doi:10.1017/S147106840100xxxx

An Algebraic Approach to Weighted Answer Set Programming

FRANCISCO COELHO
NOVA-LINCS, University of Évora
(*e-mail:* fc@uevora.pt)

BRUNO DINIS
CIMA, University of Évora
(*e-mail:* bruno.dinis@uevora.pt)

DIETMAR SEIPEL
Universität Würzburg
(*e-mail:* dietmar.seipel@uni-wuerzburg.de)

SALVADOR ABREU
NOVA-LINCS, University of Évora
(*e-mail:* spa@uevora.pt)

Abstract

Logic programs, more specifically, answer set programs, can be annotated with probabilities on facts to express uncertainty. We address the problem of propagating the probabilities from the annotated facts of an answer set program to its stable models, and from there to events (sets of literals) 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. 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.

KEYWORDS: Answer-Set Programming, Stable Models, Probabilistic Logic Programming

1 Introduction

Using a logic program (LP) to model and reason over a real world scenario is often hard 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. Medical exams illustrate both problems: some parts of a living organism can't be directly observed *i.e.* a system with unreachable parts; instrumentation has limits and imperfections *i.e.* sensors that add noise to the real values.

We aim to explore how answer set programs (ASPs) plus weight annotated facts can lead to useful characterizations for this class of problems; We assume that knowledge about a *system* includes a formal *representation*[1] and empirical *data* such that (i) the *representation* is a logic program whose stable models (SMs) are the system states; (ii) *data* is a set of events; an *event* is a set of literals; (iii) the *weights* in the *annotated facts* are propagated to the stable models and, in general, to events.

In this setting, data 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?*'.

Systems such as `Problog` (De Raedt et al. 2007), `P-log` (Baral et al. 2009) or $\mathrm{LP}^{\mathrm{MLN}}$ (Lee and Wang 2016), in line with (Kifer and Subrahmanian 1992), derive a probability distribution of the stable models from the *syntax* of an annotated logic program. The more expressive of these systems, according to (Lee and Yang 2017), is $\mathrm{LP}^{\mathrm{MLN}}$, that can embed the Maximum A Posteriori (MAP) estimation of the other systems.

|[1] Since the probability distribution results from the program's syntax, these systems are limited to handle *a priori* information and are unable to support *posterior* data. Furthermore, these distributions are limited to the SMs, while we extend their domain to any event. However, the key feature that we aim to address is concerned with the inherent uncertainty of the SMs in logic programs such as $a:0.3, b \vee c \leftarrow a$. Intuitively,

[1] |addressing
It is not made sufficiently clear why we need another probabilistic formalism based on answer set programming, or why this particular formalisms should fill that niche.

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

this program entails three SMs: ac, ab and $\neg a$. We can assign the probability of $\neg a$ as 0.7 but what about ab and ac? Systems like $\mathrm{LP}^{\mathrm{MLN}}$ assign the same probability to these SMs (Lee and Yang 2017; Cozman and Mauá 2020).

We question the underlying assumption of such assignments and propose a method where the distribution that results from the representation includes parameters (*i.e.* variables in the algebraic sense: symbols for unknown quantities) that express the lack of information concerning cases as above. The values of these parameters can be estimated *a posteriori*, in the presence of data.

|[2] To frame this setting we assume that the atoms in the representation are associated to sensors of the system's states. More specifically, following (Calimeri et al. 2020), if a is a *predicate atom*, a state can activate either sensors a or $\neg a$, the *strong* or *classical* atoms, whereas no activation is represented by the (*default*, *weak* or *naf*) *literals* $\sim a$ and $\sim\neg a$. This redundancy is required to model hidden parts of a system as well as faulty sensors. For example, in

$$\{a, \neg a, \sim b, \sim\neg b, \sim c, \neg c\}$$

both a and $\neg a$ are activated (suggesting *e.g.* a fault in the relevant sensors), b is not observed (*i.e.* hidden), and $\sim c, \neg c$ reports the (consistent) activation of $\neg c$ and no activation of c. 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 this event can be shortened to the equivalent form $a\overline{a}\overline{c}$. Here we follow the convention, set in (Gelfond and Lifschitz 1988), of denoting a model by the set of true atoms, stressing that '*falsehood*' results only from the default negation *i.e.* $\sim a$ (*i.e.* '`not a`' in logic languages). More precisely, a model can contain atoms such as a or $\overline{b}$ but not literals $\sim a, \sim\overline{b}$.

Our choice to represent sensor input using both positive and negative literals is based on the following points: (i) it can be the case that there are two different sensors, a for the 'positive' states and $\overline{a}$ for the 'negative'; (ii) rules such as $a \vee \neg a$ represent a single sensor that always yields either a or $\overline{a}$; also, (iii) a closed-world assumption, where absence of input means classical negation, can be represented by $\overline{a} \leftarrow \sim a$.

|[3] |[4] Like in $\mathrm{LP}^{\mathrm{MLN}}$, we annotate facts (*i.e.* atoms) with weights (Lee and Yang 2017) instead of probabilities, that result from normalization of the former. We then use those weights to define a function on the stable models that is then extended to all the events in the program domain. The step from facts to SMs is non-deterministic in the sense that a given set of facts may entail zero, one or more SMs. As explained below, and also in (Verreet et al. 2022; Pajunen and Janhunen 2021; Cozman and Mauá 2020; Baral et al. 2009), this constitutes a problem when propagating weights to stable models: *How to distribute the weight of a fact to the entailed SMs?* We represent non-unique choices by a parameter that can be later estimated from further information, *i.e.* data. This approach enables later refinement and scoring of a partial program of a representation from additional evidence.

ASPs (Lifschitz 2002; 2008) are logic programs based on the stable model semantics of normal programs (NPs). ASPs represent a problem and the resulting models (*answer sets*) can be found using different approaches, including SAT solving technology (Gebser et al. 2011; Adrian et al. 2018; Niemelä and Simons 1997) or through top-down searching (Alberti et al. 2017; Arias et al. 2020; Marple et al. 2017).

The distribution semantics (DS) (Sato 1995; Riguzzi 2022) is a key approach to extend logical representations with probabilistic reasoning. We are particularly interested in two application scenarios of such an extension to logic programs:

1. Support probabilistic reasoning tasks on the program domain, *i.e.* the set of all events, $\mathcal{E}$.
2. Given a dataset and a divergence measure, a program can be scored (*e.g.* by the divergence w.r.t. the empiric distribution of the dataset), and sorted amongst other programs. These are key ingredients to construct an algorithm to determine, for example, optimal representations for a dataset.

The remainder of this article is structured as follows: the next section 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 distribution over total choices and address the issue of how to propagate these probabilities from facts to events, which is done in Section 4. This method relies on an equivalence relation on the set of events. Also, we express uncertainty by polynomial expressions over variables that add up to 1 and depend on the total choices and on the stable models. Some final remarks and ideas for future developments are presented in Section 5.

2 Framework

Selecting truth values for the annotated facts will lead a total choice (TC). To propagate probabilities from total choices to events we take the following stance:

[2]|addressing hidden values and default negation.

[3]|addressing better examples that show the real usefulness of this approach maybe a toy problem with a biased coin; Analyse the examples on the other systems and note that they are limited in form.

[4]|Normalize 'fact', 'atom'.

1. The *representation* is an answer set program whose stable models are the possible system states.
2. The *data* is a set of events; an *event* is a subset of the atoms of the representation.
3. The *weights* in the *annotated facts* are propagated to the stable models and, in general, to events.

In particular, some events may coincide with a stable model but others can be contained in, or contain, some SMs or neither case (but never both) and, thus, not uniquely determine the state of the system.

Propagating Weights. Our goal is to propagate weights from TCs to SMs and from there to any possible event. This propagation process soon faces a non-deterministic problem, illustrated by ex. 1 in section 3, where multiple SMs, ab and ac, result from a single TC, a, but *there is not enough information in the representation to assign a single weight to each SM.*

Algebraic variables[2] *describe the lack of information in a representation in order to deterministically propagate the weight to the stable models and events. The values of those variables is estimated from available data.*

The lack of unique stable model from a total choice is also addressed in (Cozman and Mauá 2020) along an approach using credal sets. In another related work (Verreet et al. 2022), epistemic uncertainty (or model uncertainty) is considered as a lack of knowledge about the underlying model, that may be mitigated via further observations. This seems to presuppose a bayesian approach to imperfect knowledge in the sense that having further observations allows one to improve or correct the model. Indeed, that approach uses Beta distributions on the total choices in order to be able to learn a distribution on the events. This approach seems to be specially fitted to being able to tell when some weight lies beneath some given value. Our approach appears as similar in spirit, while remaining algebraic in the way that the propagation of weights is addressed.

3 Syntax and Semantics of Weighted ASP

We start 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 '$\neg x$' denotes classical negation and '$\sim x$' default negation.

Syntax. We slightly adapt (Calimeri et al. 2020). Let $\mathcal{A}$ be a finite set of symbols, or *positive atoms*. For $a \in \mathcal{A}$, the expressions a and $\neg a$ (also denoted $\overline{a}$) are *(classical) atoms*; If b is an atom, the expressions l and $\sim l$ are *(naf-)literals*. A *rule* is of the form

$$a_1 \vee \cdots \vee a_n \leftarrow l_1 \wedge \cdots \wedge l_m$$

where the a_i are atoms and the l_j are literals. The symbol '$\leftarrow$' separates the *head* from the *body*. The rule is a *fact* and if $n = 0$, *normal* if $n = 1$, and *disjunctive* if $n > 1$.

An *answer set program (ASP)* is a set P of facts and (both normal and disjunctive) 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 a disjunctive rule can be converted into a set of normal rules (Gebser et al. 2022).

Semantics. The standard semantics of an ASP has a few different, but equivalent, definitions (Lifschitz 2008). A common definition is as follows (Gelfond and Lifschitz 1988): 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 $\mathcal{M}(P)$, or simply $\mathcal{M}$, the set of stable models of the program P.

Evaluation without Grounding. A different approach in handling the generation of stable models is the one supported by `s(CASP)`, a system that can evaluate ASP programs with function symbols (functors) and constraints without grounding them either before or during execution, using a method similar to SLD resolution (Marple et al. 2017; Arias et al. 2020). |[5] This enables the generation of human readable explanations of the results of programs and addresses two major issues of grounding-based solvers, that (i) either do not support function symbols or, using finite domains, lead to exponential groundings of a program and (ii) compute the complete model of the grounded program when, in some scenarios, it is desirable to compute only a partial stable model containing a query.

[5]|Improve grounding and propositional cases.

[2] We explicitly write 'algebraic variables' to avoid confusion with logic variables.

WASPs and their Derived Programs

Weighted answer set programs (WASPs) extend ASPs by adding facts with weight annotations: A *weighted fact (WF)* is of the form $a:w$ where a is an atom and $w \in [0,1]$. We denote the set of weighted facts of a program by $\mathcal{W}$, and $\mathcal{A}_{\mathcal{W}}$ the set of positive atoms in $\mathcal{W}$.

Our definition of WASPs is restricted because our goal is to illustrate the core of a method to propagate weights from total choices to events. Our programs do not feature variables, relation symbols, functors or other elements common in standard ASP. Also, weight annotations are not associated to (general) clause heads or disjunctions. However, these last two restrictions do not reduce the expressive capacity of the language because, for the former, a clause with an annotated head can be rewritten as:

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

while annotated disjunctive facts

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

Derived Program. The *derived program* of a WASP is obtained by replacing each weighted fact $a:w$ by a disjunction $a \vee \overline{a}$. The *stable models* of an WASP program are the stable models of its derived program. The set of SMs of a (derived or) WASP program P is (also) denoted $\mathcal{M}(P)$ or $\mathcal{M}$.

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 $\mathcal{E}$ and. An event $e \in \mathcal{E}$ which includes a set $\{x, \overline{x}\} \subseteq e$ is said to be *inconsistent*; otherwise it is *consistent*. The set of consistent events is denoted by $\mathcal{C}$.

Example 1 (*Fruitful WASP*)
Consider the following weighted answer set program :

$$P_{\text{fr}} = \begin{cases} a:0.3, \\ b \vee c \leftarrow a \end{cases} \tag{1}$$

which has the set $\mathcal{W} = \{a : 0.3\}$ *of weighted facts. This program is transformed into the logic program*

$$P'_{\text{fr}} = \begin{cases} a \vee \overline{a}, \\ b \vee c \leftarrow a, \end{cases} \tag{2}$$

with the set $\mathcal{M} = \{\, \overline{a}, ab, ac \,\}$ *of three stable models.*

The atoms of these programs are

$$\mathcal{A} = \left\{ a, \overline{a}, b, \overline{b}, c, \overline{c} \right\} \tag{3}$$

and the events are

$$\mathcal{E} = \mathbb{P}(\mathcal{A})\,. \tag{4}$$

Total Choices and their Weights

A disjunctive head $a \vee \overline{a}$ in the derived program represents a single *choice*, either a or $\overline{a}$[3]. A *total choice* of the derived program, and of the WASP program, is $t = \{a' \mid a:p \in \mathcal{W}\}$ where each a' is either a or $\overline{a}$. We denote by $\mathcal{T}$ the set of total choices of a WASP or of its derived program.

The *weight of the total choice* $t \in \mathcal{T}$ is given by the product

$$\omega_{\mathcal{T}}(t) = \prod_{\substack{a:w \,\in\, \mathcal{W}, \\ a \,\in\, t}} w \;\times\; \prod_{\substack{a:w \,\in\, \mathcal{W}, \\ \overline{a} \,\in\, t}} \overline{w}. \tag{5}$$

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 $\mathcal{M}, \mathcal{E}$ to deal with functions of stable models and events, $\omega_{\mathcal{M}}, \omega_{\mathcal{E}}$.

Notice that in $a:w$ we have $w \in [0,1]$ but w is not interpreted as a probability but, instead, as a *balance* between the *choices* a and $\overline{a}$.

Some stable models are entailed from some total choices while other SMs are entailed by other TCs. We write $\mathcal{M}(t)$ to represent the set of stable models entailed by the total choice $t \in \mathcal{T}$.

[3] We use the term 'choice' for historical reasons, *e.g.* in (Cozman and Mauá 2020), but remark that it is not related to the usual 'choice' elements, atoms or rules from (Calimeri et al. 2020).

Our goal can now be rephrased as to know how to propagate the weights of the program's total choices, $\omega_{\mathcal{T}}$, in eq. (5) to the program's events, $\omega_{\mathcal{E}}$.

Propagation of Weights. As a first step to propagate weight from total choices to events, consider the P_{fr} program of eq. (1) and a possible propagation of $\omega_{\mathcal{T}} : \mathcal{T} \to [0,1]$ from total choices to the stable models, $\omega_{\mathcal{M}} : \mathcal{M} \to [0,1]$. It might seem straightforward, in ex. 1, to assume $\omega_{\mathcal{M}}(\overline{a}) = 0.7$ but there is no explicit way to assign values to $\omega_{\mathcal{M}}(ab)$ and $\omega_{\mathcal{M}}(ac)$. We represent this non-determinist by a parameter θ as in

$$\begin{aligned} \omega_{\mathcal{M}}(ab) &= 0.3\,\theta, \\ \omega_{\mathcal{M}}(ac) &= 0.3\,(1-\theta) \end{aligned} \tag{6}$$

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 eq. (6), $s = ab$ in the first line and $s = ac$ in the second line) and a total choice t (above $t = a$), 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 $\mathrm{LP}^{\mathrm{MLN}}$ (Lee and Wang 2016), the value 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 empirical distributions from datasets. Further discussion of this is outside the scope of this paper.

Now consider the program

$$\begin{cases} a : 0.3, \\ b \leftarrow a \wedge \sim b \end{cases} \tag{7}$$

that has a single SM, $\overline{a}$. Since the weights are not interpreted as probabilities, there is no need to have their sum equal to 1. So the weights in the TCs of eq. (7) only set

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

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

$$a \qquad \Longrightarrow \qquad a : 1.0 \tag{8}$$

The method that we are proposing does not follow the framework of (Kifer and Subrahmanian 1992) 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, and from there we propagate weights to the programs's events and then normalization provides the final probabilities.

[6]

[6] STOPED HERE.

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 modes (*e.g.* to possible worlds in `Problog`) has been studied for some time (Kifer and Subrahmanian 1992; Sato 1995).

For example, the *credal set* approach of (Cozman and Mauá 2020), defines $P_{\mathcal{T}}$ in a way similar to eq. (5) but then, for $a \in \mathcal{A}$, the probability $P(a \mid t)$ is unknown but bounded by $\underline{P}(a \mid t)$ and $\overline{P}(a \mid t)$, that can be explicitly estimated from the program.

`Problog` (Fierens et al. 2015; Verreet et al. 2022) 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 (Riguzzi and Swift 2013) *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 (Riguzzi and Swift 2013; Van Gelder et al. 1991).

Another system, based on Markov Logic (Richardson and Domingos 2006), is $\mathrm{LP}^{\mathrm{MLN}}$ (Lee and Wang 2016; Lee and Yang 2017), whose models result from *weighted rules* of the form $a \leftarrow b, n$ 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.

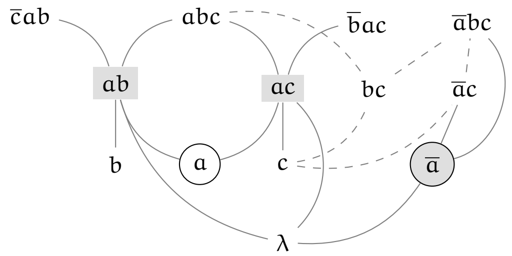

Fig. 1. This (partial sub-/super-set) diagram shows some events related to the stable models of the program eq. (1). The circle nodes are total choices and shaded nodes are stable models. Solid lines represent relations with the stable models and dashed lines some sub-/super-set relations with other events. The set of events contained in all stable models, denoted by $\bigwedge$, is $\{\lambda\}$ in this example, because $\overline{a} \cap ab \cap ac = \emptyset = \lambda$.

Towards Propagating Weights from Literals to Events.

The program in eq. (1) from ex. 1 exemplifies 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 a single weight to each stable model. This becomes a crucial problem in situations where multiple stable models result from a single total choice.

Our stance is that an ASP program describes an observable system, the program's stable models are the possible states of that system and that state events are stochastic and detected in the form of events. Then:

1. With a weight set for the stable models, we extend it to any event in the program domain, *i.e.* to any set of literals present in the program.
2. In the case where some statistical knowledge is available, for example, in the form of a distribution relating some literals, we consider it as 'external' knowledge about the parameters, that doesn't affect the propagation procedure described below.
3. That knowledge can be used to estimate the parameters $\theta_{s,t}$ and to 'score' the program.
4. If a program is but one of many possible candidates then that score can be used, *e.g.* as fitness, by algorithms searching (optimal) programs of a dataset of events.
5. If events are not consistent with the program, then we ought to conclude that the program is wrong and must be changed accordingly.

Currently, we are addressing the problem of propagating a measure (in the *measure theory* sense), possibly using formal parameters such as θ, defined on the stable models of a program, $\mu_{\mathcal{M}} : \mathcal{M} \to \mathbb{R}$, to all the events of that program: $\mu_{\mathcal{E}} : \mathcal{E} \to \mathbb{R}$. Denoting the *power set* of X by $\mathbb{P}(X)$, the latter function will then be normalized and extended into a weight $\omega_{\mathcal{E}} : \mathbb{P}(\mathcal{E}) \to [0, 1]$. This way probabilistic reasoning is consistent with the ASP program and follows our interpretation of stable models as the states of an observable system.

4 Propagating Weights

The diagram in fig. 1 illustrates the problem of propagating weights from total choices to stable models and then to general events in an *edge-wise* process, *i.e.* where the value in a node is defined from the values in its neighbors. This quickly leads to coherence problems concerning weight, with no clear systematic approach. For example, notice that bc is not directly related with any stable model. Propagating values through edges would assign a value ($\neq 0$) to bc hard to explain in terms of the semantics of the program. Instead, we propose to settle such propagation on the relation an event has with the stable models.

4.1 An Equivalence Relation

Our path to propagate probabilities starts with the perspective that stable models play a role similar to *prime factors* or *principal ideals*. The stable models of a program are the irreducible events entailed from that program and any event must be considered under its relation with the stable models.

From ex. 1 and fig. 2 consider the stable models $\overline{a}$, ab, ac and events a, abc and c. While a is related with (i.e. contained in) both ab, ac, the event c is related only with ac. So, a and c are related with different stable models. On the other hand, abc contains both ab, ac. So a and abc are related with the same stable models.

The *stable core (SC)* of the event $e \in \mathcal{E}$ is

$$[\![e]\!] := \{\, s \in \mathcal{M} \mid s \subseteq e \vee e \subseteq s \,\} \tag{9}$$

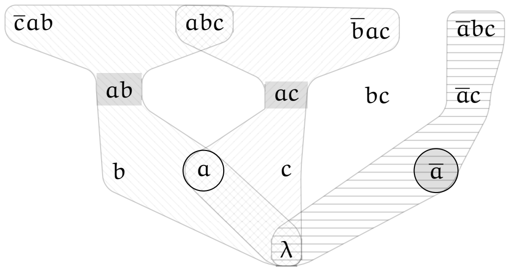

Fig. 2. Classes of (consistent) events related to the stable models of ex. 1 are defined through sub-/super-set relations. In this picture we can see, for example, that $\{\overline{c}ab, ab, b\}$ and $\{a, abc\}$ are part of different classes, represented by different fillings. As before, the circle nodes are total choices and shaded nodes are stable models. Notice that bc is not in a filled area.

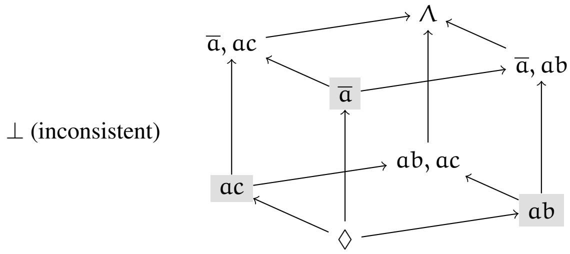

Fig. 3. Lattice of the stable cores from ex. 1. In this diagram the nodes are the different stable cores that result from the stable models, plus the *inconsistent* class ($\perp$). The bottom node ($\Diamond$) is the class of *independent* events, those that have no sub-/super-set relation with the SMs and the top node (Λ) represents events related with all the SMs *i.e.* the *consequences* of the program. As in previous diagrams, shaded nodes represent the SMs.

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

Observe 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 1 (*Equivalence Relation on Events.*)
For a given program, let $u, v \in \mathcal{E}$. The equivalence relation $\sim$ is defined by

$$u \sim v \;:\leftarrow\; u, v \notin \mathcal{C} \vee \left(u, v \in \mathcal{C} \wedge [\![u]\!] = [\![v]\!]\right). \tag{10}$$

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:

$$[e]_\sim = \begin{cases} \perp := \mathcal{E} \setminus \mathcal{C} & \text{if } e \in \mathcal{E} \setminus \mathcal{C}, \\ \left\{u \in \mathcal{C} \,\middle|\, [\![u]\!] = [\![e]\!]\right\} & \text{if } e \in \mathcal{C}. \end{cases} \tag{11}$$

Proposition 1 (*Class of the Program's Consequences*)
Let λ be the event empty set (because $\emptyset \in \mathcal{E}$), and Λ the *consequence class* of events related with all the stable models. Then

$$[\lambda]_\sim = [\![\mathcal{M}]\!] = \Lambda. \tag{12}$$

The combinations of stable models, *i.e.* the stable cores, together with the set of inconsistent events ($\perp$) forms a set of representatives for the equivalence relation $\sim$. Since all events within an equivalence class have the same relation with a specific stable core, we are interested in functions (including weight distributions), that are constant within classes. A function $f : \mathcal{E} \to Y$, where Y is any set, is said to be *coherent* if

$$\forall u \in [e]_\sim \left(f(u) = f(e)\right). \tag{13}$$

Considering coherent functions, in the specific case of eq. (1), 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.

4.2 From Total Choices to Events

Our path to set a distribution on $\mathcal{E}$ continues with the more general problem of extending *measures*, since propagating *probabilities* easily follows by means of a suitable normalization (done in eqs. (21b) and (23)), and has two phases: (1) Propagation of the probabilities, *as measures*, from the total choices to events and (2) Normalization of the measures on events, recovering a weight.

The "propagation" phase, traced by eq. (5) and eqs. (15) to (21b), starts with the weight (as a measure) of total choices, $\mu_{\mathcal{T}}(t) = \omega_{\mathcal{T}}(t)$, propagates it to the stable models, $\mu_{\mathcal{M}}(s)$, and then, within the equivalence relation from eq. (10), to a coherent measure of events, $\mu_{\mathcal{E}}(e)$, including (consistent) worlds. So we are specifying a sequence of functions

$$\mu_{\mathcal{T}}, \mu_{\mathcal{M}}, \mu_{\mathcal{C}}, \mu_{\mathcal{E}} \tag{14}$$

on successive larger domains $\mathcal{T}, \mathcal{M}, [\mathcal{E}]_{\sim}, \mathcal{E}$ so that the last function ($\mu_{\mathcal{E}}$) is a finite coherent measure on the set of events and thus, as a final step, it can easily be used to define a weight distribution of events by normalization: $\mu_{\mathcal{E}} \longrightarrow \omega_{\mathcal{E}}$.

Total choices and Stable models

Let's start by looking into the first two steps of the sequence of functions eq. (14): $\mu_{\mathcal{T}}$ and $\mu_{\mathcal{M}}$. Using eq. (5), the measure $\mu_{\mathcal{T}}$ of the total choice $t \in \mathcal{T}$ is given by

$$\mu_{\mathcal{T}}(t) := \omega_{\mathcal{T}}(t) = \prod_{\substack{\alpha p \,\in\, \mathcal{W},\\ \alpha \,\in\, t}} p \;\times \prod_{\substack{\alpha p \,\in\, \mathcal{W},\\ \overline{\alpha} \,\in\, t}} \overline{p}. \tag{15}$$

Recall that each total choice $t \in \mathcal{T}$, together with the rules and the other facts of a program, defines the set $\mathcal{M}(t)$ of stable models associated with that choice. Given a total choice $t \in \mathcal{T}$, a stable model $s \in \mathcal{M}$, and formal variables or values $\theta_{s,t} \in [0, 1]$ such that $\sum_{s \in \mathcal{M}(t)} \theta_{s,t} = 1$, we define

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

The $\theta_{s,t}$ parameters in eq. (16) express the *program's* lack of information about the measure assignment, when a single total choice entails more than one stable model. We propose to address this issue by assigning a possibly unknown parameter, *i.e.* a formal 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 but might be determined or estimated given more information, *e.g.* observed data.

As sets, the stable models can have non-empty intersection. But because different SMs represent different states of a system, we assume that the algebra of the stable models is σ-additive:

Assumption 1 (*Stable models as Disjoint Events.*)
For any set X of stable models and any total choice t,

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

Equation (17) is the basis for eq. (19a) and effectively extends $\mu_{\mathcal{M}} : \mathcal{M} \to \mathbb{R}$ to $\mu_{\mathcal{M}} : \mathbb{P}(\mathcal{M}) \to \mathbb{R}$. Notice that the pre-condition of eq. (16) can now be stated as $\mu_{\mathcal{M}}\big(\mathcal{M}(t)\,, t\big) = 1$.

Classes

Consider the next function in sequence eq. (14), $\mu_{\mathcal{C}}$ on $[\mathcal{E}]_{\sim}$. Each class of the equivalence relation $\sim$ (eq. 10) is either the inconsistent class ($\bot$) or is associated with a stable core, *i.e.* a set of stable models. Therefore, $\mu_{\mathcal{C}}$ is defined considering the following two cases:

Inconsistent class. This class contains events that are logically inconsistent, thus should never be observed and thus have measure zero:

$$\mu_{\mathcal{C}}(\bot, t) := 0.^{4} \tag{18}$$

[4] This measure being zero is independent of the total choice.

Consistent classes. For the propagation function to be coherent, it must be constant within a class and its value dependent only on the stable core:

$$\mu_{\mathcal{C}}\left([e]_{\sim}, t\right) := \mu_{\mathcal{M}}\left(\llbracket e \rrbracket, t\right) = \sum_{s \in \llbracket e \rrbracket} \mu_{\mathcal{M}}(s, t)\,. \tag{19a}$$

and we further define the following:

$$\mu_{\mathcal{C}}\left([e]_{\sim}\right) := \sum_{t \in \mathcal{T}} \mu_{\mathcal{T}}(t)\, \mu_{\mathcal{C}}\left([e]_{\sim}, t\right) \tag{19b}$$

Equation (19a) states that the measure of a class $[e]_{\sim}$ is the measure of its stable core ($\llbracket e \rrbracket$) and eq. (19b) *averages* eq. (19a) over the total choices.

Notice that eq. (19a) also applies to the independent class, $\Diamond$, because events in this class are not related with any stable model. For such an event e, $\llbracket e \rrbracket = \emptyset$ so

$$\mu_{\mathcal{C}}(\Diamond, t) = \sum_{s \in \emptyset} \mu_{\mathcal{M}}(s, t) = 0. \tag{20}$$

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:

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

and, by averaging over the total choices:

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

In order to get a weight from eq. (21b), we need a *normalizing factor*:

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

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

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

Equation (23) defines a coherent *prior*[5] weight of events and, together with external statistical knowledge, can be used to learn about the *initial* probabilities of the literals, that should not (and by prop. 3 can't) be confused with the explicit $\mu_{\mathcal{T}}$ set in the program.

Now $\omega_{\mathcal{E}} : \mathcal{E} \to [0, 1]$ can be extended to $\omega_{\mathcal{E}} : \mathbb{P}(\mathcal{E}) \to [0, 1]$ by abusing notation and setting, for $X \subseteq \mathcal{E}$,

$$\omega_{\mathcal{E}}(X) = \sum_{x \in X} \omega_{\mathcal{E}}(x)\,. \tag{24}$$

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

We can now properly state the following property about *certain facts* such as $a : 1.0$.

Proposition 2 (*Probability of Certain Facts.*)
Consider a program A with the probabilistic fact $a : 1.0$ and A' where it is replaced by the deterministic fact a. Let $\omega_{\mathcal{E}}$ be as eq. (23) for A and $\omega'_{\mathcal{E}}$ for A'. Then

$$\forall e \in \mathcal{E} \left(\omega_{\mathcal{E}}(e) = \omega'_{\mathcal{E}}(e)\right). \tag{25}$$

Since total choices are also events, one can ask, for an arbitrary total choice t, if $\omega_{\mathcal{T}}(t) = \omega_{\mathcal{E}}(t)$ or, equivalently, if $\mu_{\mathcal{T}}(t) = \mu_{\mathcal{E}}(t)$. However, it is easy to see that, in general, this cannot be true. While the domain of $\omega_{\mathcal{T}}$ is the set of total choices, for $\omega_{\mathcal{E}}$ the domain is much larger, including all the events. Except for trivial programs, where the SMs are the TCs, some events other than total choices will have non-zero weight.

Proposition 3
If a program has a stable model that is not a total choice then there is at least one $t \in \mathcal{T}$ such that

$$\omega_{\mathcal{T}}(t) \neq \omega_{\mathcal{E}}(t)\,. \tag{26}$$

[5] In the Bayesian sense that future observations might update this weight.

Proof
Suppose towards a contradiction that $\omega_{\mathcal{T}}(t) = \omega_{\mathcal{E}}(t)$ for all $t \in \mathcal{T}$. Then

$$\sum_{t \in \mathcal{T}} \omega_{\mathcal{E}}(t) = \sum_{t \in \mathcal{T}} \omega_{\mathcal{T}}(t) = 1.$$

Hence $\omega_{\mathcal{E}}(x) = 0$ for all $x \in \mathcal{E} \setminus \mathcal{T}$, in contradiction with the fact that at least for one $s \in \mathcal{M} \setminus \mathcal{T}$ it must be $\omega_{\mathcal{E}}(s) > 0$. $\square$

The essential, *counter-intuitive*, conclusion of prop. 3 is that we are dealing with *two distributions*: one, restricted to the total choices, is explicit in the annotations of the programs while the other, covering all the events, results from the explicit annotations of the program *and the structure of the stable models*. For example:

$$\begin{aligned} \omega_{\mathcal{T}}(a) &= 0.3 && \text{from the program ex. 1,} \\ \omega_{\mathcal{E}}(a) &= \frac{3}{64} && \text{from eq. (27).} \end{aligned}$$

Example 2 (*Probability of Events*)
In summary, for ex. 1, the coherent prior *weight of events of program eq.* (1) *is*

$$\begin{array}{c|ccccccccc} [\![e]\!] & \bot & \Diamond & \overline{a} & ab & ac & \overline{a}, ab & \overline{a}, ac & ab, ac & \Lambda \\ \hline \omega_{\mathcal{E}}(e) & 0 & 0 & \frac{7}{207} & \frac{1}{23}\theta & \frac{1}{23}\overline{\theta} & 0 & 0 & \frac{3}{46} & \frac{10}{23} \end{array} \tag{27}$$

We can use this table to compute the weight of any single event $e \in \mathcal{E}$ *by looking at the column of the event's stable core. For example:*

$\omega_{\mathcal{E}}(ab) = \frac{\theta}{23}$, *because* ab *is the only* SM *related with* ab *so* $[\![ab]\!] = \{ab\}$ *and the weight value is found in the respective column of eq.* (27).
$\omega_{\mathcal{E}}(abc) = \frac{3}{46}$ *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. $bc \in \Diamond$.
$\omega_{\mathcal{E}}(\overline{a}b) = \frac{7}{207}$ *because* $[\![\overline{a}b]\!] = \{\overline{a}\}$.
$\omega_{\mathcal{E}}(\overline{a}) = \frac{7}{207}$ *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. *prop. 3*), *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 the last case above, consider the complement of a consistent event e, *denoted by* $\mathcal{C}e$. *To calculate* $\omega_{\mathcal{E}}(\mathcal{C}e)$ *we look for the classes in* $[\mathcal{E}]_{\sim}$ *that are not* $[e]_{\sim}$, i.e. *the complement of* e*'s class within* $[\mathcal{E}]_{\sim}$[6], $\mathcal{C}[e]_{\sim}$. *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 $\omega_{\mathcal{E}}(\mathcal{C}a)$, *since* $[\![a]\!] = \{ab, ac\}$ *then* $\mathcal{C}[a]_{\sim} = [\mathcal{E}]_{\sim} \setminus [a]_{\sim}$ *and* $\omega_{\mathcal{E}}(\mathcal{C}a) = \omega_{\mathcal{E}}\left([\mathcal{E}]_{\sim} \setminus [a]_{\sim}\right) = 1 - \omega_{\mathcal{E}}(a)$. *Also,* $\omega_{\mathcal{E}}(\mathcal{C}\overline{a}) = 1 - \omega_{\mathcal{E}}(\overline{a})$.

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

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

Our approach generalizes to Bayesian networks in a way similar to (Cozman and Mauá 2020; Raedt et al. 2016) and (Kießling et al. 1992; Thöne et al. 1997) as follows. On the one hand, any acyclic propositional program can be viewed as the specification of a Bayesian network over binary random variables. So, we may take the structure of the Bayesian network to be the dependency graph. The random variables then correspond to the atoms and the probabilities can be read off of the probabilistic facts and rules. Conversely, any Bayesian network over binary variables can be specified by an acyclic non-disjunctive WASP.

5 Discussion and Future Work

This work is a first venture into expressing weight distributions using algebraic expressions derived from a logical program, in particular an ASP. We would like to point out that there is still much to explore

[6] All the usual set operations hold on the complement. For example, $\mathcal{C}\mathcal{C}X = X$.

concerning the full expressive power of logic programs and ASP programs. So far, we have not considered recursion, logical variables or functional symbols. Also, there is still little effort to articulate with the related fields of probabilistic logical programming, machine learning, inductive programming, *etc.*

The equivalence relation from definition 1 identifies the $s \subseteq e$ and $e \subseteq s$ cases. Relations that distinguish such cases might enable better relations between the representations and processes from the stable models.

The theory, methodology, and tools, from Bayesian Networks can be adapted to our approach. The connection with Markov Fields (Kindermann and Snell 1980) is left for future work. An example of a "program selection" application (as mentioned in item 4, section 3) is also left for future work.

We decided to set the measure of inconsistent events to 0 but, maybe, in some cases, we shouldn't. For example, since observations may be affected by noise, one can expect inconsistencies between the literals of an event to occur.

Acknowledgements

This work is partly supported by Fundação para a Ciência e Tecnologia (FCT/IP) under contracts UIDB/04516/2020 (NOVA LINCS), UIDP/04674/2020 and UIDB/04675/2020 (CIMA).

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.

References

Adrian, W. T., Alviano, M., Calimeri, F., Cuteri, B., Dodaro, C., Faber, W., Fuscà, D., Leone, N., Manna, M., Perri, S. et al.: 2018, The asp system dlv: advancements and applications, *KI-Künstliche Intelligenz* **32**, 177–179.

Alberti, M., Bellodi, E., Cota, G., Riguzzi, F. and Zese, R.: 2017, cplint on swish: Probabilistic logical inference with a web browser, *Intelligenza Artificiale* **11**(1), 47–64.

Arias, J., Carro, M., Chen, Z. and Gupta, G.: 2020, Justifications for goal-directed constraint answer set programming, *arXiv preprint arXiv:2009.10238* .

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

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T.: 2020, Asp-core-2 input language format, *Theory and Practice of Logic Programming* **20**(2), 294–309.

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.

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

Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G. and De Raedt, L.: 2015, Inference and learning in probabilistic logic programs using weighted boolean formulas, *Theory and Practice of Logic Programming* **15**(3), 358–401.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T.: 2022, *Answer set solving in practice*, Springer Nature.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T. and Schneider, M.: 2011, Potassco: The Potsdam answer set solving collection, *AI Communications* **24**(2), 107–124.

Gelfond, M. and Lifschitz, V.: 1988, The stable model semantics for logic programming., *ICLP/SLP*, Vol. 88, Cambridge, MA, pp. 1070–1080.

Kießling, W., Thöne, H. and Güntzer, U.: 1992, Database support for problematic knowledge, *Advances in Database Technology - EDBT'92: 3rd International Conference on Extending Database Technology Vienna, Austria, March 23–27, 1992 Proceedings 3*, Springer, pp. 421–436.

Kifer, M. and Subrahmanian, V.: 1992, Theory of generalized annotated logic programming and its applications, *The Journal of Logic Programming* **12**(4), 335–367.

Kindermann, R. and Snell, J. L.: 1980, *Markov random fields and their applications*, Vol. 1 of *Contemporary Mathematics*, American Mathematical Society, Providence, RI.

Lee, J. and Wang, Y.: 2016, Weighted rules under the stable model semantics, *Fifteenth international conference on the principles of knowledge representation and reasoning*.

Lee, J. and Yang, Z.: 2017, Lpmln, Weak Constraints, and P-log, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

Lifschitz, V.: 2002, Answer set programming and plan generation, *Artificial Intelligence* **138**(1), 39–54.

Lifschitz, V.: 2008, Twelve definitions of a stable model, *International Conference on Logic Programming*, Springer, pp. 37–51.

Marple, K., Salazar, E. and Gupta, G.: 2017, Computing stable models of normal logic programs without grounding, *arXiv preprint arXiv:1709.00501* .

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

Pajunen, J. and Janhunen, T.: 2021, Solution enumeration by optimality in Answer Set Programming, *Theory and Practice of Logic Programming* **21**(6), 750–767.

Raedt, L. D., Kersting, K., Natarajan, S. and Poole, D.: 2016, Statistical relational artificial intelligence: Logic, probability, and computation, *Synthesis lectures on artificial intelligence and machine learning* **10**(2), 1–189.

Richardson, M. and Domingos, P.: 2006, Markov logic networks, *Machine learning* **62**, 107–136.

Riguzzi, F.: 2022, *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*, 1 edn, River Publishers, New York.

Riguzzi, F. and Swift, T.: 2013, Well–definedness and efficient inference for probabilistic logic programming under the distribution semantics, *Theory and practice of logic programming* **13**(2), 279–302.

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

Thöne, H., Güntzer, U. and Kießling, W.: 1997, Increased robustness of bayesian networks through probability intervals, *International Journal of Approximate Reasoning* **17**(1), 37–76.

Van Gelder, A., Ross, K. A. and Schlipf, J. S.: 1991, The well-founded semantics for general logic programs, *Journal of the ACM (JACM)* **38**(3), 619–649.

Verreet, V., Derkinderen, V., Dos Martires, P. Z. and De Raedt, L.: 2022, Inference and learning with model uncertainty in probabilistic logic programs, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36, pp. 10060–10069.