Modularity of P-Log Programs

Carlos Viegas Damásio and João Moura

CENTRIA, Departamento de Informática
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal

Abstract. We propose an approach for modularizing P-log programs and corresponding compositional semantics based on conditional probability measures. We do so by resorting to Oikarinen and Janhunen's definition of a logic program module and extending it to P-log by introducing the notions of input random attributes and output literals. For answering to P-log queries our method does not imply calculating all the stable models (possible worlds) of a given program, and previous calculations can be reused. Our proposal also handles probabilistic evidence by conditioning (observations).

Keywords: P-log, Answer Set Programming, Modularization, Probabilistic Reasoning.

1 Introduction and Motivation

The P-log language [3] has emerged as one of the most flexible frameworks for combining probabilistic reasoning with logical reasoning, in particular, by distinguishing acting (doing) from observations and allowing non-trivial conditioning forms [3,4]. The P-log languages is a non-monotonic probabilistic logic language supported by two major formalisms, namely Answer Set Programming [7,11,12] for declarative knowledge representation and Causal Bayesian Networks [15] as its probabilistic foundation. In particular, ordinary Bayesian Networks can be encoded in P-log. The relationships of P-log to other alternative uncertainty knowledge representation languages like [10,16,17] have been studied in [3]. Unfortunately, the existing current implementations of P-log [1,8] have exponential best case complexity, since they enumerate all possible models, even though it is known that for singly connected Bayesian Networks (polytrees) reasoning can be performed in polynomial time [14].

The contribution of this paper is the definition of modules for the P-log language, and corresponding compositional semantics as well as its probabilistic interpretation. The semantics relies on a translation to logic program modules of Oikarinen and Janhunen [13]. With this appropriate notion of P-log modules is possible to obtain possible worlds incrementally, and this can be optimized for answering to probabilistic queries in polynomial time for specific cases, using techniques inspired in the variable elimination algorithm [20].

The rest of the paper is organized as follows. Section 2 briefly summarizes P-log syntax and semantics, as well as the essential modularity results for answer set programming. Next, Section 3 is the core of the paper defining modules for P-log language and its translation into ASP modules. The subsequent section presents the module theorem and a discussion of the application of the result to Bayesian Networks. We conclude with final remarks and foreseen work.

2 Preliminaries

In this section, we review the syntax and semantics of P-log language [3], and illustrate it with an example encoding a Bayesian Network. Subsequently, the major results regarding composition of answer set semantics are presented [13]. The reader is assumed to have familiarity with (Causal) Bayesian Networks [15] and good knowledge of answer set programming. A good introduction to Bayesian Networks can be found in [19], and to answer set programming in [2,12].

2.1 P-Log Programs

P-log is a declarative language [3], based on a logic formalism for probabilistic reasoning and action, that uses answer set programming (ASP) as its logical foundation and Causal Bayesian Networks (CBNs) as its probabilistic foundation. P-log is a complex language to present in a short amount of space, and the reader is referred to [3] for full details. We will try to make the presentation self-contained for this paper, abbreviating or even neglecting the irrelevant parts, and follows closely [3].

P-log syntax. A probabilistic logic program (P-log program) Π consists of (i) a sorted signature, (ii) a declaration part, (iii) a regular part, (iv) a set of random selection rules, (v) a probabilistic information part, and (vi) a set of observations and actions. Notice that the first four parts correspond to the actual stable models' generation, and the last two define the probabilistic information.

The declaration part defines a sort c by explicitly listing its members with a statement $c = \{x_1, \ldots, x_n\}$, or by defining a unary predicate c in a program with a single answer set. An attribute a with n parameters is declared by a statement $a : c_1 \times \ldots \times c_n \to c_0$ where each c_i is a sort $(0 \leq i \leq m)$; in the case of an attributes with no parameter the syntax $a : c_0$ may be used. By $range(a)$ we denote the set of elements of sort c_0. The sorts can be understood as domain declarations for predicates and attributes used in the program, for appropriate typing of argument variables.

The regular part of a P-log program is just a set of Answer Set Programming rules (without disjunction) constructed from the usual literals in answer set programming plus attribute literals of the form $a(\bar{t}) = t_0$ (including strongly negated literals), where $\bar{t}$ is a vector of n terms and t_0 is a term, respecting the corresponding sorts in the attribute declaration. Given a sorted signature Σ we denote by $Lit(\Sigma)$ the set of literals in Σ (i.e. Σ-literals) excluding all unary atoms $c_i/1$ used for specifying sorts.

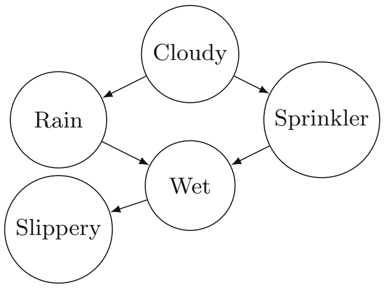
Random selection rules define random attributes and possible values for them through statements of the form $[r]\ random\,\big(a(\bar{t}) : \{X : p(X)\}\big) \leftarrow B$, expressing that if B holds then the value of $a(\overline{t})$ is selected at random from the set $\{X : p(X)\} \cap range(a)$ by experiment r, unless this value is fixed by a deliberate action, with r being a term uniquely identifying the rule. The concrete probability distribution for random attributes is conveyed by the probabilistic information part containing *pr-atoms* (probability atoms), of the form $pr_r(a(\overline{t}) = y|_c B) = v$ stating that if the value of $a(\overline{t})$ is fixed by experiment r and B holds, then the probability that r causes $a(\overline{t}) = y$ is v, with $v \in [0, 1]$. The condition B is a conjunction of literals or the default negation (not) of literals.

Finally, observations and actions are statements of the form $obs(l)$ and $do(a(\overline{t}) = y)$, respectively, where l is an arbitrary literal of the signature.

Example 1 (Wet Grass). Suppose that there are two events which could cause grass to be wet: either the sprinkler is on, or it is raining. Furthermore, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Furthermore, cloudy sky affects whether the sprinklers are on and obviously if it is raining or not. Finally, notice that, the grass being wet affects it being slippery. Then the situation can be modeled with a Bayesian network (shown in the diagram). All random variables are boolean and have no parameters; also notice how the conditional probability tables (CPTs) are encoded with pr-atoms as well as causal dependencies among random attributes. The P-log semantics will take care of completing the CPTs assuming a uniform distribution for the remaining attribute values

$boolean = \{t, f\}.$
$cloudy : boolean.$
$rain : boolean.$
$sprinkler : boolean.$
$wet : boolean.$
$slippery : boolean.$

$dangerous \leftarrow slippery = t.$



$[rc]\ random(cloudy, \{X : boolean(X)\}).$ $[rr]\ random(rain, \{X : boolean(X)\}).$
$[rsk]\ random(sprinkler, \{X : boolean(X)\}).$ $[rw]\ random(wet, \{X : boolean(X)\}).$
$[rsl]\ random(slippery, \{X : boolean(X)\}).$

$pr_{rr}(rain = t \mid_c cloudy = f) = 0.2.$ $pr_{rr}(rain = t \mid_c cloudy = t) = 0.8.$
$pr_{rsk}(sprinkler = t \mid_c cloudy = f) = 0.5.$ $pr_{rsk}(sprinkler = t \mid_c cloudy = t) = 0.1.$
$pr_{rsl}(slippery = t \mid_c wet = f) = 0.1.$ $pr_{rsl}(slippery = t \mid_c wet = t) = 0.9.$
$pr_{rw}(wet = t \mid_c sprinkler = f, rain = f) = 0.0.$
$pr_{rw}(wet = t \mid_c sprinkler = f, rain = t) = 0.9.$
$pr_{rw}(wet = t \mid_c sprinkler = t, rain = f) = 0.9.$
$pr_{rw}(wet = t \mid_c sprinkler = t, rain = t) = 0.99.$ $obs(sprinkler = t).$

Fig. 1. Bayesian Network encoded in P-log

(e.g. $cloudy = false$ will have probability 0.5). Rules can be used to extract additional knowledge from the random variables (e.g. the *dangerous* rule). In particular we will be able to query the program to determine the probability $\mathbf{P}(dangerous|sprinkler = t)$.

P-log semantics. The semantics of a P-log program Π is given by a collection of the possible sets of beliefs of a rational agent associated with Π, together with their probabilities. We refer to these sets of beliefs as possible worlds of Π. Note that due to the restriction on the signature of P-log programs the authors enforce (all sorts are finite), possible worlds of Π are always finite. The semantics is defined in two stages. First we will define a mapping of the logical part of Π into its Answer Set Programming counterpart, $\tau(\Pi)$. The answer sets of $\tau(\Pi)$ will play the role of possible worlds of Π. The probabilistic part of Π is used to define a measure over the possible worlds, and from these the probabilities of formulas can be determined. The set of all possible worlds of Π will be denoted by $\Omega(\Pi)$.

The Answer Set Program $\tau(\Pi)$ is defined in the following way, where capital letters are variables being grounded with values from the appropriate sort; to reduce overhead we omit the sort predicates for variables in the program rules. It is also assumed that any attribute literal $a(\bar{t}) = y$ is replaced consistently by the predicate $a(\bar{t}, y)$ in the translated program $\tau(\Pi)$, constructed as follows:

τ_1: For every sort $c = \{x_1, \ldots, x_n\}$ of Π, $\tau(\Pi)$ contains $c(x_1), \ldots, c(x_n)$. For any remaining sorts defined by an ASP program T in Π, then $T \subseteq \tau(\Pi)$.

τ_2: Regular part:

- **(a)** For each rule r in the regular part of Π, $\tau(\Pi)$ contains the rule obtained by replacing each occurrence of an atom $a(\bar{t}) = y$ in r by $a(\bar{t}, y)$.
- **(b)** For each attribute term $a(\bar{t})$, $\tau(\Pi)$ contains $\neg a(\bar{t}, Y_1) :\!- a(\bar{t}, Y_2),\ Y_1 \neq Y_2$ guaranteeing that in each answer set $a(\bar{t})$ has at most one value.

τ_3: Random selections:

- **(a)** For an attribute $a(\bar{t})$, we have the rule: $intervene(a(\bar{t})) :\!- do(a(\bar{t}, Y))$. Intuitively, the value of $a(\bar{t})$ is fixed by a deliberate action, i.e. $a(\bar{t})$ will not be considered random in possible worlds satisfying $intervene(a(\bar{t}))$.
- **(b)** Random selection $[r]$ $random\left(a(\bar{t}) : \{X : p(X)\}\right) \leftarrow B$ is translated into rule $1\{\ a(\bar{t}, Z)\ :\ poss(r, a(\bar{t}), Z)\ \}1 :\!- B$, not $intervene(a(\bar{t}))$ and $poss(r, a(\bar{t}), Z) :\!- c_0(Z), p(Z), B$,not $intervene(a(\bar{t}))$ with $range(a) = c_0$.

τ_4: Each pr-atom $pr_r(a(\bar{t}) = y|_c B) = v$ is translated into the following rule $pa(r, a(\bar{t}, y), v) :\!- poss(r, a(\bar{t}), y), B$ of $\tau(\Pi)$ with $pa/3$ a reserved predicate.

τ_5: $\tau(\Pi)$ contains actions and observations of Π.

τ_6: For each Σ-literal l , $\tau(\Pi)$ contains the constraint $:\!- obs(l), not\ \ l$.

τ_7: For each atom $a(t) = y$, $\tau(\Pi)$ contains the rule $a(\bar{t}, y) :\!- do(a(\bar{t}, y))$.

In the previous construction, the two last rules guarantee respectively that no possible world of the program fails to satisfy observation l, and that the atoms made true by the action are indeed true. The introduction of the reserved predicates $poss/3$ and $pa/3$ is a novel contribution to the transformation, and simplifies the presentation of the remaining details of the semantics.

P-log semantics assigns a probability measure for each world W, i.e. answer set, of $\tau(\Pi)$ from the causal probability computed deterministically from instances of predicates $poss/3$ and $pa/3$ true in the world. Briefly, if an atom $pa(r, a(\bar{t}, y), v)$ belongs to W then the causal probability $\mathbf{P}(W, a(\bar{t}) = y)$ is v, i.e. the assigned probability in the model. The possible values for $a(\bar{t})$ are collected by $poss(r, a(\bar{t}, y_k))$ instances true in W, and P-log semantics assigns a (default) causal probability for non-assigned values, by distributing uniformly the non-assigned probability among these non-assigned values. The details to make this formally precise are rather long [3] but for our purpose it is enough to understand that for each world W the causal probability $\sum_{y \in range(a)} \mathbf{P}(W, a(\bar{t}) = y) = 1.0$, for each attribute term with at least a possible value. These probability calculations can be encoded in ASP rules with aggregates ($\#sum$ and $\#count$), making use of only $pa/3$ and $poss/3$ predicates.

Example 2. Consider the P-log program of Example 1. This program has 16 possible worlds (notice that $sprinkler = t$ is observed). One possible world is W_1 containing:

$$\begin{array}{cccccc} cloudy(f) & rain(f) & wet(t) & sprinkler(t) & slippery(t) & obs(sprinkler(t)) \\ \neg cloudy(t) & \neg rain(t) & \neg wet(f) & \neg sprinkler(f) & \neg slippery(f) & dangerous \end{array}$$

Furthermore the following probability assignment atoms are true in that model:

$$\begin{array}{lll} poss(rc, cloudy, t) & poss(rc, cloudy, f) & \\ poss(rr, rain, t) & poss(rr, rain, f) & pa(rr, rain(t), 0.2) \\ poss(rsk, sprinkler, t) & poss(rsk, sprinkler, f) & pa(rsk, sprinkler(t), 0.5) \\ poss(rsl, slippery, t) & poss(rsl, slippery, f) & pa(rsl, slippery(t), 0.9) \\ poss(rw, wet, t) & poss(rw, wet, f) & pa(rw, wet(t), 0.9) \end{array}$$

which determines the following causal probabilities in the model

$$\begin{array}{ll} \mathbf{P}(W_1, cloudy = t) = \frac{1.0-0.0}{2} = 0.5 & \mathbf{P}(W_1, cloudy = f) = \frac{1.0-0.0}{2} = 0.5 \\ \mathbf{P}(W_1, rain = t) = 0.2 & \mathbf{P}(W_1, rain = f) = \frac{1.0-0.2}{1} = 0.8 \\ \mathbf{P}(W_1, sprinkler = t) = 0.5 & \mathbf{P}(W_1, sprinkler = f) = \frac{1.0-0.5}{1} = 0.5 \\ \mathbf{P}(W_1, wet = t) = 0.9 & \mathbf{P}(W_1, wet = f) = \frac{1.0-0.9}{1} = 0.1 \\ \mathbf{P}(W_1, slippery = t) = 0.9 & \mathbf{P}(W_1, slippery = f) = \frac{1.0-0.9}{1} = 0.1 \end{array}$$

The authors define next the measure μ_Π induced by a P-log program Π:

Definition 1 (Measure). *Let W be a possible world of a P-log program Π. The unnormalized probability of W induced by Π is $\hat{\mu}_\Pi(W) = \prod_{a(\bar{t},y) \in W} \boldsymbol{P}(W, a(\bar{t}) = y)$ where the product is taken over atoms for which $\boldsymbol{P}(W, a(\bar{t}) = y)$ is defined.*

If Π is a P-log program having at least one possible world with nonzero unnormalized probability, then the measure, $\mu_\Pi(W)$, of a possible world W induced by Π is the normalized probability of W divided by the sum of the unnormalized *probabilities of all possible worlds of Π, i.e., $\mu_\Pi(W) = \frac{\hat{\mu}_\Pi(W)}{\sum_{W_i \in \Omega} \hat{\mu}_\Pi(W_i)}$. When the program Π is clear from the context we may simply write $\hat{\mu}$ and μ instead of $\hat{\mu}_\Pi$ and μ_Π respectively.*

Example 3. For world W_1 of Example 2 we obtain that:

$$\begin{aligned}\hat{\mu}(W_1) = {} & \mathbf{P}(W_1, cloudy = f) \times \mathbf{P}(W_1, rain = f) \times \mathbf{P}(W_1, wet = t) \times \\ & \mathbf{P}(W_1, sprinkler = t) \times \mathbf{P}(W_1, slippery = t) = \\ = {} & 0.5 \times 0.8 \times 0.9 \times 0.5 \times 0.1 = 0.018\end{aligned}$$

Since the sum of the unconditional probability measure of all the sixteen worlds of the P-log program is 0.3 then we obtain that $\mu(W_1) = 0.06$.

The truth and falsity of propositional formulas with respect to possible worlds are defined in the standard way. A formula A, true in W, is denoted by $W \vdash A$.

Definition 2 (Probability). *Suppose Π is a P-log program having at least one possible world with nonzero unnormalized probability. The probability, $P_\Pi(A)$, of a formula A is the sum of the measures of the possible worlds of Π on which A is true, i.e. $P_\Pi(A) = \sum_{W \vdash A} \mu_\Pi(W)$.*

Conditional probability in P-log is defined in the usual way by $P_\Pi(A|B) = P_\Pi(A \wedge B)/P_\Pi(B)$ whenever $P_\Pi(B) \neq 0$, where the set B stands for the conjunction of its elements. Moreover, under certain consistency conditions on P-log programs Π, formulas A, and a set of literals B such that $P_\Pi(B) \neq 0$, it is the case that $P_\Pi(A|B) = P_{\Pi \cup obs(B)}(A)$. See the original work [3] where the exact consistency conditions are stated, which are assumed to hold subsequently.

2.2 Modularity in Answer Set Programming

The modular aspects of Answer Set Programming have been clarified in recent years [13,5] describing how and when can two program parts (modules) be composed together. In this paper, we will make use of Oikarinen and Janhunen's logic program modules defined in analogy to [6]:

Definition 3 (Module [13]). *A logic program module $\mathbb{P}$ is $\langle R, I, O, H \rangle$:*

1. *R is a finite set of rules;*
2. *I, O, and H are pairwise disjoint sets of input, output, and hidden atoms;*
3. *$At(R) \subseteq At(\mathbb{P})$ defined by $At(\mathbb{P}) = I \cup O \cup H$; and*
4. *$head(R) \cap I = \emptyset$.*

The atoms in $At_v(\mathbb{P}) = I \cup O$ are considered to be visible and hence accessible to other modules composed with $\mathbb{P}$ either to produce input for $\mathbb{P}$ or to make use of the output of $\mathbb{P}$. The hidden atoms in $At_h(\mathbb{P}) = H = At(\mathbb{P}) \backslash At_v(\mathbb{P})$ are used to formalize some auxiliary concepts of $\mathbb{P}$ which may not be sensible for other modules but may save space substantially. The condition $head(R) \cap I = \emptyset$ ensures that a module may not interfere with its own input by defining input atoms of I in terms of its rules. Thus, input atoms are only allowed to appear as conditions in rule bodies. The answer set semantics is generalized to cover modules by introducing a generalization of the Gelfond-Lifschitz's fixpoint definition. In addition to negative default literals (i.e. not l), also literals involving input atoms are used in the stability condition.

Definition 4. *An interpretation $M \subseteq At(\mathbb{P})$ is an answer set of an ASP program module $\mathbb{P} = \langle R, I, O, H\rangle$, if and only if $M = LM\left(R_I^M \cup \{a.|a \in M \cap I\}\right)$*[1] *The set of answer sets of module $\mathbb{P}$ is denoted by $AS(\mathbb{P})$.*

Given two modules $\mathbb{P}_1 = \langle R_1, I_1, O_1, H_1\rangle$ and $\mathbb{P}_2 = \langle R_2, I_2, O_2, H_2\rangle$, their composition $\mathbb{P}_1 \oplus \mathbb{P}_2$ is defined when their output signatures are disjoint, that is, $O1 \cap O2 = \emptyset$, and they respect each others hidden atoms, i.e. $H_1 \cap At(\mathbb{P}_2) = \emptyset$ and $H_2 \cap At(\mathbb{P}_1) = \emptyset$. Then their composition is $\mathbb{P}_1 \oplus \mathbb{P}_2 = \langle R_1 \cup R_2, (I_1 \cup I_2)\backslash(O_1 \cup O_2), O_1 \cup O_2, H_1 \cup H_2\rangle$. However, the conditions given for $\oplus$ are not enough to guarantee compositionality in the case of stable models:

Definition 5. *Given modules $\mathbb{P}_1, \mathbb{P}_2 \in M$, their join is $\mathbb{P}_1 \sqcup \mathbb{P}_2 = \mathbb{P}_1 \oplus \mathbb{P}_2$ provided that (i) $\mathbb{P}_1 \oplus \mathbb{P}_2$ is defined and (ii) $\mathbb{P}_1$ and $\mathbb{P}_2$ are mutually independent*[2]*.*

Theorem 1 (The module theorem). *If $\mathbb{P}_1, \mathbb{P}_2$ are modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined, then $AS(\mathbb{P}_1 \sqcup \mathbb{P}_2) = AS(\mathbb{P}_1) \bowtie AS(\mathbb{P}_2)$, where $AS(\mathbb{P}_1) \bowtie AS(\mathbb{P}_2) = \{M_1 \cup M_2 \mid M_1 \in AS(\mathbb{P}_1), M_2 \in AS(\mathbb{P}_2),$ and $M_1 \cap At_v(\mathbb{P}_2) = M_2 \cap At_v(\mathbb{P}_1)\}$.*

The module theorem also straightforwardly generalizes for a collection of modules because the module union operator $\sqcup$ is commutative, associative, and idempotent [13].

3 P-Log Modules

In this section, we define the notion of P-log modules and its semantics via a translation into logic program modules. Its probabilistic interpretation is provided by a conditional probability measure. In what follows, we assume that different modules may have different sorted signatures Σ.

Definition 6. *A P-log module $\mathfrak{P}$ over Σ is a structure $\langle \Pi, Rin, Rout\rangle$ such that:*

1. *Π is a P-log program (possibly with observations and actions);*
2. *Rin is a set of ground attribute literals $a(\bar{t}) = y$, of random attributes declared in Π such that $y \in range(a)$;*
3. *Rout is a set of ground Σ-literals, excluding attribute literals $a(\bar{t}) = y \in Rin$;*
4. *Π does not contain rules for any attribute $a(\bar{t})$ occurring in attribute literals of Rin, i.e. no random selection rule for $a(\bar{t})$, no regular rule with head $a(\bar{t}) = y$ nor a pr-atom for $a(\bar{t}) = y$.*

[1] Note that R_I^M is a reduct allowing weighted and choice rules, and LM is an operator returning the least model of the positive program argument.

[2] There are no positive cyclic dependencies among rules in different modules.

The notion of P-log module is quite intuitive. First, the P-log program specifies the possible models and corresponding probabilistic information as before, and may include regular rules. However, the P-log module is parametric on a set of attribute terms *Rin*, which can be understood as the module's parent random variables. *Rout* specifies the random attributes which are visible as well as other derived logical conclusions. The last condition ensures that there is no interference between input and output random attributes.

The semantics of a P-log module is defined again in two stages. The possible worlds of a P-log module are obtained from the Answer Sets of a corresponding logic programming module. For simplifying definitions we assume that the isomorphism of attribute literals $a(\overline{t}) = y$ with $a(\overline{t}, y)$ instances is implicitly applied when moving from the P-log side to ASP side, and vice-versa.

Definition 7. *Consider a P-log module* $\mathfrak{P} = \langle \Pi, Rin, Rout \rangle$ *over signature* Σ, *and let* $\mathbb{P}(\mathfrak{P}) = \langle R_{\mathfrak{P}}, I_{\mathfrak{P}}, O_{\mathfrak{P}}, H_{\mathfrak{P}} \rangle$ *be the corresponding ASP module such that:*

1. $R_{\mathfrak{P}}$ *is* $\tau(\Pi) \cup \{{:}{-}a(\overline{t}, y_1), a(\overline{t}, y_2) \mid a(\overline{t}) = y_1$ *and* $a(\overline{t}) = y_2$ *in Rin s.t.* $y_1 \neq y_2\} \cup \{{:}{-}\ not\ hasval_{\mathfrak{P}}(a(\overline{t}))\} \cup \{hasval_{\mathfrak{P}}(a(\overline{t})){:}{-}a(\overline{t}, y) \mid a(\overline{t}) = y \in Rin\}$, *where predicates defining sorts have been renamed apart;*
2. *The set of input atoms* $I_{\mathfrak{P}}$ *of* $\mathbb{P}(\mathfrak{P})$ *is Rin.*
3. *The set of output atoms* $O_{\mathfrak{P}}$ *of* $\mathbb{P}(\mathfrak{P})$ *is Rout union all instances of* $pa/3$ *and* $poss/3$ *predicates of random attributes in Rout;*
4. *The set of hidden atoms* $H_{\mathfrak{P}}$ *of* $\mathbb{P}(\mathfrak{P})$ *is formed by* $hasval_{\mathfrak{P}}/1$ *instances for attribute literals in Rin, the* Σ*-literals not included in the output or input atoms of* $\mathbb{P}(\mathfrak{P})$*, with all sort predicates renamed apart.*

The possible models of $\mathfrak{P}$ *are* $\Omega(\mathfrak{P}) = \{M \cap (Rin \cup Rout) \mid M \in AS(\mathbb{P}(\mathfrak{P}))\}$. *The name* $hasval_{\mathfrak{P}}$ *is local to* $\mathfrak{P}$ *and not occurring elsewhere.*

The necessity of having sort predicates renamed apart is essential to avoid name clashes between different modules using the same sort attributes. Equivalently, the program can be instantiated, and all sort predicates removed. The extra integrity constraints in $R_{\mathfrak{P}}$ discard models where a random attribute has not exactly one assigned value. The set of input atoms in $\mathfrak{P}$ is formed by the random attribute literals in *Rin*. The set of output atoms includes all the instances of $pa/3$ and $poss/3$ in order to be possible to determine the causal probabilities in each model. By convention, all the remaining literals are hidden. A significant difference to the ordinary ASP modules is that the set of possible models are projected with respect to the visible literals, discarding hidden information in the models. The semantics of a P-log module is defined by probabilistic conditional measures:

Definition 8. *Consider a P-log module* $\mathfrak{P} = \langle \Pi, Rin, Rout \rangle$ *over signature* Σ. *Let* E *be any subset of* $Rin \cup Rout$, *and* W *be a possible world of P-log module* $\mathfrak{P}$. *If* $E \subseteq W$ *then the conditional unnormalized probability of* W *given* E *induced by* $\mathfrak{P}$ *is*

$$\hat{\mu}_{\mathfrak{P}}(W|E) = \sum_{M_i \in AS(\mathbb{P}(\mathfrak{P}))\ s.t.\ M_i \cap (Rin \cup Rout) = W} \quad \prod_{a(\overline{t}, y) \in M_i} \boldsymbol{P}(M_i, a(\overline{t}) = y)$$

where the product is taken over atoms for which $\boldsymbol{P}(M_i, a(\bar{t}) = y)$ *is defined in* M_i*. Otherwise,* $E \not\subseteq W$ *and we set* $\hat{\mu}_{\mathfrak{P}}(W|E) = 0.0$.

If there is at least one possible world with nonzero unnormalized conditional probability, for a particular E, then the conditional probability measure $\mu_{\mathfrak{P}}(.|E)$ *is determined, and* $\mu_{\mathfrak{P}}(W|E)$ *for a possible world W given E induced by* $\mathfrak{P}$ *is*

$$\mu_{\mathfrak{P}}(W|E) = \frac{\hat{\mu}_{\mathfrak{P}}(W|E)}{\sum_{W_i \in \Omega(\mathfrak{P})} \hat{\mu}_{\mathfrak{P}}(W_i|E)} = \frac{\hat{\mu}_{\mathfrak{P}}(W|E)}{\sum_{W_i \in \Omega(\mathfrak{P}) \wedge E \subseteq W_i} \hat{\mu}_{\mathfrak{P}}(W_i|E)}$$

When the P-log module $\mathfrak{P}$ *is clear from the context we may simply write* $\hat{\mu}(W|E)$ *and* $\mu(W|E)$ *instead of* $\hat{\mu}_{\mathfrak{P}}(W|E)$ *and* $\mu_{\mathfrak{P}}(W|E)$ *respectively.*

The important remark regarding the above definition is that a possible world W of the P-log module can correspond to several models (the answer sets M_i) of the underlying answer set program, since hidden atoms have been projected out. This way, we need to sum the associated unconditional measures of the ASP models which originate (or contribute to) W. The attentive reader should have noticed that for any world W the unconditional probability measure $\hat{\mu}_{\mathfrak{P}}(W|E)$, for any $E \subseteq W \cap (Rin \cup Rout)$ is identical to $\hat{\mu}_{\mathfrak{P}}(W|W \cap (Rin \cup Rout))$, and zero elsewhere. So, in practice each world just requires one real value to obtain all the conditional probability measures.

Example 4. Construct P-log module 𝔖𝔭𝔯𝔦𝔫𝔨𝔩𝔢𝔯 from Example1 whose input atoms are $\{cloudy = t, cloudy = f\}$ and output atoms are $\{sprinkler = t, sprinkler = f\}$. The P-log program of 𝔖𝔭𝔯𝔦𝔫𝔨𝔩𝔢𝔯 (with the observation removed) is

$boolean = \{t, f\}. \quad cloudy : boolean. \quad sprinkler : boolean.$
$[rs]\, random(sprinkler, \{X : boolean(X)\}).$
$pr_{rs}(sprinkler = t \mid_c cloudy = f) = 0.5.$
$pr_{rs}(sprinkler = t \mid_c cloudy = t) = 0.1.$

For which, the corresponding ASP program in module $\mathbb{P}$(𝔖𝔭𝔯𝔦𝔫𝔨𝔩𝔢𝔯) is:

$hasval(cloudy) :- cloudy(t). \quad hasval(cloudy) :- cloudy(f).$
$:- \text{not } hasval(cloudy). \quad :- cloudy(t), cloudy(f).$

$-sprinkler(Y1) :- sprinkler(Y2), Y1! = Y2, boolean(Y1), boolean(Y2).$

$1\{sprinkler(Z) : poss(rsk, sprinkler, Z)\}1 :- \text{not } intervene(sprinkler).$
$poss(rsk, sprinkler, Z) :- boolean(Z), \text{not } intervene(sprinkler).$
$intervene(sprinkler) :- do(sprinkler(Y)), boolean(Y).$

$pa(rsk, sprinkler(t), 0.1) :- poss(rsk, sprinkler, t), cloudy(t).$
$pa(rsk, sprinkler(t), 0.5) :- poss(rsk, sprinkler, t), cloudy(f).$

$:- obs(sprinkler(t)), \text{not } sprinkler(t). \quad :- obs(sprinkler(f)), \text{not } sprinkler(f).$

Module $\mathbb{P}(\mathfrak{Sprinkler})$ has four answer sets all containing both $poss(rsk, sprinkler, t)$ and $poss(rsk, sprinkler, f)$, and additionally:

$$\begin{aligned}
M_1 &= \{sprinkler(t),\ \ cloudy(t),\ \ pa(rsk, sprinkler(t), 0.1)\}\\
M_2 &= \{sprinkler(f),\ \ cloudy(t),\ \ pa(rsk, sprinkler(t), 0.1)\}\\
M_3 &= \{sprinkler(t),\ \ cloudy(f),\ \ pa(rsk, sprinkler(t), 0.5)\}\\
M_4 &= \{sprinkler(f),\ \ cloudy(f),\ \ pa(rsk, sprinkler(t), 0.5)\}
\end{aligned}$$

The first two correspond to possible worlds $W_1 = \{sprinkler(t),\ \ cloudy(t)\}$ and $W_2 = \{sprinkler(f),\ \ cloudy(t)\}$ where $cloudy = t$. So, $\hat{\mu}(W_1|\{cloudy(t)\}) = 0.1$ and $\hat{\mu}(W_2|\{cloudy(t)\}) = 0.9$ and $\hat{\mu}(W_3|\{cloudy(t)\}) = \hat{\mu}(W_4|\{cloudy(t)\}) = 0.0$. Since the sum of the unconditional probability measures for all world totals 1.0, then the normalized measure coincides with the unnormalized one for the particular evidence $\{cloudy = t\}$.

Definition 9 (Conditional Probability). *Suppose $\mathfrak{P}$ is a P-log module and $E \subseteq Rin \cup Rout$ for which $\mu_\Pi(.|E)$ is determined. The probability, $P_\mathfrak{P}(A|E)$, of a formula A over literals in Rout, is the sum of the conditional probability measures of the possible worlds of $\mathfrak{P}$ on which A is true, i.e. $P_\mathfrak{P}(A|E) = \sum_{W \vdash A} \mu_\mathfrak{P}(W|E)$.*

The following theorem shows that P-log modules generalize appropriately the notion of conditional probability of P-log programs.

Theorem 2. *Let Π be P-log program Π. Consider the P-log module $\mathfrak{P} = \langle \Pi, \{\}, Lit(\Sigma) \rangle$ then for any set $B \subseteq Lit(\Sigma)$ such that $P_\Pi(B) \neq 0$ then $P_\Pi(A|B) = P_\mathfrak{P}(A|B)$.*

A P-log module corresponds to the notion of factor introduced by [20] in their variable elimination algorithm. The difference is that P-log modules are defined declaratively by a logic program with associated probabilistic semantics, instead of just matrix of values for each possible combination of parameter variables.

4 P-Log Module Theorem

This section provides a way of composing P-log modules and presents the corresponding module theorem. The composition of a P-log module mimics syntactically the composition of an answer set programming module, with similar pre-conditions:

Definition 10 (P-log module composition). *Consider P-log modules $\mathfrak{P}_1 = \langle \Pi_1, Rin_1, Rout_1 \rangle$ over signature Σ_1, and $\mathfrak{P}_2 = \langle \Pi_2, Rin_2, Rout_2 \rangle$ over signature Σ_2, such that:*

1. *$Rout_1 \cap Rout_2 = \emptyset$*
2. *$(Lit(\Sigma_1) \setminus (Rin_1 \cup Rout_1)) \cap Lit(\Sigma_2) = Lit(\Sigma_1) \cap (Lit(\Sigma_2) \setminus (Rin_2 \cup Rout_2)) = \emptyset$*
3. *The sorts of Σ_1 and Σ_2 coincide and are defined equivalently in Π_1 and Π_2.*

The composition of $\mathfrak{P}_1$ *with* $\mathfrak{P}_2$ *is the P-log module* $\mathfrak{P}_1 \oplus \mathfrak{P}_2 = \langle \Pi_1 \cup \Pi_2, (Rin_1 \cup Rin_2) \setminus (Rout_1 \cup Rout_2), (Rout_1 \cup Rout_2) \rangle$ *over signature* $\Sigma_1 \cup \Sigma_2$*.*

The join $\mathfrak{P}_1 \sqcup \mathfrak{P}_2 = \mathfrak{P}_1 \oplus \mathfrak{P}_2$ *is defined in this case whenever additionally there are no dependencies (positive or negative) among modules.*

The first condition forbids the composition of modules having a common output literal, while the second one forbids common hidden atoms (except possibly the sort predicate instances). We avoid joining modules having both negative and positive dependencies.

The compositionality result for P-log modules is more intricate since besides the compositional construction of possible worlds, it is also necessary to ensure compositionality for the underlying conditional probability measures induced by the joined module:

Theorem 3 (P-log Module Theorem). *Consider two P-log modules* $\mathfrak{P}_1$ *and* $\mathfrak{P}_2$ *such that their join* $\mathfrak{P}_1 \sqcup \mathfrak{P}_2$ *is defined. Then* $\Omega(\mathfrak{P}_1 \sqcup \mathfrak{P}_2) = \Omega(\mathfrak{P}_1) \bowtie \Omega(\mathfrak{P}_2)$ *with* $\Omega(\mathfrak{P}_1) \bowtie \Omega(\mathfrak{P}_2) = \{W_1 \cup W_2 \mid W_1 \in \Omega(\mathfrak{P}_1), W_2 \in \Omega(\mathfrak{P}_1),$ *and* $W_1 \cap (Rin_2 \cup Rout_2) = W_2 \cap (Rin_1 \cup Rout_1)\}$*.*

Let $E = E_1 \cup E_2$ *where* $E_1 = E \cap (Rin_1 \cup Rout_1)$ *and* $E_2 = E \cap (Rin_2 \cup Rout_2)$*. Then,* $\hat{\mu}_{\mathfrak{P}_1 \sqcup \mathfrak{P}_2}(W|E) = \hat{\mu}_{\mathfrak{P}_1}(W_1|E_1) \times \hat{\mu}_{\mathfrak{P}_2}(W_2|E_2)$ *with* $W = W_1 \cup W_2$ *such that* $W \in \Omega(\mathfrak{P}_1 \sqcup \mathfrak{P}_2)$*,* $W_1 \in \Omega(\mathfrak{P}_1)$ *and* $W_2 \in \Omega(\mathfrak{P}_2)$*.*

Notice that the P-log module theorem is defined only in terms of the unnormalized conditional probability measures. The normalized ones can be obtained as in the previous case dividing by the sum of unconditional measure of all worlds given the evidence. Again, we just have to consider one value for each world (i.e. when evidence is maximal).

The application to Bayesian Networks is now straightforward. First, each random variable in a Bayesian Network is captured by a P-log module having the corresponding attribute literals of the random variable as output literals, and input literals are all attribute literals obtainable from parent variables. The conditional probability tables are represented by pr-atoms, as illustrated before in Example 1. P-log module composition inherits associativity and commutativity from ASP modules, and thus P-log modules can be joined in arbitrary ways since there are no common output atoms, and there are no cyclic dependencies.

The important remark is that a P-log module is an extension of the notion of factor used in the variable elimination algorithm [20]. We only need a way to eliminate variables from a P-log module in order to simulate the behaviour of the variable elimination algorithm, but this is almost trivial:

Definition 11 (Eliminate operation). *Consider a P-log module* $\mathfrak{P} = \langle \Pi, Rin, Rout \rangle$ *over signature* Σ*, and a subset of attribute literals* $S \subseteq Rout$*. Then, P-log module* $Elim(\mathfrak{P}, S) = \langle \Pi, Rin, Rout \setminus S \rangle$ *eliminates (hides) from* $\mathfrak{P}$ *the attribute literals in* S*.*

By hiding all attribute literals of a given random variable, we remove the random attribute from the possible worlds (as expected), summing away corresponding original possible worlds. By applying the composition of P-log modules and eliminate operations by the order they are performed by the variable elimination algorithm, the exact behaviour of the variable elimination algorithm is attained. Thus, for the case of polytrees represented in P-log, we can do reasoning with P-log in polynomial time.

5 Conclusions and Future Work

We present the first approach in the literature to modularize P-log programs and to make their composition incrementally by combining compatible possible worlds and multiplying corresponding unnormalized conditional probability measures. A P-log module corresponds to a factor of the variable elimination algorithm [20,18], clarifying and improving the relationship of P-log with traditional Bayesian Network approaches. By eliminating variables in P-log modules we may reduce the space and time necessary to make inference in P-log, in contrast with previous algorithms [1,8] which require always enumeration of the full possible worlds (which are exponential on the number of random variables) and repeat calculations. As expected, it turns out that the general case of exact inference is intractable, so we must consider methods for approximate inference.

As future work, we intend to fully describe the inference algorithm obtained from the compositional semantics of P-log modules and relate it formally with the variable elimination algorithm. Furthermore we expect that the notion of P-log module may also help to devise approximate inference methods, e.g. by extending sampling algorithms, enlarging the applicability of P-log which is currently somehow restricted. Finally, we hope to generalize the P-log language to consider other forms of uncertainty representation like belief functions, possibility measures or even plausibility measures [9].

References

1. Anh, H.T., Kencana Ramli, C.D.P., Damásio, C.V.: An implementation of extended P-log using XASP. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 739–743. Springer, Heidelberg (2008)
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
3. Baral, C., Gelfond, M., Rushton, J.N.: Probabilistic reasoning with answer sets. TPLP 9(1), 57–144 (2009)
4. Baral, C., Hunsaker, M.: Using the probabilistic logic programming language P-log for causal and counterfactual reasoning and non-naive conditioning. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, pp. 243–249. Morgan Kaufmann Publishers Inc., San Francisco (2007)
5. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular Nonmonotonic Logic Programming Revisited. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 145–159. Springer, Heidelberg (2009)
6. Gaifman, H., Shapiro, E.: Fully abstract compositional semantics for logic programs. In: POPL 1989: Proc. of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 134–142. ACM, New York (1989)

7. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Proceedings of the Fifth International Conference and Symposium (ICLP/SLP), pp. 1070–1080. MIT Press, Cambridge (1988)
8. Gelfond, M., Rushton, N., Zhu, W.: Combining logical and probabilistic reasoning. In: Proc. of AAAI 2006 Spring Symposium: Formalizing AND Compiling Background Knowledge AND Its Applications to Knowledge Representation AND Question Answering, pp. 50–55. AAAI Press, Menlo Park (2006)
9. Halpern, J.Y.: Reasoning about Uncertainty. The MIT Press, Cambridge (2005)
10. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic symbolic model checker. In: Kemper, P. (ed.) Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems, pp. 7–12 (September 2001)
11. Lifschitz, V.: Twelve definitions of a stable model. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 37–51. Springer, Heidelberg (2008)
12. Lifschitz, V.: What is answer set programming? In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1594–1597. MIT Press, Cambridge (2008)
13. Oikarinen, E., Janhunen, T.: Achieving compositionality of the stable model semantics for smodels programs. Theory Pract. Log. Program. 8(5-6) (2008)
14. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco (1988)
15. Pearl, J.: Causality: Models, Reasoning and Inference. Cambridge Univ. Press, Cambridge (2000)
16. Pfeffer, A., Koller, D.: Semantics and inference for recursive probability models. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 538–544. AAAI Press, Menlo Park (2000)
17. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artif. Intell. 94, 7–56 (1997)
18. Poole, D., Zhang, N.L.: Exploiting contextual independence in probabilistic inference. J. Artif. Int. Res. 18, 263–313 (2003)
19. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall, Englewood Cliffs (2010)
20. Zhang, N., Poole, D.: A simple approach to Bayesian network computations. In: Proceedings of the Tenth Canadian Conference on Artificial Intelligence, pp. 171–178 (1994)