled joint distribution. This belief can be queried in a PLP style, which thus reflects the combined interaction of a bias towards a with the belief in other arguments and their relation with a.

Example 6 *We model our running example as:*

$0.4 :: base\_arg(a_1).$ $0.8 :: base\_arg(a_2).$
$0.3 :: base\_arg(a_3).$ $0.7 :: base\_arg(a_4).$
$0.6 :: base\_arg(a_5).$ $0.7 :: base\_arg(a_6).$
$arg(A) \leftarrow base\_arg(A).$
$0.6 :: \neg arg(a_6) \leftarrow arg(a_1).$ $0.6 :: arg(a_4) \leftarrow arg(a_5).$
$0.3 :: \neg arg(a_1) \leftarrow arg(a_4).$ $0.5 :: arg(a_1) \leftarrow arg(a_3).$
$0.8 :: \neg arg(a_1) \leftarrow arg(a_2).$ $0.7 :: \neg arg(a_2) \leftarrow arg(a_1).$

Resulting in the distribution (beliefs): $\mathbb{P}(arg(a_1)) = 0.21$*,* $\mathbb{P}(arg(a_2)) = 0.69$*,* $\mathbb{P}(arg(a_3)) = 0.3$*,* $\mathbb{P}(arg(a_4)) = 0.81$*,* $\mathbb{P}(arg(a_5)) = 0.6$*,* $\mathbb{P}(arg(a_6)) = 0.61$*.*

Example 6 shows how arguments attacking each other results in a cyclic relation where negation is involved, similar to the one presented in Example 2. Therefore, such interpretation of an argument graph would not be possible with traditional PLP semantics.

Example 7 *In Example 6 rules* $0.8 :: \neg arg(a_1) \leftarrow arg(a_2).$ *and* $0.7 :: \neg arg(a_2) \leftarrow arg(a_1).$ *correspond to the rewriting of Example 5 plus the following statements:*

$$0.8 :: g. \quad arg_{neg}(a_1) \leftarrow g, arg(a_2).$$
$$arg(a_1) \leftarrow arg_{pos}(a_1), {\sim}arg_{neg}(a_1).$$

The negation in the head thus leads to a cyclic dependency $arg(a_1) \rightarrow^- arg_{neg}(a_1) \rightarrow arg(a_2) \rightarrow^- arg_{neg}(a_2) \rightarrow arg(a_1)$ *which results in possible worlds with multiple stable models (thus invalid for ProbLog's semantics).*

The logic program not only can be queried for the marginal probabilities of the arguments, but also for conditional probabilities by specifying *evidence*, that is, a set of facts for which the truth value is determined or assumed. By means of evidence the joint probability distribution can answer queries of the form $\mathbb{P}(arg(a_i)|arg(a_j))$ which thus answer the question "What is the belief in a_i if a_j is known to hold?". This allows an update of the beliefs in arguments in the light of new information about their acceptability.

Example 8 *By querying the model from Example 6 with the addition of the statement* $evidence(arg(a_1), true).$ *we infer the following conditional probabilities:* $\mathbb{P}(arg(a_1)|arg(a_1)) = 1$*,* $\mathbb{P}(arg(a_2)|arg(a_1)) = 0.12$*,* $\mathbb{P}(arg(a_3)|arg(a_1)) = 0.43$*,* $\mathbb{P}(arg(a_4)|arg(a_1)) = 0.75$*,* $\mathbb{P}(arg(a_5)|arg(a_1)) = 0.58$*,* $\mathbb{P}(arg(a_6)|arg(a_1)) = 0.28$*.*

Our framework includes two of the most relevant extensions of the basic argument graph, namely *bipolar* argumentation (Amgoud, Cayrol, and Lagasquie-Schiex 2004), which introduces the support relation, and *weighted* argumentation (Dunne et al. 2011; Coste-Marquis et al. 2012), which introduces a fine-grained quantitative evaluation of the relations between arguments. Moreover, we can expand our framework further to *argument systems* (Nielsen and Parsons 2006), which generalize the attack between two arguments to an attack relation from a set of arguments towards a single argument (set-attacks). The belief in an argument is thus inhibited only if all the attackers of the set are believed, which we model by conjoining attackers in the body of rules, e.g. $\neg arg(a_1) \leftarrow arg(a_4), arg(a_5)$. Clearly, also in this case we can easily combine different argumentative features with each other, for example gradual set attacks, e.g. $0.6 :: \neg arg(a_1) \leftarrow arg(a_4), arg(a_5).$ or (gradual) set supports, e.g. $0.3 :: arg(a_3) \leftarrow arg(a_4), arg(a_5)$. Thanks to the generality of PLP we can also model less standard scenarios. For example bias towards arguments can be defined in terms of trust in the agent proposing the argument, with rules $0.4 :: prop.$ $0.6 :: opp.$ and:

$$arg(a_1) \leftarrow opp.\ arg(a_2) \leftarrow prop.\ \mathrm{arg}(a_3) \leftarrow opp.$$
$$arg(a_4) \leftarrow prop.\ arg(a_5) \leftarrow prop.\ arg(a_6) \leftarrow prop.$$

Moreover, PLP systems are designed to express first-order knowledge, allowing a compact encoding of high-level reasoning, as we already showed in Example 6 with the rule $arg(A) \leftarrow base\_arg(A)$. For instance, we can model that the more agents propose the same argument, the higher its bias is. Assume we express proponents and trust as $p_1 :: prop(1)., \ldots, p_n :: prop(n).$ and $proposes(i, a)$ denotes that argument a is backed by proponent i, then a rule $base\_arg(A) \leftarrow proposes(P, A), prop(P).$ compactly defines a contribution to the bias of each argument, denoted by each measure of trust p_i, from all its proponents.

6 Experiments

The goal of our experiments is to establish the feasibility of our approach, and to answer the following questions: (Q1) what is the performance impact of the more general semantics on the inference task? (Q2) Given a set of observations of accepted arguments generated from a known ProbLog program, can we learn the degrees of belief of the agent that modelled the original program?

Q1. Inference. We consider a variation of a typical PLP example where a set of people has a certain probability of having asthma or being stressed, and stress leads with some probability to smoking:

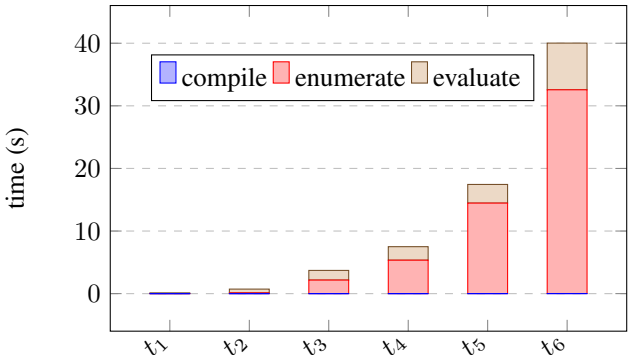
$$0.1 :: asthma(X) \leftarrow person(X).$$
$$0.3 :: stress(X) \leftarrow person(X).$$
$$0.4 :: smokes(X) \leftarrow stress(X).$$

People are related by an influence relation: if a person smokes and influences to some extent another one, then the other person will smoke, and if someone smokes there is a probability to suffer from asthma. If someone suffers from asthma then the person does not smoke (an example of a cycle with negation).

$$smokes(X) \leftarrow influences(Y, X), smokes(Y).$$
$$0.4 :: asthma(X) \leftarrow smokes(X).$$
$$\neg smokes(X) \leftarrow asthma(X).$$

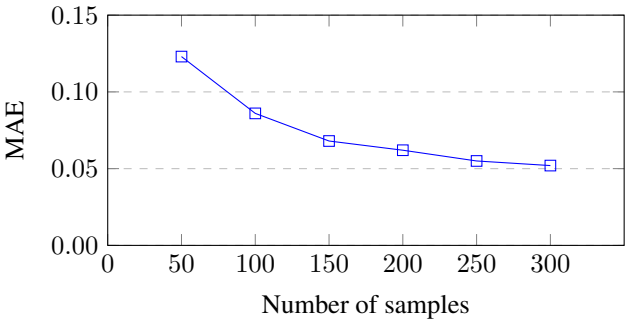
We consider examples with an increasing number of people and relationships: $t_1 = \{person(1).\ person(2).\ 0.3 :: influences(1,2).\ 0.6 :: influences(2,1).\}$, $t_2 = t_1 + \{person(3)\}$, $t_3 = t_2 + \{person(4)\}$, $t_4 = t_3 + \{0.2 :: influences(2,3).\}$, $t_5 = t_4 + \{0.7 :: influences(3,4).\}$, $t_6 = t_5 + \{0.9 :: influences(4,1).\}$. Figure (i) shows the running time of SMProbLog on the different benchmarks, where the enumeration step dominates the running time of SMProbLog, while the evaluation step adds further computational time, but not to the same extent.

Q2. Learning. We answer question 2 by considering a dataset of 283 argument graphs (Stede et al. 2016) and by deriving from each annotation a bipolar argument graph. We attach to arguments and relations random probabilities to reflect an agent's belief in each, as we did in Example 6. The graphs contain from 4 to 11 nodes (average 5.3) with an average of 1.4 attacks and 3 supports per graph, which translates in about 10 learnable parameters on average for each example. We learn such parameters from a set of increasing size of observations of accepted arguments, sampled from the original program, with an upper bound of 100 iterations. Note that we are in the case of partial observability, as the parameters are attached to the predicates $base\_arg$, arg_{pos} and arg_{neg}, but we observe only the outcome in the form of the predicates arg. We evaluate the quality of the learned programs with the mean absolute error (MAE), summarized in Figure (ii). Since the MAE decreases when more examples are given, we can answer positively to our question about learnability of beliefs in argument graphs with SMProbLog.



Benchmark	t_1	t_2	t_3	t_4	t_5	t_6
# prob. facts	10	14	18	19	20	21
# nodes circuit	156	192	228	462	750	1465

(i) Inference time on benchmarks.



(ii) Mean absolute error on learned parameters.

7 Conclusion

Approaching probabilistic argumentation from a probabilistic logic programming perspective stresses the limiting assumptions of PLP frameworks when (probabilistic) normal logic programs are concerned. For this reason in this paper we proposed a new PLP system, SMProbLog, where we implement a combination of the classical distribution semantics for probabilistic logic programs with stable model semantics. SMProbLog generalizes previous work on distributions semantics supporting inference and parameter learning tasks for a wider class of (probabilistic) logic programs.

At the same time, we propose a mapping from probabilistic argument graphs to probabilistic logic programs, which provides a novel semantics for epistemic argument graphs. With our methodology, it is possible to apply traditional Bayesian reasoning techniques to probabilistic argument graphs, such as conditioning over evidence and parameter learning. This results in a modular, expressive, extensible framework reflecting the dynamic nature of beliefs in an argumentation process.

Acknowledgements

This work was supported by the FWO project N. G066818N.

References

Amgoud, L.; Cayrol, C.; and Lagasquie-Schiex, M. 2004. On the bipolarity in argumentation frameworks. In Delgrande, J. P.; and Schaub, T., eds., *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*, 1–9. ISBN 92-990021-0-X.

Aziz, R. A.; Chu, G.; Muise, C. J.; and Stuckey, P. J. 2015. Stable Model Counting and Its Application in Probabilistic Logic Programming. In Bonet, B.; and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 3468–3474. AAAI Press. ISBN 978-1-57735-698-1.

Baral, C.; Gelfond, M.; and Rushton, J. N. 2009. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9(1): 57–144.

Cadoli, M.; and Donini, F. M. 1997. A Survey on Knowledge Compilation. *AI Communications*, 10(3-4): 137–150.

Coste-Marquis, S.; Konieczny, S.; Marquis, P.; and Ouali, M. A. 2012. Weighted Attacks in Argumentation Frameworks. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press. ISBN 978-1-57735-560-1.

Cozman, F. G.; and Mauá, D. D. 2017. On the Semantics and Complexity of Probabilistic Logic Programs. *Journal of Artificial Intelligence Research*, 60: 221–262.

Cozman, F. G.; and Mauá, D. D. 2020. The joy of Probabilistic Answer Set Programming: Semantics, complexity, expressivity, inference. *International Journal of Approximate Reasoning*, 125: 218–239.

Darwiche, A. 2004. New Advances in Compiling CNF into Decomposable Negation Normal Form. In de Mántaras, R. L.; and Saitta, L., eds., *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, 328–332. IOS Press. ISBN 1-58603-452-9.

De Raedt, L.; Frasconi, P.; Kersting, K.; and Muggleton, S., eds. 2008. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-540-78651-1.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2462–2467.

Dries, A.; Kimmig, A.; Meert, W.; Renkens, J.; den Broeck, G. V.; Vlasselaer, J.; and De Raedt, L. 2015. ProbLog2: Probabilistic Logic Programming. In Bifet, A.; May, M.; Zadrozny, B.; Gavaldà, R.; Pedreschi, D.; Bonchi, F.; Cardoso, J. S.; and Spiliopoulou, M., eds., *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III*, volume 9286 of *Lecture Notes in Computer Science*, 312–315. Springer. ISBN 978-3-319-23460-1.

Dung, P. M. 1995. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games. *Artificial Intelligence*, 77(2): 321–358.

Dunne, P. E.; Hunter, A.; McBurney, P.; Parsons, S.; and Wooldridge, M. J. 2011. Weighted argument systems: Basic definitions, algorithms, and complexity results. *Artificial Intelligence*, 175(2): 457–486.

Fierens, D.; den Broeck, G. V.; Thon, I.; Gutmann, B.; and De Raedt, L. 2011. Inference in Probabilistic Logic Programs using Weighted CNF's. In Cozman, F. G.; and Pfeffer, A., eds., *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, 211–220. AUAI Press. ISBN 978-0-9749039-7-2.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. S. 1991. The Well-Founded Semantics for General Logic Programs. *J. ACM*, 38(3): 620–650.

Gelfond, M.; and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In Kowalski, R. A.; and Bowen, K. A., eds., *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, 1070–1080. MIT Press. ISBN 0-262-61056-6.

Gutmann, B.; Thon, I.; and De Raedt, L. 2011. Learning the Parameters of Probabilistic Logic Programs from Interpretations. In Gunopulos, D.; Hofmann, T.; Malerba, D.; and Vazirgiannis, M., eds., *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I*, volume 6911 of *Lecture Notes in Computer Science*, 581–596. Springer. ISBN 978-3-642-23779-9.

Hadjichristodoulou, S.; and Warren, D. S. 2012. Probabilistic Logic Programming with Well-Founded Negation. In Miller, D. M.; and Gaudet, V. C., eds., *42nd IEEE International Symposium on Multiple-Valued Logic, ISMVL 2012, Victoria, BC, Canada, May 14-16, 2012*, 232–237. IEEE Computer Society. ISBN 978-1-4673-0908-0.

Hunter, A. 2013. A probabilistic approach to modelling uncertain logical arguments. *International Journal of Approximate Reasoning*, 54(1): 47–81.

Hunter, A.; Polberg, S.; and Thimm, M. 2020. Epistemic graphs for representing and reasoning with positive and negative influences of arguments. *Artificial Intelligence*, 281: 103236.

Kido, H.; and Okamoto, K. 2017. A Bayesian Approach to Argument-Based Reasoning for Attack Estimation. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 249–255. ijcai.org. ISBN 978-0-9992411-0-3.

Lee, J.; and Wang, Y. 2016. Weighted Rules under the Stable Model Semantics. In Baral, C.; Delgrande, J. P.; and Wolter,

F., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, 145–154. AAAI Press. ISBN 978-1-57735-755-1.

Li, H.; Oren, N.; and Norman, T. J. 2011. Probabilistic Argumentation Frameworks. In Modgil, S.; Oren, N.; and Toni, F., eds., *Theorie and Applications of Formal Argumentation - First International Workshop, TAFA 2011. Barcelona, Spain, July 16-17, 2011, Revised Selected Papers*, volume 7132 of *Lecture Notes in Computer Science*, 1–16. Springer. ISBN 978-3-642-29183-8.

Lukasiewicz, T. 1998. Probabilistic Logic Programming. In Prade, H., ed., *13th European Conference on Artificial Intelligence, Brighton, UK, August 23-28 1998, Proceedings.*, 388–392. John Wiley and Sons. ISBN 9780471984313.

Mantadelis, T.; and Bistarelli, S. 2020. Probabilistic abstract argumentation frameworks, a possible world view. *International Journal of Approximate Reasoning*, 119: 204–219.

Meert, W.; and Vennekens, J. 2014. Inhibited Effects in CP-Logic. In van der Gaag, L. C.; and Feelders, A. J., eds., *Probabilistic Graphical Models - 7th European Workshop, PGM 2014, Utrecht, The Netherlands, September 17-19, 2014. Proceedings*, volume 8754 of *Lecture Notes in Computer Science*, 350–365. Springer. ISBN 978-3-319-11432-3.

Nielsen, S. H.; and Parsons, S. 2006. A Generalization of Dung's Abstract Framework for Argumentation: Arguing with Sets of Attacking Arguments. In Maudet, N.; Parsons, S.; and Rahwan, I., eds., *Argumentation in Multi-Agent Systems, Third International Workshop, ArgMAS 2006, Hakodate, Japan, May 8, 2006, Revised Selected and Invited Papers*, volume 4766 of *Lecture Notes in Computer Science*, 54–73. Springer. ISBN 978-3-540-75525-8.

Poole, D. 2008. The Independent Choice Logic and Beyond. In (De Raedt et al. 2008), 222–243.

Sato, T. 1995. A Statistical Learning Method for Logic Programs with Distribution Semantics. In Sterling, L., ed., *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, 715–729. MIT Press. ISBN 0-262-69177-9.

Sato, T.; and Kameya, Y. 2008. New Advances in Logic-Based Probabilistic Modeling by PRISM. In (De Raedt et al. 2008), 118–155.

Stede, M.; Afantenos, S. D.; Peldszus, A.; Asher, N.; and Perret, J. 2016. Parallel Discourse Annotations on a Corpus of Short Texts. In Calzolari, N.; Choukri, K.; Declerck, T.; Goggi, S.; Grobelnik, M.; Maegaard, B.; Mariani, J.; Mazo, H.; Moreno, A.; Odijk, J.; and Piperidis, S., eds., *Proceedings of the Tenth International Conference on Language Resources and Evaluation LREC 2016, Portorož, Slovenia, May 23-28, 2016*. European Language Resources Association (ELRA).

Vennekens, J. 2013. Negation in the Head of CP-logic Rules. *CoRR*, abs/1312.6156.

Vennekens, J.; Denecker, M.; and Bruynooghe, M. 2009. CP-logic: A language of causal probabilistic events and its relation to logic programming. *TPLP*, 9(3): 245–308.