An Algebraic Approach to Weighted Answer Set Programming

Francisco Coelho Bruno Dinis Dietmar Seipel
Salvador Abreu

October 26, 2024

Abstract

We address the problem of propagating the probabilities from the more, more! annotated facts of an answer set program to its stable models, and from there to general events in the program's domain. Our approach is algebraic in the sense that it relies on an equivalence relation over the set of events, and uncertainty is expressed with variables and polynomial expressions. We propagate probability in the space of models and events, instead of dealing with the syntax of the program. We illustrate our methods with two examples, one of which shows a connection to bayesian networks.

1 Introduction and Motivation

A major limitation of logical representations in real world applications is the implicit assumption that the background knowledge is perfect. This assumption is problematic if data is affected by stochastic factors, which is often the case. Here we aim to explore how answer set programs with probabilistic facts can lead to characterizations of probability distributions on the program's domain.

Current systems such as `Problog` [7], `P-log` [4] or $\mathtt{LP}^{\mathtt{MLN}}$[17], in line with [15], derive a probability distribution from the *syntax* of a program. Instead, we propose a method where such distributions result from the program's *stable models (SMs)* and probability-annotated facts. We use those facts to define a function on the SMs and then propagate that function to a probability distribution of the events in the program domain.

The step from facts to stable models is non-deterministic in the sense that a given set of facts entails zero, one or more SMs. As explained below, and also in [33, 23, 6, 4], this poses a problem when propagating probabilities from annotated facts to stable models. We represent some non-unique choices by a parameter that can be later estimated from further information, *e.g.* observations. This approach enables later refinement and scoring of a partial program of a model from additional evidence.

Answer set programs (ASPs) [19, 20] 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 either using SAT solving technology [10, 1, 22] or through top-down searching [2, 3, 21]. Unlike `Prolog`, ASP is a truly declarative language that supports language constructs such as disjunction in the head of a rule, choice rules, and both hard and weak constraints. The distribution semantics (DS) [29, 27] is a key approach to extend logical representations with probabilistic reasoning. We can foresee two key applications of this extension of 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, the program can be scored (*e.g.* by the divergence w.r.t. the *empiric* distribution of the dataset), and weighted or sorted amongst other programs. These are key ingredients in algorithms searching, for example, optimal models of a dataset.

To propagate probabilities from total choices to events we start with the following stance:

- A program describes an observable system.
- The stable models are the possible states of that system.
- Observations (*i.e.* events) are stochastic.

In particular, one observation can be sub-complete (a proper subset of a SM) or super-complete (a proper superset of a SM), and might not uniquely determine the real state of the system. From here, probabilities must be extended from total choices (TCs) to SMs and then to any event. This propagation process soon faces a non-deterministic problem, illustrated by the example in section 2, where multiple SMs, ab and ac, result from a single TC, a, but there is not enough explicit information in the program to assign a single probability to each SM. We propose to use algebraic variables to describe that lack of information and then estimate the value of those variables from empirical data.

The lack of unique SMs is also addressed in [6] along a different approach, using credal sets. In another related work, [33] 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 to improve/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 probability lies beneath some given value. Our approach seems to be similar in spirit, while remaining algebraic in the way that the propagation of probabilities is addressed.

The example in section 2 uses the code available in the project's repository[1], developed with the `Julia` programming language [5], and the `Symbolics` [13] libraries.

2 Syntax and Semantics of Stochastic ASP

We start this overview with the syntax and ASP semantics of (propositional) normal programs and then proceed to the case of probabilistic annotations. We setup a minimal syntax, enough to illustrate our method to propagate probability from annotated facts to general events, without variables, functors or relation symbols.

[1] https://git.xdi.uevora.pt/fc/sasp

Syntax. Let $\mathcal{A}$ be a finite set. A *positive literal*, or *atom*, is any $a \in \mathcal{A}$ and a *negative literal* is $\neg a$, also denoted $\overline{a}$, for $a \in \mathcal{A}$. A *literal* is a positive or a negative literal. A *fact* is a literal. For literals $l_1, \ldots, l_n$, a *conjunction* is $l_1 \wedge \cdots \wedge l_n$, and a *disjunction* is $l_1 \vee \cdots \vee l_n$. A *choice rule* is of the form $\alpha \leftarrow \beta$, where α is a disjunction, the *head*, and β a conjunction, the *body*. In a *normal rule* the head is a single literal. An *answer set program (ASP)* is a set of facts and (both normal and choice) rules, denoted, resp. $\mathcal{F}(P)$ and $\mathcal{R}(P)$, or simply $\mathcal{F}$ and $\mathcal{R}$. Notice that a choice rule can be converted to a set of normal rules [9].

Semantics. The standard semantics of an ASP can have different, equivalent, definitions [20]. A common definition is as follows [11]. Let P be a normal program. check the reference The Gelfond/Lifschitz *reduct* of P relative to the set X of atoms results from (i) deleting rules that contain a literal $\neg l$ in the body with $l \in X$ and then (ii) deleting remaining literals $\neg l'$ 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 $\mathcal{M}(P)$, or simply $\mathcal{M}$, the set of stable models of the program P.

Evaluation without Grounding. A promising approach to handle 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 [21, 3]. This enables the generation of human readable explanations of the results of programs and addresses two major issues of grounding-based solvers, that (1) either do not support function symbols or, using finite domains, lead to exponential groundings of a program and (2) 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.

SASPs and their Derived Programs

Stochastic Answer Set Programs (SASPs) extend ASP by adding facts with probabilistic annotations: A *probabilistic fact (PF)* is $a : p$ where a is an atom and $p \in [0, 1]$. We denote the set of probabilistic facts of a program by $\mathcal{P}$, and $\mathcal{A}_{\mathcal{P}}$ the set of positive literals in $\mathcal{P}$.

Our definition of SASPs is restricted in several ways because our goal here is to illustrate the core of a method to propagate probabilities from total choices to events. So these programs do not feature variables, relation symbols, functors or other elements common in standard ASP. Also probability annotations are not associated to (general) clause heads or disjunctions. However, these two last restrictions do not reduce the expressive capacity of the language because, for the former, a clause such as $\alpha : p \leftarrow \beta$ with an annotated head can be rewritten as:

$$\alpha : p \leftarrow \beta \qquad \Longrightarrow \qquad \begin{array}{l} \gamma : p, \\ \alpha \leftarrow \beta \wedge \gamma \end{array}$$

while annotated disjunctive facts, such as $\alpha \vee \beta : p$, can be replaced by

$$\alpha \vee \beta : p \qquad \Longrightarrow \qquad \begin{array}{c} \gamma : p, \\ \alpha \vee \beta \leftarrow \gamma. \end{array}$$

Interpretation of an SASP. The *derived program* of an SASP is obtained by replacing each probabilistic fact $\mathrm{a} : \mathrm{p}$ by a disjunction $\mathrm{a} \vee \overline{\mathrm{a}}$. The *stable models* of an SASP program are the stable models of its derived program. The set of SMs of a (derived or) SASP program is denoted $\mathcal{M}$.

Example 1 (Fruitful SASP) Consider A to be the following SASP:

$$\begin{aligned} \mathrm{a} &: 0.3, \\ \mathrm{b} \vee \mathrm{c} &\leftarrow \mathrm{a} \end{aligned} \tag{1}$$

which has the set $\mathcal{P} = \{\mathrm{a} : 0.3\}$ of probabilistic facts. This program is transformed into the disjunctive logic program

$$\begin{aligned} \mathrm{a} &\vee \overline{\mathrm{a}}, \\ \mathrm{b} \vee \mathrm{c} &\leftarrow \mathrm{a}, \end{aligned} \tag{2}$$

with the set $\mathcal{M} = \{\, \overline{\mathrm{a}}, \mathrm{ab}, \mathrm{ac} \,\}$ of three stable models. We use short-hand expressions like ab to denote a set of literals such as $\{\mathrm{a}, \mathrm{b}\}$. ■ Consider also the case $\mathrm{c} \leftarrow \mathrm{a}$.

The method we are proposing here does not follow the framework of [15] by attributing probabilities directly from the syntax of the program. Our approach requires that we consider the semantics, *i.e.* the stable models of the program, no matter the syntax that provided them.

Example 2 (No Syntax Propagation) *Consider the program*

$$\begin{aligned} \mathrm{a} &: 0.3, \\ \mathrm{b} &\leftarrow \mathrm{a} \end{aligned}$$

We don't follow the clause $\mathrm{b} \leftarrow \mathrm{a}$ *to attribute a probability to* b*. Instead, that will result from considering how events are related with the stable models and these with the total choices. If one follows the steps of section 3 would get*

$$\begin{aligned} &\mathrm{P}_{\mathcal{E}}(\mathrm{a}) = \mathrm{P}_{\mathcal{E}}(\mathrm{b}) = \mathrm{P}_{\mathcal{E}}(\mathrm{ab}) = 0.05, \\ &\mathrm{P}_{\mathcal{E}}(\overline{\mathrm{a}}) = \mathrm{P}_{\mathcal{E}}(\overline{\mathrm{a}}\mathrm{b}) = \mathrm{P}_{\mathcal{E}}\left(\overline{\mathrm{a}}\overline{\mathrm{b}}\right) = \frac{7}{60}, \\ &\mathrm{P}_{\mathcal{E}}\left(\{\}\right) = 0.5. \end{aligned}$$

Events

An *event* is a set of literals. We denote the set of events by $\mathcal{E}$. An event $e \in \mathcal{E}$ containing a set $\{x, \overline{x}\} \subseteq e$ of complementary literals is *inconsistent*; otherwise e is *consistent*. The set of consistent events is denoted by $\mathcal{W}$.

Example 3 (Events of the fruitful SASP) The atoms of program eq. (1) are $\mathcal{A} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ and the literals are

$$\mathcal{L} = \left\{ \overline{\mathrm{a}}, \overline{\mathrm{b}}, \overline{\mathrm{c}}, \mathrm{a}, \mathrm{b}, \mathrm{c} \right\}.$$

In this case, $\mathcal{E}$ has $2^6 = 64$ elements. Some, such as $\{\overline{\mathrm{a}}, \mathrm{a}, \mathrm{b}\}$, contain an atom and its negation (a and $\overline{\mathrm{a}}$ in that case) and are inconsistent. The set of atoms $\mathcal{A} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ above generates 37 inconsistent events and 27 consistent events. Notice that the empty set is an event, that we denote by λ.

As above, to simplify notation we write events as $\overline{\mathrm{a}}\mathrm{ab}$ instead of $\{\overline{\mathrm{a}}, \mathrm{a}, \mathrm{b}\}$.

Total Choices and their Probability

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

Example 4 (Probabilistic facts and total choices) Consider the program B defined by

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

The probabilistic facts of B are

$$\mathcal{P}_B = \{\, a : 0.3, b : 0.6 \,\},$$

and the total choices are

$$\mathcal{T}_B = \Big\{ \, \{a, b\} \, , \{a, \overline{b}\} \, , \{\overline{a}, b\} \, , \{\overline{a}, \overline{b}\} \, \Big\} \, .$$

E.g., the total choice $\{\overline{a}, b\}$ results from choosing $a' = \overline{a}$ from the probabilistic fact $a : 0.3$ and $b' = b$ from $b : 0.6$, while $\{a, b\}$ results from choosing $a' = a$ from $a : 0.3$ and $b' = b$ from $b : 0.6$.

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

$$P_{\mathcal{T}}(t) = \prod_{\substack{a:p \, \in \, \mathcal{P}, \\ a \, \in \, t}} p \quad \times \prod_{\substack{a:p \, \in \, \mathcal{P}, \\ \overline{a} \, \in \, t}} \overline{p}. \tag{3}$$

Here, $\overline{p} = 1 - p$, and we are using the subscript in $P_{\mathcal{T}}$ to explicitly state that this probability distribution concerns total choices. Later, we'll use subscripts $\mathcal{M}, \mathcal{E}$ to deal with distributions of stable models and events, $P_{\mathcal{M}}, P_{\mathcal{E}}$.

Example 5 (Probabilities for total choices) Continuing with the program from section 2, the probabilities of the total choices are:

$$\begin{aligned} P_{\mathcal{T}}\Big(\{a, b\}\Big) &= 0.3 \times 0.6 && = 0.18, \\ P_{\mathcal{T}}\Big(\{a, \overline{b}\}\Big) &= 0.3 \times \overline{0.6} &= 0.3 \times 0.4 &= 0.12, \\ P_{\mathcal{T}}\Big(\{\overline{a}, b\}\Big) &= \overline{0.3} \times 0.6 &= 0.7 \times 0.6 &= 0.42, \\ P_{\mathcal{T}}\Big(\{\overline{a}, \overline{b}\}\Big) &= \overline{0.3} \times \overline{0.6} &= 0.7 \times 0.4 &= 0.28. \end{aligned}$$

Suppose that in this program we change the probability in $b : 0.6$ to $b : 1.0$. Then the total choices are the same but the probabilities become

$$\begin{aligned} P_{\mathcal{T}}\Big(\{a, b\}\Big) &= 0.3 \times 1.0 && = 0.3, \\ P_{\mathcal{T}}\Big(\{a, \overline{b}\}\Big) &= 0.3 \times \overline{1.0} &= 0.3 \times 0.0 &= 0.0, \\ P_{\mathcal{T}}\Big(\{\overline{a}, b\}\Big) &= \overline{0.3} \times 1.0 &= 0.7 \times 1.0 &= 0.7, \\ P_{\mathcal{T}}\Big(\{\overline{a}, \overline{b}\}\Big) &= \overline{0.3} \times \overline{1.0} &= 0.7 \times 0.0 &= 0.0. \end{aligned}$$

which, as expected from stating that $b : 1.0$, is like having b as a (deterministic) fact:

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

This will also be stated in section 3.2, when the proper definitions are set.

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 $t \in \mathcal{T}$.

Example 6 (Stable models and total choices) Continuing section 2, the total choice $t = \{\overline{a}\}$ entails a single stable model, $\overline{a}$, so $\mathcal{M}(\{\overline{a}\}) = \{\overline{a}\}$ and, for $t = \{a\}$, the program has two stable models: $\mathcal{M}(\{a\}) = \{ab, ac\}$.

$t \in \mathcal{T}$	$\mathcal{M}(t)$
$\{\overline{a}\}$	$\overline{a}$
$\{a\}$	ab, ac

The second case illustrates that propagating probabilities from total choices to stable models entails a non-deterministic step: *How to propagate the probability* $P_{\mathcal{T}}(\{a\})$ *to each one of the stable models* ab *and* ac*?*

■ Repeated? Our goal is to propagate the probability distribution of the program's total choices, $P_{\mathcal{T}}$, in eq. (3) (which is, indeed, a product of Bernoulli distributions [30]), to a distribution of the program's events, $P_{\mathcal{E}}$.

Other Approaches and Systems

■ Repeated? The core problem of setting a semantic for probabilistic logic programs is 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`).

For example, the *credal set* approach, in [6], defines $P_{\mathcal{T}}$ like in eq. (3) but then, for $a \in \mathcal{A}$, $P(a \mid t)$ is unknown but bounded by $\underline{P}(a \mid t)$ and $\overline{P}(a \mid t)$, that can be explicitly estimated from the program.

Also `Problog` [8, 33] 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 defined only for *sound* programs [28] *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 [28, 32].

Another system, based on Markov Logic [26], is $\mathtt{LP}^{\mathtt{MLN}}$ [17, 18], 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 probability of that model.

Towards Propagating Probabilities from Literals to Events

Probabilistic logic programs can be traced to [15] where the syntax of the program determines the propagation from probabilities explicitly set either in facts or other elements of the program. We take a different approach, based on the semantics of a derived program.

Consider the program in eq. (1) from section 2 in order to showcase the problem of propagating probabilities from total choices to stable models and then to events. As mentioned before, the main issue arises from the lack of information in the program on how to assign a single probability to each stable model. This becomes a crucial problem in situations where multiple stable models result from a single total choice.

We will follow through with this example in section 3, where we present our proposal for propagating probabilities from total choices to stable models.

Example 7 (Probabilities and stable models) As a first step to propagate probability from total choices to events, consider the program eq. (1) and a possible propagation of $P_{\mathcal{T}} : \mathcal{T} \to [0, 1]$ from total choices to the stable models, $P_{\mathcal{M}} : \mathcal{M} \to [0, 1]$.

The stable models are the ones from its derived program (in eq. (2)):

$$\mathcal{M} = \{\overline{a}, ab, ac\}.$$

It might seem straightforward to assume $P_{\mathcal{M}}(\overline{a}) = 0.7$ but *there is no explicit way to assign values to* $P_{\mathcal{M}}(ab)$ *and* $P_{\mathcal{M}}(ac)$. Instead, we assume this lack of information and use a parameter θ as in

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

to express our knowledge that ab and ac are models related in a certain way and, simultaneously, our uncertainty about that relation. In general, there might be necessary several such parameters, each associated to a stable model s and a total choice t, so we write $\theta = \theta_{s,t}$.

A value for $\theta_{s,t}$ can't be determined just with the information given in that program, but might be estimated with the help of further information, such as empirical distributions from datasets.

Our stance is that an ASP program describes some system and then:

1. With a probability 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 extension procedure described below.

3. That knowledge can be used to estimate parameters 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 observations.

5. If observations are not consistent with the program, then we ought to conclude that the program is wrong and must be changed accordingly.

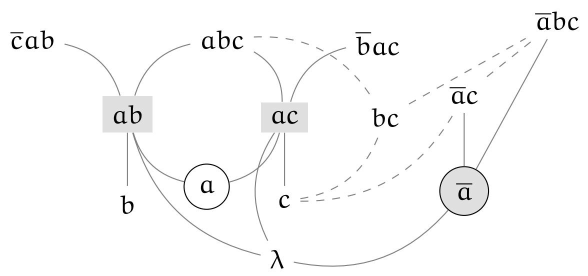


Figure 1: This (partial sub-/super-set) diagram shows some events related to the stable models of the program eq. (1). The circle nodes are total choices and shaded nodes are stable models. Solid lines represent relations with the stable models and dashed lines some sub-/super-set relations with other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

Currently, we are addressing the problem of propagating a probability function (possibly using parameters such as θ above), defined on the stable models of a program, $P_{\mathcal{M}} : \mathcal{M} \to [0, 1]$, to all the events of that program: $P_{\mathcal{E}} : \mathcal{E} \to [0, 1]$. This extension must satisfy the Kolmogorov axioms of probability so that probabilistic reasoning is consistent with the ASP program and follows our interpretation of stable models as the states of an observable system.

As sets, the stable models can have non-empty intersection. But as states of a system, we assume that the stable models are *distinct observations* in the following sense:

Assumption 1 *Stable models are disjoint probabilistic events. I.e., for any set X of stable models,*

$$P_{\mathcal{M}}(X) = \sum_{s \in X} P_{\mathcal{M}}(s) \tag{4}$$

Consider the stable models ab, ac from section 2, that result from the rule $b \vee c \leftarrow a$ and the total choice $\{a\}$. Since we intend to associate each stable model with a state of the system, ab and ac should be *disjoint* events. From that particular rule, no further relation between b and c is entailed. This does not prevent that other rules set further dependencies between b and c, which in turn may change the stable models.

By not making prior distribution assumptions based on the syntax of the program, we can state properties on its models, as we've done in assumption 1.

3 Propagating Probabilities

The diagram in fig. 1 illustrates the problem of propagating probabilities 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 probability, 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 justify in terms of the semantics of the program. Instead, we propose to base the extension in the relation an event has with the stable models.

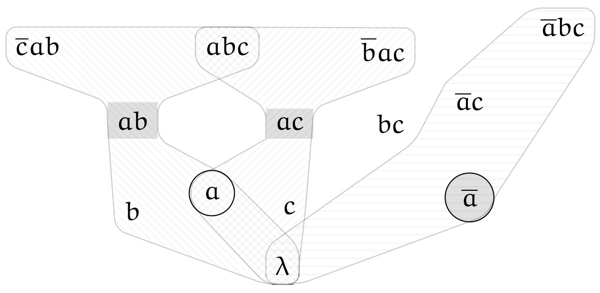


Figure 2: Classes of (consistent) events related to the stable models of section 2 are defined through sub-/super-set relations. In this picture we can see, for example, that $\{\overline{c}ab, ab, b\}$ and $\{a, abc\}$ are part of different classes, represented by different fillings. As before, the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

$$\begin{tikzcd}
 & \overline{a}, ac \arrow[rr] & & \Lambda & \\
 & & \boxed{\overline{a}} \arrow[ul] \arrow[rr] & & \overline{a}, ab \arrow[ul] \\
\text{(inconsistent) } \bot & & & & \\
 & \boxed{ac} \arrow[uuu] \arrow[rr] & & ab, ac \arrow[uuu] & \\
 & & \Diamond \arrow[ul] \arrow[uuu] \arrow[rr] & & \boxed{ab} \arrow[ul] \arrow[uuu]
\end{tikzcd}$$

Figure 3: Lattice of the stable cores from section 2. In this diagram the nodes are the different stable cores that result from the stable models plus the *inconsistent* class ($\bot$). 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. As in previous diagrams, shaded nodes represent the SMs.

3.1 An Equivalence Relation

Our path to propagate probabilities starts with a perspective of stable models as playing 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 section 2 and fig. 2 consider the stable models $\overline{a}, ab, ac$ and events a, abc and c. While a is related with (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

$$[\![e]\!] := \{ s \in \mathcal{M} \mid s \subseteq e \vee e \subseteq s \} \tag{5}$$

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.

Example 8 (Stable cores) Continuing section 2, depicted in figs. 1 to 3, and $\mathcal{M} = \{ab, ac, \overline{a}\}$, consider the following stable cores of some events:

$$
\begin{aligned}
[\![a]\!] &= \{s \in \mathcal{M} \mid s \subseteq a \vee a \subseteq s\} && = \{ab, ac\} \\
[\![abc]\!] &= \{s \in \mathcal{M} \mid s \subseteq abc \vee abc \subseteq s\} && = \{ab, ac\} \\
[\![\overline{c}ab]\!] &= \{s \in \mathcal{M} \mid s \subseteq \overline{c}ab \vee \overline{c}ab \subseteq s\} && = \{ab\} \\
[\![bc]\!] &= \{s \in \mathcal{M} \mid s \subseteq bc \vee bc \subseteq s\} && = \emptyset \\
[\![\lambda]\!] &= \{s \in \mathcal{M} \mid s \subseteq \emptyset \vee \emptyset \subseteq s\} = \{ab, ac, \overline{a}\} && = \Lambda
\end{aligned}
$$

Events a and abc have the same SC, while $\overline{c}ab$ has a different SC. Also, bc is *independent of* (*i.e.* not related to) any stable model. Since events are sets of literals, the empty set is an event and a subset of any SM.

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 *For a given program, let* $u, v \in \mathcal{E}$. *The equivalence relation* $\sim$ *is defined by*

$$u \sim v \;:\leftarrow\; u, v \notin \mathcal{W} \vee \left(u, v \in \mathcal{W} \wedge [\![u]\!] = [\![v]\!]\right). \tag{6}$$

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

$$[e]_\sim = \begin{cases} \bot := \mathcal{E} \setminus \mathcal{W} & \text{if } e \in \mathcal{E} \setminus \mathcal{W}, \\ \left\{u \in \mathcal{W} \;\middle|\; [\![u]\!] = [\![e]\!]\right\} & \text{if } e \in \mathcal{W}. \end{cases} \tag{7}$$

Proposition 1 (Bottom class) *If we denote by* λ *the empty set as an event, and by* Λ *the class (the* bottom class*) of events related with all the stable models, then*

$$[\lambda]_\sim = [\![\mathcal{M}]\!] = \Lambda. \tag{8}$$

The combinations of the stable models, together with the set of inconsistent events ($\bot$) form a set of representatives.

Example 9 (Events classes) Consider again section 2. As previously stated, the stable models are the elements of $\mathcal{M} = \{\overline{a}, ab, ac\}$ so the quotient set of this relation is

$$[\mathcal{E}]_\sim = \left\{ \begin{array}{lll} \bot, & \Diamond, & [\![\overline{a}]\!], \\ [\![ab]\!], & [\![ac]\!], & [\![\overline{a}, ab]\!], \\ [\![\overline{a}, ac]\!], & [\![ab, ac]\!], & \Lambda \end{array} \right\},$$

where $\Diamond$ denotes the class of *independent events* e such that $[\![e]\!] = \{\emptyset\}$, while $\Lambda = [\![\mathcal{M}]\!]$ is the set of events related with all SMs. We have:

$[\![e]\!]$	$[e]_\sim$	$\#[e]_\sim$
$\bot$	$\overline{a}a, \ldots$	37
$\Diamond$	$\overline{b}, \overline{c}, bc, \overline{b}a, \overline{b}c, \overline{b}\overline{c}, \overline{c}a, \overline{c}b, \overline{b}\overline{c}a$	9
$\overline{a}$	$\overline{a}, \overline{a}b, \overline{a}c, \overline{a}\overline{b}, \overline{a}\overline{c}, \overline{a}bc, \overline{a}\overline{c}b, \overline{a}\overline{b}c, \overline{a}\overline{b}\overline{c}$	9
ab	$b, ab, \overline{c}ab$	3
ac	$c, ac, \overline{b}ac$	3
$\overline{a}, ab$		0
$\overline{a}, ac$		0
ab, ac	a, abc	2
Λ	λ	1
$[\mathcal{E}]_\sim$	$\mathcal{E}$	64

Notice that $bc \in \Diamond$, as hinted by figs. 1 and 2.

Since all events within an equivalence class have the same relation with a specific set of stable models, we are interested in functions, including probability, that are constant within classes. A function $f : \mathcal{E} \to Y$, where Y is any set, is *coherent* if

$$\forall u \in [e]_\sim \left(f(u) = f(e)\right). \tag{9}$$

■ Adopt the term "coherent" on the following text.

In the specific case of eq. (1), instead of dealing with the $64 = 2^6$ events, we need to consider only the $9 = 2^3 + 1$ classes, well defined in terms of combinations of the stable models, to define coherent functions.

3.2 From total choices to events

Our path to set a distribution on $\mathcal{E}$ starts with the more general problem of extending *measures*, since extending *probabilities* easily follows by means of a suitable normalization (done in eqs. (17) and (19)), and has two phases:

1. Propagation of the probabilities, *as measures*, from the total choices to events.
2. Normalization of the measures on events, recovering a probability.

The "propagation" phase, traced by eq. (3) and eqs. (10) to (16), starts with the measure (probability) of total choices, $\mu_{\mathcal{T}}(t) = P_{\mathcal{T}}(t)$, propagates it, as a *measure*, to the stable models, $\mu_{\mathcal{M}}(s)$, and then, within the equivalence relation from eq. (6), to a coherent measure of events, $\mu_{\mathcal{E}}(e)$, including (consistent) worlds. So we are setting a sequence of functions

$$\mu_{\mathcal{T}} \longrightarrow \mu_{\mathcal{M}} \longrightarrow \mu_{C} \longrightarrow \mu_{\mathcal{E}}$$

on successive larger domains

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

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 define a probability distribution of events by normalization:

$$\mu_{\mathcal{E}} \longrightarrow P_{\mathcal{E}}.$$

Total choices.

Using eq. (3), for $t \in \mathcal{T}$,

$$\mu_{\mathcal{T}}(t) := P_{\mathcal{T}}(t) = \prod_{\substack{a:p \,\in\, \mathcal{P},\\ a \,\in\, t}} p \times \prod_{\substack{a:p \,\in\, \mathcal{P},\\ \overline{a} \,\in\, t}} \overline{p}. \tag{10}$$

Stable models.

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 variables or values $\theta_{s,t} \in [0,1]$ such that $\sum_{s \in \mathcal{M}(t)} \theta_{s,t} = 1$, we define

$$\mu_{\mathcal{M}}(s,t) := \begin{cases} \theta_{s,t} & \text{if } s \in \mathcal{M}(t) \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

The $\theta_{s,t}$ parameters in eq. (11) 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 problem by assigning a possibly unknown parameter $(\theta_{s,t})$ conditional on the total choice (t) to each 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.

Example 10 (Stable models and parameters) The program from section 2 has no information about the probabilities of the stable models that result from the total choice $t = \{a\}$. These models are $\mathcal{M}(\{a\}) = \{ab, ac\}$ so we need two parameters $\theta_{ab,\{a\}}, \theta_{ac,\{a\}} \in [0,1]$ and such that (*cf.* eq. (11))

$$\theta_{ab,\{a\}} + \theta_{ac,\{a\}} = 1.$$

If we let $\theta = \theta_{ab,\{a\}}$ then

$$\theta_{ac,\{a\}} = 1 - \theta = \overline{\theta}.$$

Also

$$\theta_{ab,\{\overline{a}\}} = 0,$$
$$\theta_{ac,\{\overline{a}\}} = 0$$

because $ab, ac \not\in \mathcal{M}(\overline{a})$.

Classes.

Each class is either the inconsistent class $(\bot)$ or is represented by a stable core, *i.e.* a set of stable models.

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

$$\mu_C(\bot, t) := 0.^{2} \tag{12}$$

Independent class. An event related with no stable model corresponds to a non-state, according to the program. So the respective measure is also set to zero:

$$\mu_C(\Diamond, t) := 0. \tag{13}$$

Other classes. For the propagation function to be coherent, it must be constant within a class, its value dependent only on the stable core, and respect assumption 1 (*i.e.* stable models are disjoint probabilistic events):

$$\mu_C\left([e]_\sim, t\right) := \mu_{\mathcal{M}}\left(\llbracket e \rrbracket, t\right) = \sum_{s \in \llbracket e \rrbracket} \mu_{\mathcal{M}}(s,t) \tag{14}$$

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

and we further define

$$\mu_C\big([e]_\sim\big) := \sum_{t \in \mathcal{T}} \mu_C\big([e]_\sim, t\big)\, \mu_{\mathcal{T}}(t)\,. \tag{15}$$

Equation (14) states that the measure of a class $[e]_\sim$ is the sum over its stable core ($[\![e]\!]$) and eq. (15) *marginalizes* eq. (14) over the total choices.

Example 11 (Measure of stable models and classes)

	A	B	C	D	E	F	G	H	I	J
	$[\![e]\!]$	$\mu_{\mathcal{M}}\big(s, \{\overline{a}\}\big)$			$\mu_{\mathcal{M}}\big(s, \{a\}\big)$			$\mu_C\big([e]_\sim, \{\overline{a}\}\big)$	$\mu_C\big([e]_\sim, \{a\}\big)$	$\mu_C\big([e]_\sim\big)$
		$\overline{a}$	ab	ac	$\overline{a}$	ab	ac	$\mu_{\mathcal{T}}\big(\{\overline{a}\}\big)$	$\mu_{\mathcal{T}}\big(\{a\}\big)$	
		1	0	0	0	θ	$\overline{\theta}$	0.7	0.3	
1	$\overline{a}$	1			0			1	0	0.7
2	ab		0			θ		0	θ	0.3θ
3	ac			0			$\overline{\theta}$	0	$\overline{\theta}$	$0.3\overline{\theta}$
4	$\overline{a}, ab$	1	0		0	θ		1	θ	$0.7 + 0.3\theta$
5	$\overline{a}, ac$	1		0	0		$\overline{\theta}$	1	$\overline{\theta}$	$0.7 + 0.3\overline{\theta}$
6	ab, ac		0	0		θ	$\overline{\theta}$	0	$\theta + \overline{\theta} = 1$	0.3
7	Λ	1	0	0	0	θ	$\overline{\theta}$	1	$\theta + \overline{\theta} = 1$	1

Continuing section 2, we show the propagation of $\mu_{\mathcal{T}}$ to $\mu_{\mathcal{M}}$ (eq. (10)) and then to μ_C (eqs. (14) and (15)). The table above resumes the calculations to compute $\mu_C\big([e]_\sim\big)$ for each $e \in \mathcal{E}$. For example, $e = abc$ the calculation of $J6 = \mu_C\big([abc]_\sim\big)$ follows these steps:

1. $[\![abc]\!] = \{ab, ac\}$ — is in line 6 of the table.
2. Since $\mathcal{T} = \big\{\{a\}, \{\overline{a}\}\big\}$, we need to calculate $I6 = \mu_C\big([abc]_\sim, \{a\}\big)$ and $H6 = \mu_C\big([abc]_\sim, \{\overline{a}\}\big)$. By eq. (14):

$$H6 = \mu_C\big([abc]_\sim, \{\overline{a}\}\big) = \sum_{s \in [\![abc]\!]} \mu_{\mathcal{M}}\big(s, \{\overline{a}\}\big) = \mu_{\mathcal{M}}\big(ab, \{\overline{a}\}\big) + \mu_{\mathcal{M}}\big(ac, \{\overline{a}\}\big)$$

$$I6 = \mu_C\big([abc]_\sim, \{a\}\big) = \sum_{s \in [\![abc]\!]} \mu_{\mathcal{M}}\big(s, \{a\}\big) = \mu_{\mathcal{M}}\big(ab, \{a\}\big) + \mu_{\mathcal{M}}\big(ac, \{a\}\big)$$

3. The $\mu_{\mathcal{M}}(s, t)$ above result from eq. (11) — the non-empty cells in columns $B : D$ and $E : G$:

$$\begin{aligned} C6 = \mu_{\mathcal{M}}\big(ab, \{\overline{a}\}\big) &= 0 \\ D6 = \mu_{\mathcal{M}}\big(ac, \{\overline{a}\}\big) &= 0 \\ F6 = \mu_{\mathcal{M}}\big(ab, \{a\}\big) &= \theta \\ G6 = \mu_{\mathcal{M}}\big(ac, \{a\}\big) &= \overline{\theta} \end{aligned}$$

4. So we have — columns H, I:

$$\begin{aligned} H6 = \mu_C\big([abc]_\sim, \{\overline{a}\}\big) &= 0 + 0 &&= 0 \\ I6 = \mu_C\big([abc]_\sim, \{a\}\big) &= \theta + \overline{\theta} &&= 1 \end{aligned}$$

5. At last, by eq. (15) — columns H, I and J:

$$\begin{aligned} J6 = \mu_C\left([abc]_\sim\right) &= \sum_{t\in\mathcal{M}} \mu_C\left([abc]_\sim, t\right) \mu_{\mathcal{T}}(t) \\ &= \mu_C\left([abc]_\sim, \{\overline{a}\}\right) \mu_{\mathcal{T}}\left(\{\overline{a}\}\right) + \mu_C\left([abc]_\sim, \{a\}\right) \mu_{\mathcal{T}}\left(\{a\}\right) \\ &= 0\overline{\theta} + 1\theta = 0 \times 0.7 + 1 \times 0.3 = 0.3 \end{aligned}$$

Events and probability.

Each (non-inconsistent) event $e \in \mathcal{E}$ is in the class defined by its stable core, $[\![e]\!]$. So, denoting by $\#X$ the number of elements in X, we set:

$$\mu_{\mathcal{E}}(e, t) := \begin{cases} \frac{\mu_C\left([e]_\sim, t\right)}{\#[e]_\sim} & \text{if } \#[e]_\sim > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

and

$$\mu_{\mathcal{E}}(e) := \sum_{t\in\mathcal{T}} \mu_{\mathcal{E}}(e, t)\, \mu_{\mathcal{T}}(t)\,. \tag{17}$$

The *normalizing factor* is:

$$Z := \sum_{e\in\mathcal{E}} \mu_{\mathcal{E}}(e) = \sum_{[e]_\sim \in [\mathcal{E}]_\sim} \mu_C\left([e]_\sim\right), \tag{18}$$

and now eq. (17) provides a straightforward way to define the *probability of (observing) a single event* $e \in \mathcal{E}$:

$$P_{\mathcal{E}}(e) := \frac{\mu_{\mathcal{E}}(e)}{Z}. \tag{19}$$

Equation (19) defines a coherent *prior* probability 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 section 3.2 can't) be confused with the explicit $\mu_{\mathcal{T}}$ set in the program.

Example 12 (Coherent probability of events) In section 3.2 we determined $\mu_C\left([e]_\sim, t\right)$ from eq. (14) and also $\mu_C\left([e]_\sim\right)$, the measure of each class, using eq. (15), that marginalizes the total choices.

$[\![e]\!]$	μ_C	$\#[e]_\sim$	$\mu_\mathcal{E}$	$P_\mathcal{E}$
$\bot$	0	37	0	0
$\Diamond$	0	9	0	0
$\overline{a}$	$\frac{7}{10}$	9	$\frac{7}{90}$	$\frac{7}{207}$
ab	$\frac{3}{10}\theta$	3	$\frac{1}{10}\theta$	$\frac{1}{23}\theta$
ac	$\frac{3}{10}\overline{\theta}$	3	$\frac{1}{10}\overline{\theta}$	$\frac{1}{23}\overline{\theta}$
$\overline{a}, ab$	$\frac{7+3\theta}{10}$	0	0	0
$\overline{a}, ac$	$\frac{7+3\overline{\theta}}{10}$	0	0	0
ab, ac	$\frac{3}{10}$	2	$\frac{3}{20}$	$\frac{3}{46}$
Λ	1	1	1	$\frac{10}{23}$
	$Z = \frac{23}{10}$		$\frac{\mu_C}{\#[e]_\sim}$	$\frac{\mu_\mathcal{E}}{X}$

From there we can follow eq. (16) to calculate the measure $\mu_\mathcal{E}(e, t)$ of each event given t, by simply dividing $\mu_C\left([e]_\sim, t\right)$ by $\#[e]_\sim$, the total number of elements in $[e]_\sim$. Then we marginalize t in $\mu_\mathcal{E}(e, t)$ to get $\mu_\mathcal{E}(e)$. Finally, the normalization factor from eq. (18) and eq. (19) provide a coherent *prior* probability for each event.

In summary, the coherent *prior* probability of events of program eq. (1) is

$$\begin{array}{c|ccccccccc} [\![e]\!] & \bot & \Diamond & \overline{a} & ab & ac & \overline{a}, ab & \overline{a}, ac & ab, ac & \Lambda \\ \hline P_\mathcal{E}(e) & 0 & 0 & \frac{7}{207} & \frac{1}{23}\theta & \frac{1}{23}\overline{\theta} & 0 & 0 & \frac{3}{46} & \frac{10}{23} \end{array} \tag{20}$$

It is now straightforward to check that $P_\mathcal{E}$ satisfies the Kolmogorov axioms of probability for $\Omega = \{X \mid X \subseteq \mathcal{E}\}$ because it is explicitly defined in each $e \in \mathcal{E}$.

Now we can properly formulate the property stated in section 2 about probabilistic facts such as $a : 1.0$.

Proposition 2 *Consider a program A with the probabilistic fact $a : 1.0$ and A' where it is replaced by the deterministic fact a. Let $P_\mathcal{E}$ be as eq. (19) for A and $P'_\mathcal{E}$ for A'. Then*

$$\forall e \in \mathcal{E} \left(P_\mathcal{E}(e) = P'_\mathcal{E}(e)\right). \tag{21}$$

Since total choices are also events, one can ask, for an arbitrary total choice t, if $P_\mathcal{T}(t) = P_\mathcal{E}(t)$ or, equivalently, if $\mu_\mathcal{T}(t) = \mu_\mathcal{E}(t)$. However, it is easy to see that, in general, that cannot be true. While the domain of $P_\mathcal{T}$ is the set of total choices, for $P_\mathcal{E}$ 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 have non-zero probability.

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*

$$P_\mathcal{T}(t) \neq P_\mathcal{E}(t). \tag{22}$$

Proof 1 *Suppose towards a contradiction that $P_\mathcal{T}(t) = P_\mathcal{E}(t)$ for all $t \in \mathcal{T}$. Then*

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

Hence $P_\mathcal{E}(x) = 0$ 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 $P_\mathcal{E}(s) > 0$.

The essential conclusion of section 3.2 is that we are dealing with *two distributions*: one, on the total choices, explicit in the annotations of the programs and another one, on the events, entailed by the explicit annotations *and the structure of the stable models*.

Our approach generalizes to Bayesian networks in a way similar to [6, 25] and [14, 31] 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 SASP.

4 Discussion and Future Work

This work is a first venture into expressing probability 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 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, probabilistic logical programming, machine learning, inductive programming, *etc.*

The equivalence relation from section 3.1 identifies the $s \subseteq e$ and $e \subseteq s$ cases. Relations that distinguish such cases might enable better relations between the models and processes from the stable models.

The example from appendix A.2 shows that the theory, methodology, and tools, from Bayesian Networks can be adapted to our approach. The connection with Markov Fields [16] is left for future work. An example of a "program selection" application (as mentioned in item 4, section 2) is also left for future work.

■ The fruitful example in the appendix must be adaped to illustrate the estimations of θ in the three scenarios. Related with the remark at the end of appendix A.1, on the tendency of $\widehat{\theta}$ to under- or over- estimate θ, notice that the error function in (23) expresses only one of many possible "distances" between the empirical and prior distributions. Variations include normalizing this function by the size of $\mathcal{E}$ or using the Kullback-Leibler divergence. The key contribution of this function in this work is to find an optimal θ. Moreover, further experiments, not included in this paper, with $\alpha = 0.0$, lead to $\widehat{\theta} \approx \gamma$, *i.e.* setting the prior noise to zero leads to full recovering θ from the observations.

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 observation 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

[1] Weronika T Adrian et al. “The ASP system DLV: advancements and applications”. In: *KI-Künstliche Intelligenz* 32 (2018), pp. 177–179.

[2] Marco Alberti et al. “cplint on SWISH: Probabilistic logical inference with a web browser”. In: *Intelligenza Artificiale* 11.1 (2017), pp. 47–64.

[3] Joaquín Arias et al. “Justifications for goal-directed constraint answer set programming”. In: *arXiv preprint arXiv:2009.10238* (2020).

[4] Chitta Baral, Michael Gelfond, and Nelson Rushton. “Probabilistic reasoning with Answer Sets”. In: *Theory and Practice of Logic Programming* 9.1 (2009), pp. 57–144.

[5] Jeff Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671.

[6] Fabio Gagliardi Cozman and Denis Deratani Mauá. “The joy of probabilistic answer set programming: semantics, complexity, expressivity, inference”. In: *International Journal of Approximate Reasoning* 125 (2020), pp. 218–239.

[7] Luc De Raedt et al. “ProbLog: A probabilistic Prolog and its application in link discovery”. In: *IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence*. IJCAI-INT JOINT CONF ARTIF INTELL. 2007, pp. 2462–2467.

[8] Daan Fierens et al. “Inference and learning in probabilistic logic programs using weighted boolean formulas”. In: *Theory and Practice of Logic Programming* 15.3 (2015), pp. 358–401.

[9] Martin Gebser et al. *Answer set solving in practice*. Springer Nature, 2022.

[10] Martin Gebser et al. “Potassco: The Potsdam answer set solving collection”. In: *AI Communications* 24.2 (2011), pp. 107–124.

[11] Michael Gelfond and Vladimir Lifschitz. “The stable model semantics for logic programming.” In: *ICLP/SLP*. Vol. 88. Cambridge, MA. 1988, pp. 1070–1080.

[12] Stuart Geman and Donald Geman. “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (1984), pp. 721–741. DOI: 10.1109/TPAMI.1984.4767596.

[13] Shashi Gowda et al. “High-performance symbolic-numerics via multiple dispatch”. In: *arXiv preprint arXiv:2105.03949* (2021).

[14] 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*. Springer. 1992, pp. 421–436.

[15] Michael Kifer and VS Subrahmanian. “Theory of generalized annotated logic programming and its applications”. In: *The Journal of Logic Programming* 12.4 (1992), pp. 335–367.

[16] Ross Kindermann and J. Laurie Snell. *Markov random fields and their applications*. Vol. 1. Contemporary Mathematics. American Mathematical Society, Providence, RI, 1980, pp. ix+142. ISBN: 0-8218-5001-6.

[17] 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.

[18] Joohyung Lee and Zhun Yang. "LPMLN, Weak Constraints, and P-log". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[19] Vladimir Lifschitz. "Answer set programming and plan generation". In: *Artificial Intelligence* 138.1 (2002), pp. 39–54. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(02)00186-8.

[20] Vladimir Lifschitz. "Twelve definitions of a stable model". In: *International Conference on Logic Programming*. Springer. 2008, pp. 37–51.

[21] Kyle Marple, Elmer Salazar, and Gopal Gupta. "Computing stable models of normal logic programs without grounding". In: *arXiv preprint arXiv:1709.00501* (2017).

[22] 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*. Springer. 1997, pp. 420–429.

[23] Jukka Pajunen and Tomi Janhunen. "Solution enumeration by optimality in Answer Set Programming". In: *Theory and Practice of Logic Programming* 21.6 (2021), pp. 750–767.

[24] 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, pp. 552–552. ISBN: 0-934613-73-7.

[25] Luc De Raedt et al. "Statistical relational artificial intelligence: Logic, probability, and computation". In: *Synthesis lectures on artificial intelligence and machine learning* 10.2 (2016), pp. 1–189.

[26] Matthew Richardson and Pedro Domingos. "Markov logic networks". In: *Machine learning* 62 (2006), pp. 107–136.

[27] Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*. en. 1st ed. New York: River Publishers, Sept. 2022. ISBN: 978-1-00-333819-2. DOI: 10.1201/9781003338192. (Visited on 03/01/2023).

[28] Fabrizio Riguzzi and Terrance Swift. "Well–definedness and efficient inference for probabilistic logic programming under the distribution semantics". In: *Theory and practice of logic programming* 13.2 (2013), pp. 279–302.

[29] Taisuke Sato. "A Statistical Learning Method for Logic Programs with Distribution Semantics". In: *International Conference on Logic Programming*. 1995.

[30] Jozef L. Teugels. "Some representations of the multivariate Bernoulli and binomial distributions". In: *J. Multivariate Anal.* 32.2 (1990), pp. 256–268. ISSN: 0047-259X,1095-7243. DOI: 10.1016/0047-259X(90)90084-U. URL: https://doi.org/10.1016/0047-259X(90)90084-U.

[31] Helmut Thöne, Ulrich Güntzer, and Werner Kießling. "Increased robustness of Bayesian networks through probability intervals". In: *International Journal of Approximate Reasoning* 17.1 (1997), pp. 37–76.

[32] Allen Van Gelder, Kenneth A Ross, and John S Schlipf. “The well-founded semantics for general logic programs”. In: *Journal of the ACM (JACM)* 38.3 (1991), pp. 619–649.

[33] Victor Verreet et al. “Inference and learning with model uncertainty in probabilistic logic programs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 2022, pp. 10060–10069.

$[\![e]\!]$	$\#Z$	$P_S([e]_\sim)$	$P_\mathcal{E}([e]_\sim)$
$\perp$	0	0	0
$\lozenge$	24	$\frac{24}{1000}$	0
$\overline{a}$	647	$\frac{647}{1000}$	$\frac{7}{23}$
ab	66	$\frac{66}{1000}$	$\frac{3}{23}\theta$
ac	231	$\frac{231}{1000}$	$\frac{3}{23}\overline{\theta}$
$\overline{a}, ab$	0	0	0
$\overline{a}, ac$	0	0	0
ab, ac	7	$\frac{7}{1000}$	$\frac{3}{23}$
$\overline{a}, ab, ac$	25	$\frac{25}{1000}$	$\frac{10}{23}$
	$n = 1000$		

Table 1: *Experiment 1: Bias to* ac. Results from an experiment where $n = 1000$ samples where generated following the *Model+Noise* procedure with parameters $\alpha = 0.1, \beta = 0.3, \gamma = 0.2$. Column #Z lists the number of observations on each class, the *empirical* distribution is represented by P_S and the *prior*, as before, is denoted by $P_\mathcal{E}$.

A Developed Examples

Here we apply the methods from section 3 to section 2 and to a well known Bayesian network: the Earthquake, Burglar, Alarm toy problem.

A.1 Estimating Parameters of the Fruitful Example

Table 1 lists the empirical results from an experiment where samples are classified according to the classes of section 2. These results can be *generated by simulation* in a two-step process, where (1) a "system" is *simulated*, to gather some "observations" and then (2) empirical distributions from those samples are *related* with the prior distributions from section 3.2. Tables 1 and 2 summarize some of those tests, where datasets of $n = 1000$ observations are generated and analyzed.

Simulating a System. Following some criteria, more or less related to the given program, a set of events, that represent observations, is generated. Possible simulation procedures include:

- *Random.* Each sample is a Random Set of Literals (RSL). Additional sub-criteria may require, for example, consistent events, a Random Consistent Event (RCE) simulation.
- *Model+Noise.* Gibbs' sampling [12] tries to replicate the program model and also to add some noise. For example, let $\alpha, \beta, \gamma \in [0, 1]$ be some parameters to control the sample generation. The first parameter, α is the "out of model" samples ratio; β represents the choice a or $\overline{a}$ (explicit in the model) and γ is the simulation representation of θ. A single sample is then generated following the

$[\![e]\!]$	$\#_{0.2}$	$\#_{0.8}$	$\#_{0.5}$
$\bot$	0	0	0
$\Diamond$	24	28	23
$\overline{a}$	647	632	614
ab	66	246	165
ac	231	59	169
$\overline{a}, ab$	0	0	0
$\overline{a}, ac$	0	0	0
ab, ac	7	8	4
$\overline{a}, ab, ac$	25	27	25

Table 2: *Experiments 2 and 3.* Results from experiments, each with $n = 1000$ samples generated following the *Model+Noise* procedure, with parameters $\alpha = 0.1, \beta = 0.3, \gamma = 0.8$ (Experiment 2: bias to ab.) and $\gamma = 0.5$ (Experiment 3: balanced ab and ac.). Empirical distributions are represented by the random variables $S_{0.8}$ and $S_{0.5}$ respectively. Data from experience table 1 is also included, and denoted by $S_{0.2}$, to provide reference. Columns $\#_{0.2}$, $\#_{0.8}$ and $\#_{0.5}$ contain $\#\left\{S_{0.2} \in [e]_\sim\right\}$, $\#\left\{S_{0.8} \in [e]_\sim\right\}$ and $\#\left\{S_{0.5} \in [e]_\sim\right\}$, the respective number of events in each class.

probabilistic choices below:

$$\begin{cases} \alpha & \text{by RCE} \\ & \begin{cases} \beta & \overline{a} \\ & \begin{cases} \gamma & ab \\ & ac \end{cases} \end{cases} \end{cases},$$

where

$$\begin{cases} p & x \\ & y \end{cases}$$

denotes "*the value of* x *with probability* p*, otherwise* y" — notice that y might entail x and *vice-versa*: E.g. some ab can be generated in the RCE.

- *Other Processes.* Besides the two sample generations procedures above, any other processes and variations can be used. For example, requiring that one of $x, \overline{x}$ literals is always in a sample or using specific distributions to guide the sampling of literals or events.

Relating the Empirical and the Prior Distributions. The data from the simulated observations is used to test the prior distribution. Consider the prior, $P_\mathcal{E}$, and the empirical, P_S, distributions and the following error function:

$$\mathrm{err}(\theta) := \sum_{e \in \mathcal{E}} \left(P_\mathcal{E}(e) - P_S(e)\right)^2. \tag{23}$$

Since $P_\mathcal{E}$ depends on θ, one can ask how does the error varies with θ, what is the *optimal* (i.e. minimum) error value

$$\hat{\theta} := \arg\min_\theta \mathrm{err}(\theta) \tag{24}$$

and what does it tell us about the program.

In order to illustrate this analysis, consider the experiment summarized in table 1.

1. Equation (23) becomes
$$\mathrm{err}(\theta) = \frac{20869963}{66125000} + \frac{477}{52900}\theta + \frac{18}{529}\theta^2.$$

2. The minimum of $\mathrm{err}(\theta)$ is at $\frac{477}{52900} + 2\frac{18}{529}\theta = 0$. Since this leads to a negative θ value $\theta \in [0, 1]$, it must be $\widehat{\theta} = 0$, and
$$\mathrm{err}\left(\widehat{\theta}\right) = \frac{20869963}{66125000} \approx 0.31561.$$

The parameters α, β, γ of that experiment favour ac over ab. In particular, setting $\gamma = 0.2$ means that in the simulation process, choices between ab and ac favour ac, 4 to 1. For completeness sake, we also describe one experiment that favours ab over ac (setting $\gamma = 0.8$) and one balanced ($\gamma = 0.5$).

For $\gamma = 0.8$, the error function is
$$\begin{aligned}\mathrm{err}(\theta) &= \frac{188207311}{529000000} - \frac{21903}{264500}\theta + \frac{18}{529}\theta^2 \\ &\approx 0.35579 - 0.08281\theta + 0.03403\theta^2\end{aligned}$$

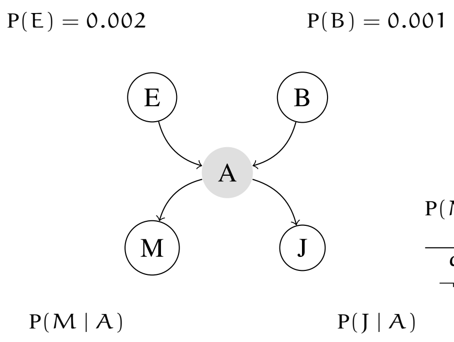
and, with $\theta \in [0, 1]$ the minimum is at $-0.08281 + 0.06805\theta = 0$, *i.e.*:
$$\widehat{\theta} : \frac{0.08281}{0.06805} \approx 1.21683 \quad > 1.$$

So, $\widehat{\theta} = 1, \mathrm{err}\left(\widehat{\theta}\right) \approx 0.30699$.

For $\gamma = 0.5$, the error function is
$$\begin{aligned}\mathrm{err}(\theta) &= \frac{10217413}{33062500} - \frac{2181}{66125}\theta + \frac{18}{529}\theta^2 \\ &\approx 0.30903 - 0.03298\theta + 0.03402\theta^2\end{aligned}$$

and, with $\theta \in [0, 1]$ the minimum is at $-0.03298 + 0.06804\theta = 0$, *i.e.*:
$$\begin{aligned}\widehat{\theta} &\approx \frac{0.03298}{0.06804} \approx 0.48471 \approx \frac{1}{2}, \\ \mathrm{err}\left(\widehat{\theta}\right) &\approx 0.30104\end{aligned}$$

These experiments show that data can indeed be used to estimate the parameters of the model. However, we observe that the estimated $\widehat{\theta}$ has a tendency to over- or under-estimate the θ used to generate the samples. More precisely, in experiment 1 data is generated with $\gamma = 0.2$ (the surrogate of θ) which is under-estimated with $\widehat{\theta} = 0$ while in experiment 2, $\gamma = 0.8$ leads the over-estimation $\widehat{\theta} = 1$. This suggests that we might need to refine the error estimation process. However, experiment 3 data results from $\gamma = 0.5$ and we've got $\widehat{\theta} \approx 0.48471 \approx 0.5$, which is more in line with what is to be expected.



$P(A \mid B \wedge E)$

		a	$\neg a$
b	e	0.95	0.05
b	$\neg e$	0.94	0.06
$\neg b$	e	0.29	0.71
$\neg b$	$\neg e$	0.001	0.999

$P(M \mid A)$

	m	$\neg m$
a	0.9	0.1
$\neg a$	0.05	0.95

$P(J \mid A)$

	j	$\neg j$
a	0.7	0.3
$\neg a$	0.01	0.99

Figure 4: The Earthquake, Burglary, Alarm model

A.2 An Example Involving Bayesian Networks

As it turns out, our framework is suitable to deal with more sophisticated cases, in particular cases involving bayesian networks. In order to illustrate this, in this section we see how the classical example of the Burglary, Earthquake, Alarm [24] works in our setting. This example is a commonly used example in bayesian networks because it illustrates reasoning under uncertainty. The gist of the example is given in fig. 4. It involves a simple network of events and conditional probabilities.

The events are: Burglary (B), Earthquake (E), Alarm (A), Mary calls (M) and John calls (J). The initial events B and E are assumed to be independent events that occur with probabilities $P(B)$ and $P(E)$, respectively. There is an alarm system that can be triggered by either of the initial events B and E. The probability of the alarm going off is a conditional probability given that B and E have occurred. One denotes these probabilities, as per usual, by $P(A \mid B)$, and $P(A \mid E)$. There are two neighbors, Mary and John who have agreed to call if they hear the alarm. The probability that they do actually call is also a conditional probability denoted by $P(M \mid A)$ and $P(J \mid A)$, respectively.

We follow the convention of representing the (upper case) random variable X by the (lower case) positive literal x. Considering the probabilities given in fig. 4 we obtain the following specification:

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

For the table giving the probability $P(M \mid A)$ we obtain the program:

$$\begin{aligned} p_{m|a} &: 0.9, \\ p_{m|\overline{a}} &: 0.05, \\ m &\leftarrow a \wedge p_{m|a}, \\ m &\leftarrow \neg a \wedge p_{m|\overline{a}}. \end{aligned}$$

The latter program can be simplified (abusing notation) by writing $m : 0.9 \leftarrow a$ and $m : 0.05 \leftarrow \neg a$.

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

$$\begin{gathered} \mathsf{j} : 0.7 \leftarrow \mathsf{a}, \\ \mathsf{j} : 0.01 \leftarrow \neg \mathsf{a}, \end{gathered}$$

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

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

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.