Expressing Probabilistic Graphical Models in RCC

Cristina Cornelio
IBM T.J. Watson Research Center
Yorktown Heights, New York, U.S.
ccornel@us.ibm.com

Vijay Saraswat
IBM T.J. Watson Research Center
Yorktown Heights, New York, U.S.
vijay@saraswat.org

Abstract

The purpose of this paper is to show the expressiveness of two different formalisms that combine logic and probabilistic reasoning: Stochastic Logic Programs (SLPs) and Probabilistic Concurrent Constraint Programs (PCCs). We analyse the relation between the two and show that we are able to express, using PCC programs, some of the main probabilistic graphical models: Bayesian Networks, Markov random fields, Markov chains, Hidden Markov models, Stochastic Context Free Grammars and Markov Logic Networks. We express this last framework also in SLPs.

Introduction

Logical reasoning is a well studied and developed area of research, but in recent years it has become evident that logical reasoning is not enough to represent in a realistic way real-world events and situations. Often in a real-world context it is necessary to be able to handle noisy data, exceptions or uncertainty. We need to provide the most likelihood explanations or consequences of our events, instead of a pure logical reasoning that maps to zero probability an event that violates even just a single rule. Therefore there has been a significant interest in extending logic representations with probability theory. Many frameworks have been developed, the main are: Stochastic Logic programs (Muggleton 1996; Cussens 2000) that define a probability distribution over the proofs for a specific goal, labelling each logical rule with a parameter proportional or coincident to the probability that that rule is true; Markov Logic networks (Richardson and Domingos 2006) that correspond to a probabilistic template for Markov networks (Markov Random Fields); Problog frameworks (Raedt, Kimmig, and Toivonen 2007) that associate weights to Prolog rules; Probabilistic Soft Logic (Bach et al. 2015) that uses soft truth values: each value in the interval [0, 1] instead of 0 (false) and 1 (true) only; Probabilistic Concurrent Constraint programs (Gupta, Jagadeesan, and Saraswat 1997) that are a probabilistic extension of Concurrent Constraint programs allowing the explicit definition of random variables; and many others as Bayesian logic programs (BLPs) (Raedt and Kersting 2003; Kersting and De Raedt 2001), Logic Programs with Annotated Disjunctions (LPADs) (Vennekens et al. 2004), CP-logic (Vennekens, Denecker, and Bruynooghe 2009), Independent Choice Logic (ICL) (Poole 1993; 1997), Programming in Statistical Modelling (PRISM) (Sato 1995; Sato and Kameya 2001), etc.

The two main tasks performed on these frameworks are inference and learning. In this context inference consists in evaluating the probability distribution defined by a program and learning consists instead in determine the parameters that define the probability distributions directly from data. Both are particularly challenging problems in probabilistic and statistical programming. In many of these models the inference task is based on sampling, where the samples represent the single variables or partial worlds, for which there exists efficient techniques well known in the literature.

In this paper we will focus on Stochastic Logic programs (Cussens 2000; Muggleton 1996) and Probabilistic Concurrent Constraint programs (Gupta, Jagadeesan, and Saraswat 1997) since the underline non-probabilistic version of them are closely related; moreover they are fragment of RCC (Jagadeesan, Nadathur, and Saraswat 2005). Therefore this work is a study of the expressiveness of the RCC model. We will show that we are able to express, using PCC programs, some of the main probabilistic models: Bayesian Networks (D’Ambrosio 1999; Dechter 1999; Russell and Norvig 1994), Markov random fields (Kindermann and Laurie 1980), Markov chains (Baum and Petrie 1966), Hidden Markov Models (Baum and Petrie 1966), Stochastic Context Free Grammars (Eddy and Durbin 1994; Sakakibara et al. 1994) and Markov Logic networks (Richardson and Domingos 2006). We express this last framework also in SLPs.

The paper is organised as follows: We provide a background section with the basic notions about SLP and PCC. Then, in Section “Relation between SLP and PCC” we show how to encode a SLP program to a PCC agent and viceversa. In the main Section “Relation to probabilistic frameworks” we formalise respectively Bayesian networks, Markov Random Fields, Markov Chains, Hidden Markov Models and Probabilistic SFGs in PCC and Markov Logic networks in in SLP and PCC. We conclude providing some final remarks and some possible future directions.
Background

In what follow we will restrict on finitely termination queries avoiding infinite execution programs.

SLP

Stochastic logic programs (SLPs) have been introduced by Muggleton (Muggleton 1996) and subsequently extended by Cussens (Cussens 2000) as a probabilistic extension of Logic Programs. SLPs can be seen as a generalisation of HMMs, SCFGs and Markov networks as has been shown by Muggleton in (Muggleton 1996). A stochastic logic program is a logic program, a collection of logical rules, in which some of them are stochastic clauses. A stochastic clause is defined as $p : B \leftarrow A_1, A_2, \ldots, A_n$ where the atom $B$ is the head, the atoms $A_1, \ldots, A_n$ are the body and $p$ is a non-negative number proportional to the probability associated with that clause. If all the parameters $l$ associated to the clauses belong to the interval $[0, 1]$ and all the parameters for clauses whose heads share the same predicate symbol sum to one, the SLP is called normalised, otherwise it is called unnormalised. If all the clauses that define a SLP have an associated parameter, the SLP is called pure, impure otherwise. Muggleton in 1996 (Muggleton 1996) introduced SLP as pure and normalised.

Example 0.1. We show an example of pure and normalised SLP defined over three predicates $p(X)$, $q(X)$ and $r(X)$ and two constants $a$ and $b$:

\[
\begin{align*}
0.4: & \; p(X) \leftarrow r(X). \\
0.6: & \; s(X) \leftarrow q(X). \\
0.2: & \; a, \; 0.8: \; b. \\
0.3: & \; p(a), \; 0.7: \; p(b).
\end{align*}
\]

A pure normalised SLP $S$, defined by a set of stochastic clauses $C_i$, induces three probability distributions. Given a goal $G$ and a parameter vector $\lambda$ defined as $\lambda_i = \log l_i$ where $l_i$ is the parameter of the clause $C_i$, we have the following three probability distributions $\psi(\lambda, S, G)$, $f(\lambda, S, G)$ and $p(\lambda, S, G)$. $\psi(\lambda, S, G)$ is probability distribution over derivations of $G$ using $S$. A derivation could be an infinite derivation, a finite derivation ending in $\square$ (a refutation of the goal $G$) or a finite derivation ending with $fail$. The probability of a derivation $x$ is defined as:

\[
\psi(\lambda, S, G)(x) = \prod_{i=1}^{n} \nu_i(x)
\]

where $\nu_i(x)$ is the number of times that the clause $C_i$ is used in the derivation $x$. $f(\lambda, S, G)$ is the probability distribution over refutations. The probability of a refutation $x$ is defined as:

\[
f(\lambda, S, G)(x) = Z_{(\lambda, S, G)}^{-1} \psi(\lambda, S, G)(x)
\]

where

\[
Z_{(\lambda, S, G)} = \sum_{y \in R} \psi(\lambda, S, G)(y)
\]

and $R$ is the set of all refutations of $G$ given $S$. $p(\lambda, S, G)$ is the probability distribution over atoms. It is important to notice that $p(\lambda, S, G)$ defines a distribution over atoms, not over the truth values of atoms. The probability of an atom $x$ is defined as:

\[
p(\lambda, S, G)(x) = \sum_{r \in R_x} f(\lambda, S, G)(r)
\]

where $R_x$ is the set of refutations of $G$ using $S$ that give answer $x$. In what follow we consider as probability distribution the probability distribution over derivations: $\psi(\lambda, S, G)$.

PCC

Probabilistic concurrent constraint programs (PCC) are an extension of Concurrent Constraint Programs (Saraswat 1993; Olarte, Rueda, and Valencia 2013) that allow the specification of random variables (Gupta, Jagadeesan, and Saraswat 1997).

A Concurrent Constraint Program is based on four basic constructs: the tell operator that adds a constraint to the store, the parallel composition of two agents $A$ and $B$, that provides the conjunction of the constraints of $A$ and $B$, the ask operator, that checks if a constraint $a$ is satisfied in the store and in that case call an agent $A$, and the hiding operator, that introduce a new local variable for an agent $A$.

A Probabilistic Concurrent Constraint Program (PCC) is a Concurrent Constraint Program (CCP) with the addition of a new operator that corresponds to the introduction of random variables: $\mathfrak{r} \sim \rho d$ that add to the agent $A$ a new random variable $R$ with $pdler$ as probability mass function. Thus has been introduced a new type of agent: sampling agent $a \sim \rho d$.

Example 0.2. We show an example of a PCC program defining a predicate $p(X)$ over two random variables $X$ and $Y$ with domain $\{a, b\}$:

\[
p(X) \rightarrow Y \sim a/0.3+b/0.7,
Y=a \rightarrow X \sim a/0.8+b/0.2,
Y=b \rightarrow X \sim a/0.1+b/0.9.
\]

Similarly to SLPs, all the random variables are considered independent; the correlation between them is determined by the use of constraints. In the example above the correlation between $X$ and $Y$ is established by the constraints $X = a$ and $Y = b$. A PCC program can describe recursive definition of predicates like logic programs. An execution of a PCC program proceeds by accumulating constraints in a store (through execution of tell operations), and using them to answer constraints: the agent $c \rightarrow A$ is reduced to $A$ if the store entails the constraint $c$. A PCC program $P$ defines a probability distribution over the proofs (executions) $\psi(\lambda, P)$ of a given goal $G$. The probability of a proof $x$ corresponds to the product of the weights associated with each choice among random variables taken in the proof. The probability distribution over refutations (successful executions) $f(\lambda, P)$ is computed similarly to SLP, normalising over the probability of the set of refutations. Given a refutation $x$ then

\[
f(\lambda, P)(x) = Z_{P,G}^{-1} \psi(\lambda, P)(x)
\]

where $Z_{P,G}$ is the corresponding normalising factor.

Relation between SLP and PCC

Every SLP program can be rewritten as a PCC logic program over discrete probability distributions. Given the following SLP program in normal form:
we can encode it in the following PCC agent that has trivially the same posteriori probability distribution over proofs and refutations:

\[ q(x_1, \ldots, x_m) \rightarrow y \sim 1/p_1 \cdots + n/p_n, \]
\[ \{y=1\} \rightarrow \text{body}_1, \ldots, \{y=n\} \rightarrow \text{body}_n. \]

However, with a PCC agent we can define only normalised SLP. We can extend PCC to represent also un-normalised SLP performing sampling using non negative weights instead of probability parameters and using a proper normalisation constant at the end of the computation.

The viceversa is not always true, for the simple fact that the two underlined semantics, CCP (Concurrent Constraint Programs semantics) and CLP (Constraint Logic Programs semantics) are different.

A PCC logic program over discrete random variables can be thought as an impure SLP. The basic idea of this encoding is explained by the following simple PCC agent:

\[ q(x) \rightarrow x \sim t_1/p_1 \cdots + t_n/p_n, \]

defining a predicate \( q(X) \) over the terms \( t_1, \ldots, t_n \), can be thought as the following SLP program:

\[ p_1: q(t_1) \ldots p_n: q(t_n). \]

**Relation to existing probabilistic frameworks**

**Bayesian networks (BNs)**

Bayesian networks (D’Ambrosio 1999; Dechter 1999; Russell and Norvig 1994) are the classical framework to represent probabilities over a world defined by a set of features. BNs induce a probability distribution over the set of complete assignments of the set of features.

A formalisation of BN in SLP has been developed by Cussens (Cussens 2001).

In what follow we describe a characterisation of BNs in terms of PCC.

**Example 0.3.** We consider a BN in Figure 1 defined over 5 binary variables \( X_1, \ldots, X_5 \) (with domains \( \{0, 1\} \)) with the following dependencings \( X_1 \rightarrow X_2, X_2 \rightarrow X_1, X_3 \rightarrow X_4 \) and \( X_3 \rightarrow X_5 \), and the following probability distributions:

\[ X_1 : \text{pd}_1; X_2 : (X_1 = 0 : \text{pd}_2, X_1 = 1 : \text{pd}_3); X_3 : \text{pd}_4; X_4 : (X_2X_3 = 00 : \text{pd}_5, X_2X_3 = 01 : \text{pd}_6, X_2X_3 = 10 : \text{pd}_7, X_2X_3 = 11 : \text{pd}_8); X_5 : (X_3 = 0 : \text{pd}_9, X_3 = 1 : \text{pd}_{10}). \]

We can encode it in the following PCC agent:

\[
\begin{aligned}
\text{joint}(X_1, \ldots, X_5) \rightarrow \\
X_1 \sim \text{pd}_1, X_2 \sim \text{pd}_4, \\
(X_2=0 \rightarrow X_5 \sim \text{pd}_2), (X_2=1 \rightarrow X_5 \sim \text{pd}_3), \\
(X_2=0, X_3=0 \rightarrow X_4 \sim \text{pd}_8), (X_2=0, X_3=1 \rightarrow X_4 \sim \text{pd}_9), \\
(X_2=1, X_3=0 \rightarrow X_4 \sim \text{pd}_5), (X_2=1, X_3=1 \rightarrow X_4 \sim \text{pd}_6), \\
(X_3=0 \rightarrow X_5 \sim \text{pd}_7), (X_3=1 \rightarrow X_5 \sim \text{pd}_{10}).
\end{aligned}
\]

In general, given a Bayesian Network defined on \( n \) variables \( Var = \{X_1, \ldots, X_n\} \) we can define the following PCC agent:

\[
\text{joint}(X_1, \ldots, X_n) \rightarrow \text{R}_1, \ldots, \text{R}_n, Q_1, \ldots, Q_n,
\]

where \( n_I \) is the number of independent variables in \( Var \). \( R_i \) is the i-th independent variable in \( Var \) and \( pd_{X_i} \) corresponds to the probability distribution defined by the BN over the values in the domain of \( X_i \); \( n_D \) is the number of dependent variables in \( Var \). Each \( Q_i \) is associated to a dependent variable and corresponds to a conjunction of formulas. Given a dependent variable \( X \) there is exactly one sub-formula in its conjunction to each assignment of the \( X \)'s parent nodes. Given the i-th dependent variable \( X \) with parents \( Y_1, \ldots, Y_p \subset Var \) we have thus a sub formula in \( Q_i \) for each assignment \( a = a_1, \ldots, a_p \) of \( Y_1, \ldots, Y_p \) and \( Q_i(a) \) is of the form:

\[
(y_1=a_1, \ldots, y_p=a_p) \rightarrow x \sim \text{pd}_{X_i}^a,
\]

where \( \text{pd}_{X_i}^a \) corresponds to the probability distribution defined by the BN over the values in the domain of \( X \) for the assignment of the parents nodes \( a \).

**Theorem 0.1.** Given a BN \( B \) and its PCC translation \( \mathcal{P} \) defined over the set of variables \( Var = \{X_1, \ldots, X_n\} \), the probability distribution \( pd_{\mathcal{P}} \) over the complete assignments of the variables defined by \( B \) is equal to the probability distribution \( pd_{\mathcal{P}} \) over the complete assignments of the variables defined by \( \mathcal{P} \). Hence we have:

\[
\prod_{i \in \{1, \ldots, n\}} \mathbb{P}[X_i = x_i | Pa(X_i)] = x \mid Pa(X_i)]
\]

where \( x \mid Pa(X_i) \) is the restriction of \( x \) on \( Pa(X_i) \subset Var \) (the set of parent nodes of \( X \)).

The probability to obtain \( x = x_1, x_2, \ldots, x_n \) in \( \mathcal{P} \) will be the probability to have a successful execution of the agent:

\[
\text{joint}(x_1, \ldots, x_n) \rightarrow
\]

\[
X_1 \sim pd_1, X_3 \sim pd_4, \\
(X_2=0 \rightarrow X_5 \sim pd_2), (X_2=1 \rightarrow X_5 \sim pd_3), \\
(X_2=0, X_3=0 \rightarrow X_4 \sim pd_8), (X_2=0, X_3=1 \rightarrow X_4 \sim pd_9), \\
(X_2=1, X_3=0 \rightarrow X_4 \sim pd_5), (X_2=1, X_3=1 \rightarrow X_4 \sim pd_6), \\
(X_3=0 \rightarrow X_5 \sim pd_7), (X_3=1 \rightarrow X_5 \sim pd_{10}).
\]

We have described the only successful execution and it has the same probability as in the corresponding case in the BN \( B \).
Markov Random Fields (MRFs)

A Markov Random Field (MRF) also called Markov network (MN) is a graphical model defined over a set of $n$ variables. The variables are grouped into cliques $C_1, \ldots, C_k$ where each non-terminal state is associated with the joint probability $p_{d_1}, \ldots, p_{d_k}$ on the assignments of the variables in that clique. The joint probability of a complete assignment of the variables of the MRF corresponds to the product of the joint probability of its cliques.

Cussens in (Cussens 2001) has formalised an MRF as a SLP. We can express the same joint probability over the possible assignments of the set of variables with the following PCC agent:

$$\text{joint}(X_1, \ldots, X_n) \rightarrow C_1 \sim p_{d_1}, \ldots, C_k \sim p_{d_k},$$

$$C_i \rightarrow \{0, 0, \ldots, 0\} \rightarrow (X_i^1 = 0, \ldots, X_i^k = 0),$$

$$\ldots,$$

$$C_i \rightarrow \{1, 1, \ldots, 1\} \rightarrow (X_i^1 = 1, \ldots, X_i^k = 1).$$

for all cliques $C_i$ defined over binary variables $X_i^1, \ldots, X_i^k$, with domain $\{0, 1\}$. Similarly for the non-binary case.

**Theorem 0.2.** Given a Markov Random Field $\mathcal{M}$ and its PCC encoding $\mathcal{P}$ defined over the same set of variables $\text{Var} = \{X_1, \ldots, X_n\}$, the probability distribution $p_{d\mathcal{M}}$ over the complete assignments of the variables defined by $\mathcal{M}$ is equal to the probability distribution $p_{d\mathcal{P}}$ over the complete assignments of the variables defined by $\mathcal{P}$.

It is easy to see that the formulation of MRF is similar to the BNs one. The proof of the above theorem follows the same steps as the proof regarding BNs.

Hidden Markov Models (HMMs)

Hidden Markov Models are one of the most popular probabilistic models particularly interesting since is a stochastic extension of regular grammars (Chomsky 1957). A HMM is defined as a series of observed outputs generated by a hidden state sequence of Hidden Markov Models. A similar approach can be used to model non-binary random variables. The variables are grouped into cliques $\mathcal{C}$ for each non-terminal state $s$.

A Markov Random Field is a Markov net over the complete assignments of the set of variables defined by $\mathcal{M}$.

The joint probability of a complete assignment of the variables defined by $\mathcal{M}$ is equal to the probability distribution $p_{d\mathcal{M}}$.

$$\text{PCC agent:}$$

$$\text{hmm}(s, 0) \rightarrow 0 \rightarrow \text{O} = \emptyset.$$

where $s_{t}$ corresponds to the current state, $O$ to the current output and the clause for each terminal state $s_{t}$:

$$\text{hmm}(s_{t}, 0) \rightarrow \text{O} = \emptyset.$$

Asking $\text{hmm}(s, Y)$ we obtain the probability distribution over all the possible sequences $Y$ generated starting from an initial state $s$.

Markov Chains (MCs)

A discrete-time Markov chain is a sequence of random variables defined over the same domain: a set of states $s_1, \ldots, s_n$. Each random variable is defined by a probability distribution over the set of states, that corresponds to the probability of moving to next state and it depends only on the present state and not on the previous and future ones. The probability to move from a state $s_i$ to a state $s_j$ is given by the parameter $p_{ij}$, that could also be 0, and satisfies the property: $\sum_{j=1}^n p_{ij} = 1$.

The SLP formulation of MCs can be seen as special case of HMMs (Muggleton 1996; Cussens 2001). Given a MC defined on $n$ states $s_1, \ldots, s_n$ we represent its SLP formulation explicitly as:

$$p_{00}: \text{MC}(s_{0}, X) \Leftarrow \text{MC}(s_{0}, X).$$

$$\ldots,$$

$$p_{n0}: \text{MC}(s_{n}, X) \Leftarrow \text{MC}(s_{n}, X).$$

$$\text{MC}(s_{f}, s_{f}) .$$

where $s_f$ are the absorbing states (states with $p_{ij} = 1$ and $p_{0k} = 0 \forall k \neq f$).

The query $\text{MC}(s_{0}, X)$ will produce all the possible final states from an initial state $s_0$.

The PCC formulation is:

$$\text{MC}(s_{0}, X) \rightarrow (Y \sim s_0/p_{00} + \ldots + s_n/p_{0n}, \text{MC}(Y, X)).$$

$$\text{MC}(s_{f}, s_{f}) \rightarrow \text{MC}(s_{f}, s_{f}).$$

The queries $\text{MC}(s_{0}, s_{f})$ returns respectively the probability to start from the state $s_0$ and to arrive in the state $s_f$ and the probability distribution over the set of final states, starting from the state $s_0$.

Stochastic Context Free Grammars (SCFGs)

Stochastic Context Free Grammars (SCFGs) (Eddy and Durbin 1994; Sakakibara et al. 1994) extend context-free grammars allowing a probabilistic description of production rules, and are defined as a tuple $(W, N, R, P)$ where $N$ is the set of non-terminal symbols, $W$ is the set of terminal symbols, $R$ is the set of production rules of the form $\text{A} \rightarrow B_1 \cdots B_m$ where $A \in N$ and $B_1, \ldots, B_m \in N \cup W$ and $P$ is the set of probabilities on production rules.

We consider SCFGs in Chomsky Normal Form (from a SCFG not in normal form we can always recover an equivalent Chomsky Normal Form version (Chomsky 1956)), thus we consider only production rules of the form $A \rightarrow B_1, B_2$ or $A \rightarrow C$ where $B_1, B_2 \in N$ and $C \in W$.

A formalisation of SCFGs as a SLP program can be found in (Muggleton 1996) and in (Cussens 2001). In particular Muggleton (Muggleton 1996) explicitly introduced SLPS as generalisations stochastic context-free grammars.

Given a SCFG in Chomsky Normal Form, each non terminal element $nt_i$ is associated with a probability distribution over the set $N \times N \cup W$ such that $nt_i \rightarrow (nt, nt)j$ with probability $p_{ij}$ and $nt_i \rightarrow t_k$ with probability $q_i^k$ and $\sum_j p_{ij} + \sum_k q_i^k = 1$. We can define the following PCC agent...
inducing the same probability over the productions: for the $i$-th non-terminal $N$, we have:

$$scfg(G, [N|K], T) \rightarrow \begin{array}{c}
X \sim [nt, nt]^i/p_1^i + \cdots + [nt, nt]^m/p_m^i \\
t_i/q_1^i + \cdots + t_m/q_m^i,
\end{array}$$

$T = [N|T]^i$, append$(X, K, XX), scfg(G, XX, T^i)$. 

for every terminal $d$, we have:

$$scfg(G, [d|K], T) \rightarrow \begin{array}{c}
O \sim [d|O]^i, T \sim [d|T]^i, scfg(O, X, K, T^i),
\end{array}$$

and for an empty sequence:

$$scfg(G, [], T) \rightarrow \begin{array}{c}
O = [], T = [
\end{array}$$

where $(nt, nt)^i_j$ are all the possible pairs of non terminal elements and $t^i_j$ are all the possible terminal elements. The first argument of the agent $scfg$ corresponds to the current output string, the second argument to the list of elements to process (terminal or non-terminal), and the last argument corresponds to the parsing tree.

We can generate sentences using the following query:

$$scfg(G, r, T) \rightarrow$$

The output will be the probability distribution over all the possible strings $S$ and the corresponding parsing trees $T$, given the non-terminal element corresponding to the root $r$.

**Markov Logic networks (MLNs)**

Markov Logic networks (Richardson and Domingos 2006) are defined as a collection of first order logical formulas $F_j$, each associated with a real number $w_j$, the weight of the rule. Given a set of constants $\{c_1, \ldots, c_k\}$ (in a 1-1 relationship with intended domain of interpretation), the joint probability is the given truth values $X_1 \ldots X_n$ of the ground atoms $a_1, \ldots, a_n$ is defined as:

$$joint(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{j=1}^{m} e^{w_j n_j}$$

where $n_j$ is the number of true groundings of $F_j$ given the assignment of ground atoms $X_1, \ldots, X_n$, and $Z$ is the normalization factor.

Let the atomic formulas occurring in all groundings of all formulas be enumerated as $a_1, \ldots, a_n$. For any formula $F_i$ let $F_i^1, \ldots, F_i^m$ enumerate its $p_i$ groundings. For any ground formula $F_i$ with $[F]$ stand for the formula obtained by replacing each occurrence of $a_j$ by $X_j$, for all $j$. Then we can define: We can represent an MLN with formulas $F_1, \ldots, F_k$ as an ACC agent, defined on the $m$ variables $X_1, \ldots, X_m$ each one with domain {true, false} and corresponding to a ground atom $a_j$, and random variables $Y_{ij}$ that correspond to the truth assignments of the grounding formula $[F_i]^j$ described above:

$$\begin{align*}
\mathcal{X}(X_1, \ldots, X_n) & \rightarrow \\
Y_1 & \sim true/e^{w_1} + false/e^0, \quad v([F_1], Y_1), \\
\ldots \\
Y_k & \sim true/e^{w_k} + false/e^0, \quad v([F_k], Y_k), \\
\ldots
\end{align*}$$

The use of predicates $b(x_1), \ldots, b(x_m)$ ensure the binding of the variables $x_1, \ldots, x_m$. The last part of the ACC program (last 8 lines) defines the meta-interpretor for full FOL. This allow to define MLN in ACC and SLP not only on definite clauses but in full first order logic. We use the idea developed in a work of Becket and Posegga (Becket and Posegga 1994) in which they provide a Prolog implementaion (see also (Becket and Posegga )) for first order logic, restricting to finite proofs.

**Example 0.4.** Following the example given by Richardson et al. (Richardson and Domingos 2006), we consider the following three formulas $F_1, F_2$ and $F_3$: “Smoking causes cancer” $F_1 = \neg sm(X) \lor ca(X)$ with weight $w_1 = 1.5$; “If two people are friends, either both smoke or neither does” $F_2 = \neg fr(X, Y) \lor sm(X) \lor \neg sm(Y)$ and $F_3 = \neg fr(X, Y) \lor \neg sm(X) \lor sm(Y)$ with weights $w_2 = w_3 = 1.1$. In Figure 2 we can see the graph of the ground Markov

![Figure 2: MLN of Example 0.4](image-url)
It is important to notice that the PCC representation of a MLN has un-normalised weights for its random variables. It is easy to find a normalisation constant that allow to recover the original MLN probability distribution over complete truth assignment of the variables (corresponding to successful executions of the PCC agent) since in the PCC formulation the probability of an execution is the product of the probability over the full set of variables (in each execution we sample all the random variables). So we consider a random variable of the form:

$$Y^j \sim \text{true}/e^{w_i} + \text{false}/e^0$$

and at the end of the computation we multiply every execution probability for the following normalisation constant:

$$Z = \prod_{i=1}^{k} p_i (e^{w_i} + e^0) = \prod_{i=1}^{k} (e^{w_i} + e^0)^{p_i}$$

**Theorem 0.3.** Given a MLN $M$ defined over a set of clauses $F^j$ and its PCC translation $P$ the probability distribution $pd_M$ defined by $M$ over the complete truth assignments of the ground atoms is equal to the probability distribution $pd_P$ defined by $P$ over the complete truth assignments of the ground atoms.

**Proof.** Given a input $x_1, \ldots, x_n$ where $x_1, \ldots, x_n$ are the truth values of the ground atoms $a_1, \ldots, a_n$ generated by the MLN, we want to prove that the probability of $pd_P$ is equal to $pd_M$ on this input. Given the definition of MLN, we have that $pd_M(x_1, \ldots, x_n) = \prod_j (F^j)^{e_{j,n}}$.

Each execution of the PCC agent provide a complete instantiation of the variables $Y^j$ by definition of the predicate $j$. We have thus that the truth value of each grounding formula $[F^j]^i_k$ is defined: $\nu([F^j]^i_k, Y^j)$. The probability of such combination (and the corresponding execution) is the product of the factors associated with each choice: we have a factor $e^{w_i}$ for each $\nu([F^j]^i_k, \text{true})$ used and $e^0$ for each $\nu([F^j]^i_k, \text{false})$ used. Only one of these combinations will succeed, since the input $x_1, \ldots, x_n$ determines uniquely only one consistent instantiation of the $[F^j]^i_k$. This is implied by the fact that each variable is bounded during the execution and the correctness is ensured by the meta-interpreter. Thus there is only one sampling execution that will succeeds, since every input $x_1, \ldots, x_m$ is consistent with only one instantiation of the $[F^j]^i_k$. \[\square\]

We can define a the same encoding of MLNs in PCCs also in SLPs with minor modifications that we omit due to lack of space. It is important to notice that the SLP formalisation of MLN corresponds to an un-normalised and impure SLP program: un-normalised because the sum of the weights for clauses whose heads share the same predicate could be greater than 1, and impure since there are rules that don’t have a weight.

**Conclusion and Future work**

In this paper we have shown the expressiveness of two stochastic approaches for logic programming: Stochastic Logic Programs and Probabilistic Concurrent Constraint Programs. We have first demonstrated that we can express SLP programs with a PCC agent over discrete random variables, and vice versa for a normalised SLP. We then generalised to un-normalised SLPs. SLPs and PCCs can express main of the existing probabilistic frameworks. We have shown the direct encoding of some of them in PCC: Bayesian Networks, Markov random fields, Markov chains, Hidden Markov models and Probabilistic Context Free Grammars. We plan to extend these results to other probabilistic frameworks such as Bayesian logic programs (BLPs) (Raedt and Kersting 2003; Kersting and De Raedt 2001), Bayesian logic (Li and Russell 2013), Problog (Raedt, Kimmig, and Toivonen. 2007), etc. The main future direction is to investigate efficient inference and learning techniques such as ProPPR (Wang, Mazaitis, and Cohen 2013; 2014; Wang et al. 2015), FAM (Cussens 2001) and log-linear models for learning (Cussens 1999). Our goal is to understand the applicability of these parameter estimation techniques to PCC providing efficient inference and learning algorithms. Another interesting future direction is to show the expressiveness of models that combine probabilistic-logical reasoning to temporal reasoning: we will focus on the relation between Timed PCC (Gupta, Jagadeesan, and Saraswat 1997) and other existing time-based formalisms such as Dynamic Bayesian networks (DBNs) (Murphy 2002) and Continuous time Bayesian networks (Nodelman, Shelton, and Koller 2002). We plan also to provide a probabilistic extension of RCC (Jagadeesan, Nadathur, and Saraswat 2005) that combines the power of higher-order hereditary Harrop formulas with CCP.

**References**


\[907\]