Elements for Weighted Answer-Set Programming

Francisco Coelho[1]
NOVA-LINCS, University of Évora, Portugal

Bruno Dinis
CIMA, University of Évora, Portugal

Dietmar Seipel
Universität Würzburg, Germany

Salvador Abreu
NOVA-LINCS, University of Évora, Portugal

Abstract

Logic programs, more specifically, answer-set programs, can be annotated with probabilities on facts to express uncertainty. We address the problem of propagating weight annotations on facts (*e.g.* probabilities) of an answer-set program to its stable models, and from there to events (defined as sets of atoms) in a dataset over the program's domain.

We propose a novel approach which is algebraic in the sense that it relies on an equivalence relation over the set of events. Uncertainty is then described as polynomial expressions over variables.

We propagate the weight function in the space of models and events, rather than doing so within the syntax of the program. As evidence that our approach is sound, we show that certain facts behave as expected. Our approach allows us to investigate weight annotated programs and to determine how suitable a given one is for modeling a given dataset containing events. It's core is illustrated by a running example and the encoding of a Bayesian network.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Answer-Set Programming, Stable Models, Probabilistic Logic Programming

Digital Object Identifier 10.4230/OASIcs.SLATE.2025.3

Related Version *Full Version*: `https://arxiv.org/abs/2503.20849`

Funding This work is supported by UID/04516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP.

Acknowledgements The authors are grateful to Lígia Henriques-Rodrigues, Matthias Knorr and Ricardo Gonçalves for valuable comments on a preliminary version of this paper, and Alice Martins for contributions on software development.

1 Introduction

Using a logic program (LP) to model and reason over a real world scenario is often difficult because of uncertainty underlying the problem being worked on. Classic LPs represent knowledge in precise and complete terms, which turns out to be problematic when the scenario is characterized by stochastic or observability factors. We aim to explore how answer-set programs (ASPs) plus weight annotated facts can lead to useful characterizations for this class of problems. To setup a working framework, we make the following assumption:

[1] Corresponding Author

▶ **Assumption 1** (System Representation, Data and Propagation). *Consider a* system *whose states are* partially observable *(*i.e. *observations can miss some state information) or* stochastic *(*i.e. *observed values are affected by random noise). We assume that knowledge about such system features a formal specification including* weighted facts *and empirical* data *such that:*

Representation *The system has a formal* representation[2] *in the form of a certain logic program; The program's stable models correspond one-to-one with the system states.*

Data Data *is a set of observations; a single* observation *(of the system states) results from a set of (boolean)* sensors*.*

Propagation *The* weights *in* facts *are* propagated *to the stable models of the representation.*

In this setting, data from observations can be used to estimate some parameteres used in the propagation process and, more importantly, to address the question of "*How accurate is the representation of the system?*". Other probabilistic logic programming (PLP) systems such as `Problog` [7], `P-log` [4] or LP^{MLN} [15], in line with Kifer and Subrahmanian [13], derive probability distributions from program syntax, limiting them to prior information. We question such methods and address stable model (SM) uncertainty by including parameters for unknown quantities, estimable with data. To frame this setting we extend our assumptions:

▶ **Assumption 2** (Sensor Representation and Events).

Sensors *The* sensors *of the system's states are associated to some atoms in the representation;*

Events *An* event *is a set of atoms from the representation.*

Following the terminology set by Calimeri *et al.* [5], sensors activate atoms (a) whereas no activation is represented by *default negation* (naf) $\sim a$. Negated atoms ($\neg a$) work similarly. By assumption 2, events can represent hidden or faulty sensors, like $\{a, \neg a, \sim b, \sim\neg b, \sim c, \neg c\}$, where a and $\neg a$ are activated, b is hidden, and $\neg c$ is consistently activated. Not all events are observations, and some may not uniquely determine system states – how to associate events to stable models and, thus, to system states, is addressed in Section 4.

If we (i) omit the naf-literals; (ii) use $\overline{x}$ to denote the classical $\neg x$; (iii) and use expressions like ab to denote sets of literals such as $\{a, b\}$, then the event $\{a, \neg a, \sim b, \sim\neg b, \sim c, \neg c\}$ can be shortened to the equivalent form $a\overline{a}\overline{c}$. We follow the convention of denoting a model by "*true*" atoms (positive or negative), with "*falsehood*" resulting from default negation [11]. Models include atoms like a or $\overline{b}$ but not literals $\sim a$. Sensor input is represented using positive and negative atoms, considering different sensors or a single sensor yielding positive/negative values.

Weights, not probabilities, are used to define a weight function on SMs, extended to all events. The step from facts to SMs is non-deterministic, and parameters represent non-unique choices, estimable with data. ASPs, based on SMs semantics, use SAT solving [10, 1, 19] or top-down search [2, 3, 18]. Distribution semantics (DS) [25, 23] extends logic programs with probabilistic reasoning. We are particularly interested in the following setting and application scenarios of such an extension to logic programs:

Partial observability A system's state can have *hidden variables*, not reported by the sensors.

Sensor error Information gathered from the sensors can carry *stochastic perturbations.*

Representation induction Combine representations and data to induce *more accurate representations.*

[2] We use "representation" instead of "model" to avoid confusion with the *stable models* of answer-set programs.

Probabilistic tasks Support common probabilistic tasks such as maximum a posteriori (MAP), maximum likelihood estimation (MLE) and Bayesian inference (BI) on the *representation domain i.e.* the set of all events.

Probabilistic tasks and simple examples

Maximum a posteriory (MAP) MAP estimates model parameters by combining prior beliefs with data likelihood to find the most probable values:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} \mathrm{P}(\theta \mid X)$$

For a biased coin with heads probability θ following a Beta distribution, observing 7 heads in 10 flips, the MAP estimate of θ maximizes the posterior distribution combining the prior Beta and the likelihood of the observations.

Maximum likelihood estimation (MLE) MLE estimates model parameters by maximizing the likelihood of observed data:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \mathrm{P}(X \mid \theta)$$

For 7 heads in 10 flips, the MLE estimate of the probability of heads is $\theta = 0.7$.

Bayesian inference (BI) Bayesian Inference updates hypothesis probabilities by combining prior beliefs with observed data to produce a new probability distribution.

$$\mathrm{P}(\theta \mid X) = \frac{\mathrm{P}(X \mid \theta)\,\mathrm{P}(\theta)}{\mathrm{P}(X)}$$

Bayesian Inference updates a Beta prior (α, β) with 7 heads and 3 tails from 10 flips, resulting in a Beta posterior $(\alpha + 7, \beta + 3)$.

The remainder of this article is structured as follows: Section 2 provides necessary context. In Section 3 we discuss the syntax and semantics of our proposed language for weighted answer-set programs (WASPs). We also define a weight function over total choices and address the issue of how to propagate these weights from facts to events, in Section 4. This method relies on an equivalence relation on the set of events. Furthermore, we express uncertainty by polynomial expressions over variables which depend on the total choices and on the stable models. By then the *Partial Observability* and *Sensor Error* points are addressed. An evidence that our approach is sound is given by Equation (28) where we show that replacing certain facts (with weight 1.0) by deterministic facts does not change the probability. Some final remarks and ideas for future developments including *Representation Induction* and the *Probabilistic Tasks* are presented in Section 5.

2 Framework

We start by refining assumptions 1 and 2 to set "representation" as an "ASP with weights":

▶ **Assumption 3** (Representation by answer-set programs with weigths)**.**

Answer-set programs and weights *A* representation *of a system is an* answer-set program *that includes* weighted facts*.*

A weighted fact (WF) *or an* annotated fact *has the from "*$a : w$*" where* a *is an atom,* $w \in [0, 1]$*, and "derives" the disjunctive fact "*$a \vee \overline{a}$*". A model may include either* a *or* $\overline{a}$ *but never both.*

Selecting one of $a, \overline{a}$ for each WF in a program will lead to a *total choice (TC)* [3]. Propagating weights from WFs to TCs is relatively straightforward (see Equation (6)) but propagation to events requires a step trough the program's stable models, addressed in Section 4.

About propagating weights from total choices

Propagating weights from TCs to SMs and events faces non-determinism, as seen in program P_1 from ex. 1, where multiple SMs (ab, ac) result from a single TC (a) without clear weight assignment. We use algebraic variables to address this, allowing deterministic propagation. Variable values can be estimated from data, contributing to Representation Induction. Related works use credal sets [6] or Bayesian approaches [28] to handle uncertainty, while our method remains algebraic.

3 Syntax and Semantics of Weighted ASP

We start the formal definition of *weighted answer-set program* with the setup and discussion of a minimal syntax and semantics of propositional ASP, without variables, functors or relation symbols, but enough to illustrate our method to propagate weights from annotated facts to events. From now on "$\neg x$" and "$\overline{x}$" denote classical negation and "$\sim x$" default negation.

Syntax

We slightly adapt Calimeri *et al.*'s approach [5]. Let $\mathcal{A}$ be a finite set of symbols, the *positive atoms*. For $a \in \mathcal{A}$, the expressions a and $\neg a$ (the later a *negative atom*, also denoted $\overline{a}$) are *(classical) atoms*. If a is an atom, the expressions a and $\sim a$ are *(naf-)literals*. A *rule* is of the form

$$h_1 \vee \cdots \vee h_n \leftarrow b_1 \wedge \cdots \wedge b_m$$

where the h_i are atoms and the b_j are literals. The symbol "$\leftarrow$" separates the *head* from the *body*. A rule is a *constraint*[4] if $n = 0$, *normal* if $n = 1$, *disjunctive* if $n > 1$, and a *fact* if $m = 0$.

An *answer-set program (ASP)* is a set P of facts and other rules, denoted, resp. $\mathcal{F}(P)$ and $\mathcal{R}(P)$, or simply $\mathcal{F}$ and $\mathcal{R}$. In a *normal program* all the rules are normal. Notice that programs with constraint or disjunctive rules can be converted into normal programs [9].

Semantics

The standard semantics of an ASP has a few different, but equivalent, definitions [17]. A common definition is as follows [11]: let P be a normal program. The Gelfond/Lifschitz *reduct* of P relative to the set X of atoms results from (i) deleting rules that contain a literal of the form $\sim p$ in the body with $p \in X$ and then (ii) deleting the remaining literals of the form $\sim q$ from the bodies of the remaining rules. Now, M is a *stable model (SM)* of P if it is the minimal model of the reduct of P relative to M. We denote by $\mathcal{S}(P)$, or simply $\mathcal{S}$, the set of stable models of the program P.

---

[3] We use the term "choice" for historical reasons, *e.g.* see [6] even though not related to the usual "choice" elements, atoms or rules from *e.g.* [5]

[4] An "*integrity constraint*" in [5].

Evaluation without grounding

While the most common form to generate stable models is based on *grounding*, a different approach is the one supported by `s(CASP)`, that evaluates ASP programs with function symbols and constraints without grounding, enabling human-readable explanations and addressing grounding-based solvers' issues of exponential growth and unnecessary complete model computation.

WASPs and their Derived Programs

If a is an atom and $w \in [0,1]$, a *weighted fact (WF)* (or a *fact with weight annotation*) is $a : w$. Notice that we have $w \in [0,1]$ but w is interpreted as a *balance* between the *choices* a and $\overline{a}$, and *not a probability*.

Weighted answer-set programs (WASPs) extend ASPs by adding WFs. We denote the set of weighted facts of a program P by $\mathcal{W}(P)$, and $\mathcal{A}_{\mathcal{W}}(P)$ the set of positive atoms in $\mathcal{W}$. When possible we simplify notation as $\mathcal{W}, \mathcal{A}_{\mathcal{W}}$.

Our WASPs definition is restricted to illustrate weight propagation from TCs to events, excluding logical variables, relation symbols, and functors. Weight annotations aren't used on rule heads or disjunctions, but this doesn't limit expressiveness:

$$\alpha : w \leftarrow \beta \qquad \Longrightarrow \qquad \begin{cases} \gamma : w, \\ \alpha \leftarrow \beta \wedge \gamma \end{cases} \tag{1}$$

while annotated disjunctive facts

$$\alpha \vee \beta : w \qquad \Longrightarrow \qquad \begin{cases} \gamma : w, \\ \alpha \vee \beta \leftarrow \gamma, \\ \overline{\alpha} \leftarrow \overline{\gamma}, \quad \overline{\beta} \leftarrow \overline{\gamma}. \end{cases} \tag{2}$$

Derived program

The *derived program* of a WASP is the ASP obtained by replacing each weighted fact $a : w$ by the derived disjunction $a \vee \overline{a}$. The *stable models* of a WASP program are the stable models of its derived program. So, we also denote the set of SMs of a (derived or) WASP program P by $\mathcal{S}(P)$ or $\mathcal{S}$.

Events

An *event* of a program P is a set of atoms from P. We denote the set of events by $\mathcal{E}(P)$ or simply $\mathcal{E}$. An event $e \in \mathcal{E}$ which includes a set $\{x, \overline{x}\}$ is said to be *inconsistent*; otherwise it is *consistent*. The set of consistent events is denoted by $\mathcal{C}$.

▶ **Example 1** (A simple weighted answer-set program)**.** Consider the following WASP :

$$P_1 = \begin{cases} a : 0.3, \\ b \vee c \leftarrow a \end{cases} \tag{3}$$

with weighted facts $\mathcal{W} = \{a : 0.3\}$. The derived program is the ASP

$$P_1' = \begin{cases} a \vee \overline{a}, \\ b \vee c \leftarrow a, \end{cases} \tag{4}$$

with SMs $\mathcal{S} = \{\, \overline{a}, ab, ac \,\}$. The atoms are $\mathcal{A} = \left\{ a, \overline{a}, b, \overline{b}, c, \overline{c} \right\}$ and the events are $\mathcal{E} = 2^{\mathcal{A}}$ [5].

[5] 2^X is the *power set* of X: $A \in 2^X \Leftrightarrow A \subseteq X$.

Total Choices and their Weights

A disjunctive head $a \vee \overline{a}$ in the derived program represents a single *choice*, either a or $\overline{a}$. We define the set of *total choices (TCs)* of a set of atoms by the recursion

$$\begin{cases} \mathcal{T}(\emptyset) = \emptyset, \\ \mathcal{T}(X \cup a) = \bigcup\limits_{t \in \mathcal{T}(X)} (t \cup a) \quad \cup \quad \bigcup\limits_{t \in \mathcal{T}(X)} (t \cup \overline{a}) \end{cases} \tag{5}$$

where X is a set of atoms and a is an atom. The total choices of a WASP P are the TCs of the positive atoms in it's weighted facts: $\mathcal{T}(P) = \mathcal{T}\big(\mathcal{A}_{\mathcal{W}}(P)\big)$. When possible we write simply $\mathcal{T}$. Given a WASP, the *weight of the total choice* $t \in \mathcal{T}$ is given by the product

$$\omega_{\mathcal{T}}(t) = \prod_{\substack{a : w \in \mathcal{W}, \\ a \in t}} w \quad \times \quad \prod_{\substack{a : w \in \mathcal{W}, \\ \overline{a} \in t}} \overline{w}. \tag{6}$$

Here $\overline{w} = 1 - w$, and we use the subscript in $\omega_{\mathcal{T}}$ to explicitly state that this function concerns total choices. Later we'll use subscripts $\mathcal{S}, \mathcal{E}$ to deal with weight functions of stable models and events, $\omega_{\mathcal{S}}, \omega_{\mathcal{E}}$. Some stable models are entailed from some total choices while other SMs are entailed by other TCs. We write $\mathcal{S}(t)$ to represent the set of stable models entailed by the total choice $t \in \mathcal{T}$. Our goal can now be rephrased as to know how to propagate the weights of the program's total choices, $\omega_{\mathcal{T}}$, in Equation (6) to the program's events, $\omega_{\mathcal{E}}$ to be defined later, in Equations (21a) and (21b).

Propagation of weights

As a first step to propagate weight from total choices to events, consider the program P_1 of Equation (3) and a possible propagation of $\omega_{\mathcal{T}} : \mathcal{T} \to [0, 1]$ from total choices to the stable models, $\omega_{\mathcal{S}} : \mathcal{S} \to [0, 1]$ (still informal, see Equation (16)). It might seem straightforward, in ex. 1, to set $\omega_{\mathcal{S}}(\overline{a}) = 0.7$ but there is no *explicit* way to assign values to $\omega_{\mathcal{S}}(ab)$ and $\omega_{\mathcal{S}}(ac)$. We represent this non-determinist by a parameter θ as in

$$\begin{aligned} &\omega_{\mathcal{S}}(ab) = 0.3\,\theta, \\ &\omega_{\mathcal{S}}(ac) = 0.3\,(1 - \theta) \end{aligned} \tag{7}$$

to express our knowledge that ab and ac are models entailed from a specific choice and, simultaneously, the inherent non-determinism of that entailment. In general, it might be necessary to have several such parameters, each associated to a given stable model s (in Equation (7), $s = ab$ in the first line and $s = ac$ in the second line) and a total choice t ($t = a$ above), so we write $\theta_{s,t}$. Obviously, for reasonable $\theta_{s,t}$, the total choice t must be a subset of the stable model s.

Unless we introduce some bias, such as $\theta = 0.5$ as in $\mathtt{LP}^{\mathtt{MLN}}$ [15], the values for $\theta_{s,t}$ can't be determined just with the information given in the program. But it might be estimated with the help of further information, such as an empirical distribution from a dataset. Further discussion of this point is outside the scope of this paper.

Now consider the program

$$\begin{cases} a : 0.3, \\ b \leftarrow a \wedge {\sim}b \end{cases} \tag{8}$$

that has a single SM, $\overline{a}$. Since the weights are not interpreted as probabilities, there is no need to have the sum on the stable models equal to 1. So the weights in the TCs of Equation (8) only set

$$\omega_{\mathcal{S}}(\overline{a}) = 0.7.$$

In this case, if we were to derive a probability of the SMs, normalization would give $\mathrm{P}(\overline{a}) = 1.0$.

Also facts without annotations can be transformed into facts with weight 1:

$$a \qquad \Longrightarrow \qquad a : 1.0. \tag{9}$$

The method that we are proposing does not follow the framework of Kifer and Subrahmanian [13] and others, where the syntax of the program determines the propagation from probabilities explicitly set either in facts or other elements of the program. Our approach requires that we consider the semantics, *i.e.* the stable models of the program, independently of the syntax that provided them. From there we propagate weights to the program's events and then, if required, normalization provides the final probabilities. Moreover, we allow the occurrence of variables in the weights, in order to deal with the non-determinism that results from the non-uniqueness of SMs entailed from a single TC. These variables can be later estimated from available data.

Related Approaches and Systems

The core problem of setting a semantics for probabilistic logic programs, the propagation of probabilities from total choices to stable models in the case of ASP or to other types in other logic programming systems (*e.g.* to possible worlds in `Problog`) has been studied for some time [13, 25]. For example, the *credal set* approach [6], defines $\mathrm{P}_{\mathcal{T}}$ in a way similar to Equation (6) but then, for $a \in \mathcal{A}, t \in \mathcal{T}$, the probability $\mathrm{P}(a \mid t)$ is unknown but bounded by $\underline{\mathrm{P}}(a \mid t)$ and $\overline{\mathrm{P}}(a \mid t)$, that can be explicitly estimated from the program.

`Problog` [8, 28] extends `Prolog` with probabilistic facts so that a program specifies a probability distribution over possible worlds. A *world* is a model of $T \cup R$ where T is a total choice and R the set of rules of a program. The semantics is only defined for *sound* programs [24] *i.e.*, programs for which each possible total choice T leads to a well-founded model that is two-valued or *total*. The probability of a possible world that is a model of the program is the probability of the total choice. Otherwise the probability is 0 [24, 27]. Another system, based on Markov Logic [22], is $\mathtt{LP}^{\mathtt{MLN}}$ [15, 16], whose models result from *weighted rules* of the form $a \leftarrow b \wedge n$ where a is disjunction of atoms, b is conjunction of atoms and n is constructed from atoms using conjunction, disjunction and negation. For each model there is a unique maximal set of rules that are satisfied by it and the respective weights determine the weight of that model, that can be normalized to a probability.

Towards Propagating Weights from Total Choices to Events

The program P_1 in Equation (3) from ex. 1 showcases the problem of propagating weights from total choices to stable models and then to events. The main issue arises from the lack of information in the program on how to assign *un-biased* weights to the stable models. This becomes crucial in situations where multiple stable models result from a single total choice.

Our assumptions 1–3 enunciate that a WASP program represents a *system*; the *states* of that system, which are partially observable and stochastic, are associated to the program's stable models; and state *observations* are encoded as events, *i.e.* sets of atoms of the program. Then:

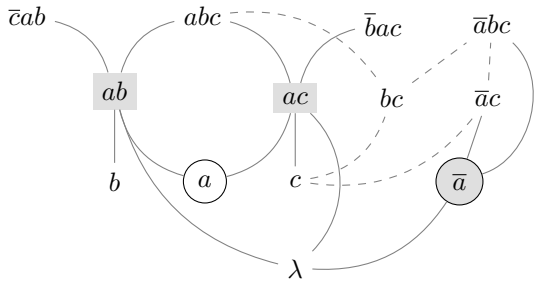


Figure 1 This diagram shows events related to the stable models of program P_1. Circle nodes are total choices, shaded nodes are SMs. Solid lines show SM relations, dashed lines show subset/superset relations. The set of events in all SMs, Λ, is $\{\lambda\}$ because $\overline{a} \cap ab \cap ac = \emptyset$.

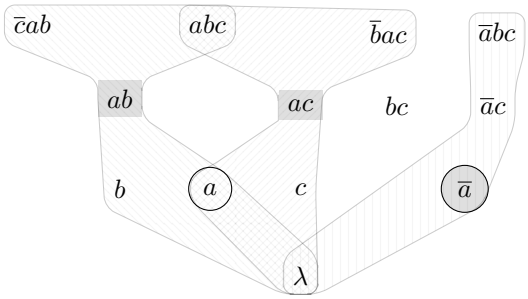


Figure 2 Classes of (consistent) events related to the stable models of P_1 are defined through sub-/super-set relations. In this picture we can see, for example, that $\{\overline{c}ab, ab, b\}$ and $\{a, abc\}$ are part of different classes, represented by different fillings. As before, the circle nodes are total choices and shaded nodes are stable models. Notice that bc is not in a filled area.

1. With a weight set for the stable models, we extend it to any event in the program domain.
2. If statistical knowledge is available, it's considered external and doesn't affect the propagation procedure.
3. However, that knowledge can be used to estimate the parameters $\theta_{s,t}$ and to "score" the program.
4. If a program is one of many candidates, its score can be used as fitness by algorithms searching for optimal programs in a dataset.
5. If events are not consistent with the program, then we ought to conclude that the program is wrong and must be changed accordingly.

We address propagating weights from stable models to events using parameters like θ. This defines a function $\mu_{\mathcal{E}}$ that can be normalized to set probabilities $\mathrm{P}_{\mathcal{E}}$, ensuring consistent probabilistic reasoning with the ASP program.

4 Propagating Weights

The diagram in Figure 1 shows the challenge of propagating weights from total choices to stable models and then to general events, where node values depend on neighbours. This leads to interpretation issues, such as assigning unexplained values to nodes like bc. We propose basing propagation on the event's relation to stable models instead.

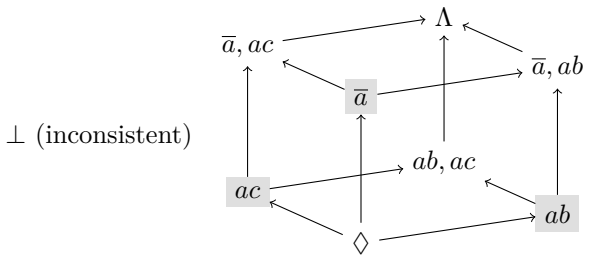


Figure 3 This diagram shows the lattice of stable cores from ex. 1. Nodes are stable cores derived from stable models, plus the inconsistent class. The bottom node is the independent events class, with no relation to SMs. The top node represents events related to all SMs (*consequences*). Shaded nodes are SMs.

4.1 An Equivalence Relation

Our approach to propagating weights views stable models as *prime factors* or "irreducible events". Events are considered based on their relation to SMs. In ex. 1, a relates to ab and ac, while c only relates to ac. Thus, a and c relate to different SMs, but a and abc relate to the same SMs. We formalize this relation. The *stable core (SC)* of the event $e \in \mathcal{E}$ is

$$[\![e]\!] := \{\, s \in \mathcal{S} \mid s \subseteq e \vee e \subseteq s \,\} \tag{10}$$

where $\mathcal{S}$ is the set of stable models.

Notice that the minimality of stable models implies that either e is a stable model or at least one of $\exists s\,(s \subseteq e)\,, \exists s\,(e \subseteq s)$ is false *i.e.,* no stable model contains another. We now define an equivalence relation so that two events are related if either both are inconsistent or both are consistent and, in the latter case, with the same stable core.

▶ **Definition 2** (Equivalence Relation on Events). *For a given program, let $u, v \in \mathcal{E}$. The equivalence relation $u \sim v$ is defined by*

$$u, v \notin \mathcal{C} \vee \big(u, v \in \mathcal{C} \wedge [\![u]\!] = [\![v]\!]\big)\,. \tag{11}$$

This equivalence relation defines a partition on the set of events, where each class holds a unique relation with the stable models. In particular we denote each class by:

$$[e]_\sim = \begin{cases} \bot := \mathcal{E} \setminus \mathcal{C} & \text{if } e \in \mathcal{E} \setminus \mathcal{C}, \\ \big\{u \in \mathcal{C} \;\big|\; [\![u]\!] = [\![e]\!]\big\} & \text{if } e \in \mathcal{C}. \end{cases} \tag{12}$$

where $\bot$ denotes the set $\mathcal{E} \setminus \mathcal{C}$ of *inconsistent* events, *i.e.* events that contain $\{x, \overline{x}\}$ for some atom x.

Let λ be the empty set event (notice that $\lambda = \emptyset \in \mathcal{E}$) [6], and Λ the *consequence class* of (consistent) events related with all the stable models. Then

$$[\lambda]_\sim = \Lambda. \tag{13}$$

The combinations of stable models, *i.e.* the stable cores, together with the set of inconsistent events ($\bot$) forms a set of representatives for the equivalence relation $\sim$. Since all

[6] We adopt the notation "λ" for *empty word*, from formal languages, to distinguish "$\emptyset \in \mathcal{E}$" from "$\emptyset \subset \mathcal{E}$".

events within a consistent equivalence class have the same stable core, we are interested in functions (including weight assignments), that are constant within classes. A function $f : \mathcal{E} \to Y$, where Y is any set, is said to be *coherent* if

$$\forall e \in \mathcal{E}\ \forall u \in [e]_\sim \left(f(u) = f(e)\right). \tag{14}$$

Considering coherent functions, in the specific case of Equation (3), instead of dealing with the $2^6 = 64$ events, we need to consider only the $2^3 + 1 = 9$ classes, well defined in terms of combinations of the stable models, to define coherent functions. In general, a program with n atoms and m stable models has 2^{2n} events and $2^m + 1$ stable cores – but easily $m \gg n$.

4.2 From Total Choices to Events

The "propagation" phase, traced by Equation (6) and Equations (16)–(21b), starts with the weight of total choices, $\omega_\mathcal{T}(t)$, propagates it to the stable models, $\omega_\mathcal{S}(s)$, and then, within the equivalence relation from Equation (11), to a coherent weight of events, $\omega_\mathcal{E}(e)$. So we are specifying a sequence of functions

$$\omega_\mathcal{T} \longrightarrow \omega_\mathcal{S} \longrightarrow \omega_\mathcal{R} \longrightarrow \omega_\mathcal{E} \tag{15}$$

on successive larger domains

$$\mathcal{T} \longrightarrow \mathcal{S} \longrightarrow [\mathcal{E}]_\sim \longrightarrow \mathcal{E}$$

such that the last function ($\omega_\mathcal{E}$) is a finite coherent function on the set of events and thus, as a final step, it can easily be used to define a probability distribution of events by normalization:

$$\omega_\mathcal{E} \longrightarrow \mathrm{P}_\mathcal{E}.$$

Total choices and Stable models

Let's start by looking into the first two steps of the sequence of functions Equation (15): $\omega_\mathcal{T}$ and $\omega_\mathcal{S}$. The weight $\omega_\mathcal{T}$ of the total choice $t \in \mathcal{T}$ is already given by Equation (6). Recall that each total choice $t \in \mathcal{T}$, together with the rules and the other facts of a program, defines the set $\mathcal{S}(t)$ of stable models associated with that choice. Given a total choice $t \in \mathcal{T}$, a stable model $s \in \mathcal{S}$, and formal variables or values $\theta_{s,t} \in [0, 1]$ such that $\sum_{s \in \mathcal{S}(t)} \theta_{s,t} = 1$, we define

$$\omega_\mathcal{S}(s, t) := \begin{cases} \theta_{s,t} & \text{if } s \in \mathcal{S}(t) \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

The $\theta_{s,t}$ parameters in Equation (16) express the *program's* lack of information about the weight assignment, when a single total choice entails more than one stable model. We address this issue by assigning a possibly unknown parameter, *i.e.* a formal algebraic variable ($\theta_{s,t}$) associated with a total choice (t) and a stable model (s). This allows the expression of a quantity that does not result from the program's syntax but can be determined or estimated given more information, *e.g.* observed data.

As sets, two stable models can have non-empty intersection. But because different SMs represent different states of a system –which are *disjoint events* – we assume that the algebra of the stable models is σ-additive.

▶ **Assumption 4** (Stable models as disjoint events). *For any set X of stable models and any total choice t,*

$$\omega_{\mathcal{S}}(X,t) = \sum_{s\in X} \omega_{\mathcal{S}}(s,t)\,. \tag{17}$$

Equation (17) is the basis for Equation (19a) and effectively extends $\omega_{\mathcal{S}} : \mathcal{S}\times\mathcal{T}\to\mathbb{R}$ to $\omega_{\mathcal{S}} : 2^{\mathcal{S}}\times\mathcal{T}\to\mathbb{R}$. Now the pre-condition of Equation (16) can be stated as $\omega_{\mathcal{S}}\big(\mathcal{S}(t)\,,t\big) = 1$.

Classes

The next step in the sequence of Equation (15) is the function $\omega_{\mathcal{R}}$ on $[\mathcal{E}]_\sim$. Each class of the relation $\sim$ (eq. 11) is either the inconsistent class ($\bot$) or is associated with a stable core, *i.e.* a set of stable models. Therefore, $\omega_{\mathcal{R}}$ is defined considering the following two cases.

Inconsistent class

This class contains inconsistent events; they should not be observed and have weight zero.

$$\omega_{\mathcal{R}}(\bot,t) := 0. \tag{18}$$

Consistent classes

To be coherent, the propagation function must be constant within a class and its value dependent only on the stable core:

$$\omega_{\mathcal{R}}\big([e]_\sim,t\big) := \omega_{\mathcal{S}}\big(\llbracket e\rrbracket\,,t\big) = \sum_{s\in\llbracket e\rrbracket} \omega_{\mathcal{S}}(s,t)\,. \tag{19a}$$

and we further define the following:

$$\omega_{\mathcal{R}}\big([e]_\sim\big) := \sum_{t\in\mathcal{T}} \omega_{\mathcal{T}}(t)\,\omega_{\mathcal{R}}\big([e]_\sim,t\big) \tag{19b}$$

Equation (19a) states that the weight of a class $[e]_\sim$ is the weight of its stable core ($\llbracket e\rrbracket$) and Equation (19b) *averages* Equation (19a) over the total choices. Notice that Equation (19a) also applies to the independent class, $\lozenge = \big\{e \;\big|\; \llbracket e\rrbracket = \emptyset\big\}$, because events in this class are not related with any stable model:

$$\omega_{\mathcal{R}}(\lozenge,t) = \sum_{s\in\emptyset} \omega_{\mathcal{S}}(s,t) = 0. \tag{20}$$

Events and Probability

Each consistent event $e\in\mathcal{E}$ is in the class defined by its stable core $\llbracket e\rrbracket$. So, denoting the number of elements in X as $\#X$, we set:

$$\omega_{\mathcal{E}}(e,t) := \begin{cases} \frac{\omega_{\mathcal{R}}\left([e]_\sim,t\right)}{\#[e]_\sim} & \text{if } \#[e]_\sim > 0,\\ 0 & \text{otherwise.}\end{cases} \tag{21a}$$

and, by averaging over the total choices:

$$\omega_{\mathcal{E}}(e) := \sum_{t\in\mathcal{T}} \omega_{\mathcal{T}}(t)\,\omega_{\mathcal{E}}(e,t)\,. \tag{21b}$$

The Equation (21b) is the main goal of this paper: propagate the weights associated to facts of an WASP to the set of all events of that program. In order to get a probability from Equation (21b), concerning the *Probabilistic Tasks* goal, we define the *normalizing factor*:

$$Z := \sum_{e \in \mathcal{E}} \omega_{\mathcal{E}}(e) = \sum_{[e]_\sim \in [\mathcal{E}]_\sim} \omega_{\mathcal{R}}\left([e]_\sim\right), \tag{22}$$

and now Equation (21b) provides a straightforward way to define the *probability of a single event* $e \in \mathcal{E}$:

$$\mathrm{P}_{\mathcal{E}}(e) := \frac{\omega_{\mathcal{E}}(e)}{Z}. \tag{23}$$

Equation (23) defines a coherent *prior*[7] probability of events and, together with external statistical knowledge, can be used to learn about the *initial* probabilities of the atoms.

The effect of propagation

To assess weight propagation from SMs to events, compare $\mathrm{P}_{\mathcal{E}}$ with syntactically induced probabilities from fact weights. It is sufficient to check if $\mathrm{P}_{\mathcal{E}}(t) = \mathrm{P}_{\mathcal{T}}(t)$ for all total choices. Normalize TCs weights to get a probability distribution. For $t \in \mathcal{T}$,

$$\mathrm{P}_{\mathcal{T}}(t) = \frac{\omega_{\mathcal{T}}(t)}{\sum_{\tau \in \mathcal{T}} \omega_{\mathcal{T}}(\tau)} \tag{24}$$

And now we ask if these probabilities coincide in $\mathcal{T}$: $\forall t \in \mathcal{T}\left(\mathrm{P}_{\mathcal{E}}(t) = \mathrm{P}_{\mathcal{T}}(t)\right)$?

It is easy to see that, in general, this cannot be true. While the domain of $\mathrm{P}_{\mathcal{T}}$ is the set of total choices, for $\mathrm{P}_{\mathcal{E}}$ the domain is much larger, including all the events. Except for trivial programs, some events other than total choices will have non-zero weight: If a program has a consistent event $e \in \mathcal{C} \setminus \mathcal{T}$ such that $\mathrm{P}_{\mathcal{E}}(e) \neq 0$ then there is at least one $t \in \mathcal{T}$ such that

$$\mathrm{P}_{\mathcal{T}}(t) \neq \mathrm{P}_{\mathcal{E}}(t)\,. \tag{25}$$

The essential, perhaps *counter-intuitive*, conclusion of Equation (25) is that we are dealing with *two distributions*: $\mathrm{P}_{\mathcal{T}}$, restricted to the total choices, results *syntactically* from the annotations of the program, while $\mathrm{P}_{\mathcal{E}}$, extended to events, results from both the annotations and the program's *semantics*, *i.e.* the stable models. For ex. 1:

$\mathrm{P}_{\mathcal{T}}(a) = 0.3$ from the program and $\mathrm{P}_{\mathcal{E}}(a) = \dfrac{3}{64}$ from Equation (29).

Now $\mathrm{P}_{\mathcal{E}} : \mathcal{E} \to [0,1]$ can be extended to $\mathrm{P}_{\mathcal{E}} : 2^{\mathcal{E}} \to [0,1]$ by abusing notation and setting, for $X \subseteq \mathcal{E}$,

$$\mathrm{P}_{\mathcal{E}}(X) = \sum_{x \in X} \mathrm{P}_{\mathcal{E}}(x)\,. \tag{26}$$

It is straightforward to verify that the latter satisfies the Kolmogorov axioms of probability.

We can now properly state the following property about *certain facts* such as $a : 1.0$. Consider a program A with the weighted fact $\alpha : 1.0$ and B that results from A by replacing that fact by the deterministic fact α. Then

$$\forall e \in \mathcal{E}\left(\omega^A_{\mathcal{E}}(e) = \omega^B_{\mathcal{E}}(e)\right). \tag{27}$$

[7] In the Bayesian sense that future observations might update this probability.

Normalization of Equation (27) entails that, for $\mathrm{P}^A_{\mathcal{E}}$ given by Equation (23) for A and $\mathrm{P}^B_{\mathcal{E}}$ for B, then

$$\forall e \in \mathcal{E}\left(\mathrm{P}^A_{\mathcal{E}}(e) = \mathrm{P}^B_{\mathcal{E}}(e)\right). \tag{28}$$

▶ **Example 3** (Probability of Events). The coherent *prior* probability of events of program P_1 in ex. 1 is

$[\![e]\!]$	$\bot$	$\Diamond$	$\overline{a}$	ab	ac	$\overline{a}, ab$	$\overline{a}, ac$	ab, ac	Λ
$\mathrm{P}_{\mathcal{E}}(e)$	0	0	$\frac{7}{207}$	$\frac{1}{23}\theta$	$\frac{1}{23}\overline{\theta}$	0	0	$\frac{3}{46}$	$\frac{10}{23}$

(29)

To compute the probability of $e \in \mathcal{E}$ find the column of the event's stable core:

- $\omega_{\mathcal{E}}(ab) = \frac{\theta}{23}$, because ab is the only SM related with ab so $[\![ab]\!] = \{ab\}$ and the weight value is found in the respective column of Equation (29).
- $\omega_{\mathcal{E}}(abc) = \frac{3}{46}$ because $abc \supset ab$ and $abc \supset ac$. So $[\![abc]\!] = \{ab, ac\}$.
- $\omega_{\mathcal{E}}(bc) = 0$ because, since there is no SM s that either $s \subset bc$ or $bc \subset s$, $[\![bc]\!] = \emptyset$ *i.e.* $bc \in \Diamond$.
- $\omega_{\mathcal{E}}(\overline{a}) = \frac{7}{207}$ and $\omega_{\mathcal{E}}(a) = \frac{3}{46}$.

Notice that $\omega_{\mathcal{E}}(\overline{a}) + \omega_{\mathcal{E}}(a) \neq 1$. This highlights the fundamental difference between $\omega_{\mathcal{E}}$ and $\omega_{\mathcal{T}}$ (*cf.* Equation (25)), where the former results from the lattice of the stable cores and the latter directly from the explicit assignment of probabilities to literals. Related with this case, consider the *complement* of a consistent event e, denoted by $\complement e$. To calculate $\mathrm{P}_{\mathcal{E}}(\complement e)$ find the classes in $[\mathcal{E}]_\sim$ that are not $[e]_\sim$, *i.e.* the complement of e's class within $[\mathcal{E}]_\sim$[8], $\complement[e]_\sim$. Considering that $[\mathcal{E}]_\sim$ is in a one-to-one correspondence with the stable cores plus $\bot$,

$$[\mathcal{E}]_\sim \simeq \left\{\bot, \Diamond, \{\overline{a}\}, \{ab\}, \{ac\}, \{\overline{a}, ab\}, \{\overline{a}, ac\}, \{ab, ac\}, \Lambda\right\}.$$

In particular, for $\omega_{\mathcal{E}}(\complement a)$, since $[\![a]\!] = \{ab, ac\}$ then $\complement[a]_\sim = [\mathcal{E}]_\sim \setminus [a]_\sim$ and $\mathrm{P}_{\mathcal{E}}(\complement a) = \mathrm{P}_{\mathcal{E}}([\mathcal{E}]_\sim \setminus [a]_\sim) = 1 - \mathrm{P}_{\mathcal{E}}(a)$. Also, $\mathrm{P}_{\mathcal{E}}(\complement\overline{a}) = 1 - \mathrm{P}_{\mathcal{E}}(\overline{a})$.

While not illustrated in our examples, this method also applies to programs that have more than one probabilistic fact, like

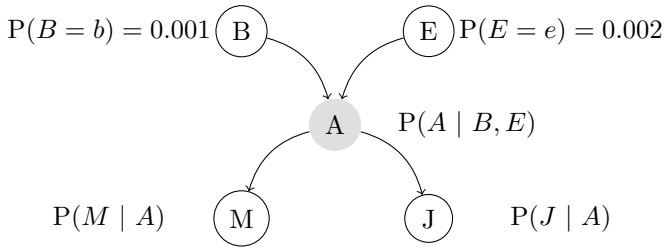
$$\begin{cases} a : 0.3, \qquad b : 0.6, \\ c \vee d \leftarrow a \wedge b. \end{cases}$$

4.3 Encoding of Bayesian Networks

Our approach generalizes to Bayesian networks similarly to previous works [6, 21, 12, 26]. Acyclic propositional programs can be seen as Bayesian networks with binary variables, where the structure is the dependency graph, and variables correspond to atoms with probabilities from facts and rules. Conversely, any binary Bayesian network can be specified by an acyclic non-disjunctive WASP. A classic example in Bayesian networks involves the events Burglary (B), Earthquake (E), Alarm (A), and calls from Mary (M) and John (J) [20]. These events have conditional probabilities, such as the alarm going off given a burglary or earthquake, and neighbors calling if they hear the alarm. The example illustrates reasoning under uncertainty. We follow the convention of representing the (upper case) random variable X by the (lower case) positive atom x. From Figure 4 we obtain the following specification:

$$\begin{cases} b : 0.001, \\ e : 0.002 \end{cases}$$

[8] All the usual set operations hold on the complement. For example, $\complement\complement X = X$.



P(M | A)	m	$\overline{m}$
a	0.9	0.1
$\overline{a}$	0.05	0.95

P(J | A)	j	$\overline{j}$
a	0.7	0.3
$\overline{a}$	0.01	0.99

P(A | B, E)		a	$\overline{a}$
b	e	0.95	0.05
b	$\overline{e}$	0.94	0.06
$\overline{b}$	e	0.29	0.71
$\overline{b}$	$\overline{e}$	0.001	0.999

Figure 4 The Earthquake, Burglary, Alarm model.

For the table giving the probability $\mathrm{P}(M \mid A)$ we obtain, cf. Equation (1), the program

$$\begin{cases} m : 0.9 \leftarrow a, \\ m : 0.05 \leftarrow \overline{a} \end{cases}$$

Similarly, for the probability $\mathrm{P}(J \mid A)$ we obtain

$$\begin{cases} j : 0.7 \leftarrow a, \\ j : 0.01 \leftarrow \overline{a}, \end{cases}$$

Finally, for the probability $\mathrm{P}(A \mid B \wedge E)$ we obtain

$$\begin{cases} a : 0.95 \leftarrow b \wedge e, \quad a : 0.94 \leftarrow b \wedge \overline{e}, \\ a : 0.29 \leftarrow \overline{b} \wedge e, \quad a : 0.001 \leftarrow \overline{b} \wedge \overline{e}. \end{cases}$$

One can then proceed as in the previous subsection and analyze this example. The details of such analysis are not given here since they are analogous, albeit admittedly more cumbersome.

5 Discussion and Future Work

This work introduces weight assignments using algebraic expressions from ASPs, with much to explore regarding their full expressive power, including recursion and logical variables. Bayesian Networks' theory and tools can be adapted, while connections to Markov Fields [14] and program selection applications are future work. The equivalence relation identifies subset cases, and better relations between SMs are possible. Inconsistent events' weights are set to 0, but inconsistencies from noise in observations may require reconsideration.

References

1 Weronika T Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri, et al. The asp system dlv: advancements and applications. *KI-Künstliche Intelligenz*, 32:177–179, 2018. doi:10.1007/S13218-018-0533-0.

2 Marco Alberti, Elena Bellodi, Giuseppe Cota, Fabrizio Riguzzi, and Riccardo Zese. cplint on swish: Probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11(1):47–64, 2017. doi:10.3233/IA-170106.

3 Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. Justifications for goal-directed constraint answer set programming. *arXiv preprint arXiv:2009.10238*, 2020.

4 Chitta Baral, Michael Gelfond, and Nelson Rushton. Probabilistic reasoning with Answer Sets. *Theory and Practice of Logic Programming*, 9(1):57–144, 2009. doi:10.1017/S1471068408003645.

5 Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, 2020. doi:10.1017/S1471068419000450.

6 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. doi:10.1016/J.IJAR.2020.07.004.

7 Luc De Raedt, Angelika Kimmig, Hannu Toivonen, and M Veloso. Problog: A probabilistic Prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence*, pages 2462–2467. IJCAI-INT JOINT CONF ARTIF INTELL, 2007.

8 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. doi:10.1017/S1471068414000076.

9 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer set solving in practice*. Springer Nature, 2022.

10 Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011. doi:10.3233/AIC-2011-0491.

11 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080. Cambridge, MA, 1988.

12 Werner Kießling, Helmut Thöne, and Ulrich Güntzer. Database support for problematic knowledge. In *Advances in Database Technology - EDBT'92: 3rd International Conference on Extending Database Technology Vienna, Austria, March 23–27, 1992 Proceedings 3*, pages 421–436. Springer, 1992. doi:10.1007/BFB0032446.

13 Michael Kifer and VS Subrahmanian. Theory of generalized annotated logic programming and its applications. *The Journal of Logic Programming*, 12(4):335–367, 1992. doi:10.1016/0743-1066(92)90007-P.

14 Ross Kindermann and J. Laurie Snell. *Markov random fields and their applications*, volume 1 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 1980.

15 Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *Fifteenth international conference on the principles of knowledge representation and reasoning*, 2016.

16 Joohyung Lee and Zhun Yang. Lpmln, Weak Constraints, and P-log. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

17 Vladimir Lifschitz. Twelve definitions of a stable model. In *International Conference on Logic Programming*, pages 37–51. Springer, 2008. doi:10.1007/978-3-540-89982-2_8.

18 Kyle Marple, Elmer Salazar, and Gopal Gupta. Computing stable models of normal logic programs without grounding. *arXiv preprint arXiv:1709.00501*, 2017. arXiv:1709.00501.

19 Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Logic Programming And Nonmonotonic Reasoning: 4th International Conference, LPNMR'97 Dagstuhl Castle, Germany, July 28–31, 1997 Proceedings 4*, pages 420–429. Springer, 1997.

20 Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, CA, 1988.

21 Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning*, 10(2):1–189, 2016.

22 Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62:107–136, 2006. doi:10.1007/S10994-006-5833-1.

23 Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*. River Publishers, New York, 1 edition, September 2022. doi:10.1201/9781003338192.

24 Fabrizio Riguzzi and Terrance Swift. Well–definedness and efficient inference for probabilistic logic programming under the distribution semantics. *Theory and practice of logic programming*, 13(2):279–302, 2013. doi:10.1017/S1471068411000664.

25 Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *International Conference on Logic Programming*, 1995.

26 Helmut Thöne, Ulrich Güntzer, and Werner Kießling. Increased robustness of bayesian networks through probability intervals. *International Journal of Approximate Reasoning*, 17(1):37–76, 1997. doi:10.1016/S0888-613X(96)00138-7.

27 Allen Van Gelder, Kenneth A Ross, and John S Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649, 1991. doi:10.1145/116825.116838.

28 Victor Verreet, Vincent Derkinderen, Pedro Zuidberg Dos Martires, and Luc De Raedt. Inference and learning with model uncertainty in probabilistic logic programs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10060–10069, 2022. doi:10.1609/AAAI.V36I9.21245.