Electronic Notes in Theoretical Computer Science 47 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume47.html pp. 1 – 10

A Statistical view of Probabilistic Finite Domains

Nicos Angelopoulos

Dept. of Computer Science
University of York
York, UK.
nicos@cs.york.ac.uk

David R. Gilbert

Dept. of Computing
City University
London, UK.
drg@soi.city.ac.uk

Abstract

We present a statistical account of a Logic Programming extension for probabilistic reasoning. Unlike other approaches to the field, Probabilistic Finite Domains (Pfd,[Ang01]) capitalise on ideas introduced in Constraint Logic Programming. In particular, we use Finite Domains to ascribe probability values, and the Store to hold probabilistic relations. In this paper we elucidate the connection between the programming constructs (viz. syntax) of Pfd and the basic elements of Probability Theory (viz. experiment, random variables, events and sample space).

1 Introduction

The area of probabilistic reasoning within some form of Logic Programming has yield a number of interesting approaches. The ones that seem to lie closest to our work can be broadly parted into two categories. On one hand there are the fully compositional approaches, which attempt to ascribe probabilities to logical objects in a manner that renders such objects full compositional probabilistic units. Such approaches include, ([CM82,Sha83,Bal87,NS92,Luk99]). On the other hand generative formalisms, such as ([Rie98,Cus00,KS00]) that sprung from areas such as Probabilistic Context Free Grammars and Machine Learning, are best in representing distributions over objects/terms generated by the underlying program (particularly so for terms from a top-level predicate).

What is common to both the categories, is that probability distributions are defined over objects that do not readily map to the underlying statistical concepts. Namely, the concepts of experiment, sample space, random variable and event. It seems intuitive that reducing programming constructs, to these basic concepts, leads to wider applicability (particularly interpretation-free probabilistic formalisms) and improved understanding over computed results.

In this paper, we will try to reduce Pfd syntax to the building blocks of Probability Theory. The two main features of Pfd that assists the most in this task, are (1) the separation of a (probabilistic) variable's finite domain from its probability function, and (2) an intuitive mapping from predicates to events. We should note that our interest in computing with probabilities, is oriented towards practical programming rather than towards formal analysis.

2 Elements of Probability Theory

An *experiment* is a central abstraction in Probability Theory. It refers to any process that generates a number of observable outcomes. A random variable maps each outcome to a unique real number, and the set of these numbers is the variable's *sample space* (Ω). An event is a subset, of the sample space. Let A, and A_i $(i = 1, \dots)$ denote events.

Definition 2.1 Probability P is a function over sets (events), such that it observes the following properties :

(i) $P(A) \geq 0$

(ii) $P(\Omega) = 1$

(iii) for any k in the positive integers, and events

$A_1, A_2, \dots, A_k$ such that, $A_i \cap A_j = \emptyset$ with $i \neq j$

then $P(A_1 \cup A_2 \cup \ldots \cup A_k) = P(A_1) + P(A_2) + \ldots + P(A_k)$

We should note that finite additivity, as expressed in (iii), is adequate for our purposes.

The *conditional probability* of an event (A_1) given the occurrence of some other event (A_2) is the probability that A_1 will occur when it is known that A_2 has already happened. This is a fundamental property of probability and can be thought of as narrowing Ω to one of its subsets (A_2 in this case).

Definition 2.2 The *conditional probability* of event A_1 given that A_2 has occur, is

$$P(A_1 \mid A_2) = \frac{P(A_1 \cap A_2)}{P(A_2)} \qquad \text{(for } P(A_2) > 0\text{)}$$

3 Integrated Domains

Probabilistic variables are the basic computational units of Pfd. A probabilistic variable has two parts; (a) a *finite domain*, which at each stage holds the collection of possible values that can be assigned to the variable, and (b) a *probability assigning function* (or simply function) which is used to assign probabilities to the elements of the domain.

One of the novel aspects of Pfd is that the two constituents of a probabilistic variable are kept in isolation from each other. There are two benefits in such an approach. Firstly, propagation techniques from finite domains research ([MR93]) are retained, since a domain's representation is not altered Secondly, probabilistic functions capture the probabilistic behaviour in a manner which is, to a large extent, independent of the particular domain values. The application of the probabilistic function over the finite domain, of the same variable, should be thought as happening at a particular time, and reflecting the state of affairs at that exact point of the particular computation.

It is natural to question the meaning of these assigned probabilities. For example what is the meaning of the statement "Variable X takes value x_1 with probability 2/3 while it takes value x_2 with 1/3." To this, Pfd 's answer is that it means anything the programmer wants to model. Thus, we take a measure-theoretic viewpoint, in the respect that we do not impose a particular interpretation on the probabilities. Moreover, Pfd can be used to compute with problems that stem from either of the two primary interpretations of Probability Theory: the subjectivist and the frequency based interpretations.

Definition 3.1 Probabilistic variables are introduced with the $\in_{\mathbf{p}}$ operator:

$$Variable \ \in_{\mathbf{p}} \ PFunction(Domain\{, Arguments\}*)$$

where, $Variable$ is an unbound logical variable, $PFunction$ is the probability ascribing function for $Variable$, $Domain$ is its finite domain, and $Arguments$ is a possibly empty sequence of arguments that are known to $PFunction$. Note that this definition allows for seamless incorporation of user defined functions.

Each function should satisfy the axiomatic definition of probability functions (Definition 2.1). Probabilistic variables constitute the first tier of mapping back to statistical concepts; in effect, they parcel a random variable and a probability density function over the finite number of outcomes. In particular, the experiment associated with a probabilistic variable is logical instantiation, and the random variable is a map from finite domain elements to successive natural numbers (starting from 1). The corresponding sample space is the set of first n numbers.

Two examples of how Definition 3.1 can be applied in practice are:

$$Die \ \in_{\mathbf{p}} \ uniform([i, ii, iii, iv, v, vi]) \tag{1}$$

PFunction	Domain	Arguments	Probabilities
Methods Description			
$uniform$	$[El_1, \ldots, El_n]$	$[]$	$[1/n, \cdots, 1/n]$
$finite\_geometric$	$[El_1, El_2]$	$[Fr]$	$[Fr/Fr+1, 1/Fr+1]$
$finite\_geometric$	$[El_1, \ldots, El_n]$	$[Fr]$	$[{}^{n}_{j=1} Fr^{n-j} / \sum_{i=0}^{n-1} 2^i]$
Examples			
$uniform$	$[i, ii, iii, iv, v, vi]$	$[]$	$[1/6, 1/6, 1/6, 1/6, 1/6, 1/6]$
$uniform$	$[i, ii, iv, v, vi]$	$[]$	$[1/5, 1/5, 1/5, 1/5, 1/5]$
$finite\_geometric$	$[head, tail]$	$[3]$	$[3/4, 1/4]$
$finite\_geometric$	$[low, med, high]$	$[2]$	$[4/7, 2/7, 1/7]$
$finite\_geometric$	$[low, high]$	$[2]$	$[2/3, 1/3]$

Table 1
Example Definitions of Probabilistic Variables

$$Coin \in_{\mathbf{p}} finite\_geometric([head, tail], 2) \tag{2}$$

The first example (1) uses function $uniform$, which takes no other function specific arguments. The behaviour it describes is that the remaining elements of its domains (a subset of $\{i, ii, iii, iv, v, vi\}$ in this case) are, in the absence of other evidence, equiprobable. The function used in the second example ($finite\_geometric$, in Constraint 2) defines a geometric probability distribution over the domain elements. In this particular instance the growth factor of the is equal to $\frac{1}{2}$. Table 1, presents some variations on how the two example Constraints (1,2) can be used. The rightmost column, tabulating the probability values, should be read as that of the probability distribution at the time of definition.

4 Probability of Predicates

In the set-theoretic terms of Definition 2.1 a probability function over a predicate (E) can be defined as follows. Let $pvars(E)$ be the probabilistic variables of E, e be a vector collecting one element from the finite domain of each variable in $pvars(E)$, $p(e)$ the corresponding vector of probability values, $\mathcal{P}$ be the program defining E, and $\mathcal{S}$ be a particular constraint store (a set of constraints). E/e denotes predicate E with its probabilistic variables replaced by their respective elements in e. The experiment we relate to E is the random choice of vector e and consequently whether $\mathcal{S} \cup \mathcal{P} \vdash E/e$. An appropriate random variable (R_E) is,

$$R_E = \begin{cases} 0 & \text{if } \mathcal{S} \cup \mathcal{P} \not\vdash E/e \\ 1 & \text{if } \mathcal{S} \cup \mathcal{P} \vdash E/e \end{cases}$$

The probability of R_E, which depends to a particular store, can then be defined as the sum of products of the involved $p(e)$. We denote this probability as $P_S(R_E = X)$. Clearly, $P_S(\Omega) = 1$ and the other two conditions of Definition 2.1 also hold. In what follows we use $P_S(E)$ to mean $P_S(R_E = 1)$.

Definition 4.1 The probability of predicate E with respect to store $\mathcal{S}$ is

$$P_{\mathcal{S}}(E) = \sum_{\substack{\forall e \\ \mathcal{S}\cup\mathcal{P}\vdash E/e}} P_{\mathcal{S}}(e) = \sum_{\substack{\forall e \\ \mathcal{S}\cup\mathcal{P}\vdash E/e}} \prod_i p(e_i)$$

In Pfd the probability of a predicate, in relation to the current store, can be found, by :

$$Probability \ is \ \mathbf{p}(Predicate) \tag{3}$$

where *Probability* is an unbound (logical) variable and *Predicate* is the predicate for which $P_{\mathcal{S}}$ is sought.

5 The Conditional Constraint

The conditional constraint is a formulation of the conditional probability of events (Defn. 2.2) that seeks to strike a compromise between the needs of representation and those of computation.

Definition 5.1 Conditional constraints are introduced by:

$$Dependent \ \mathbf{I}_\pi \ Qualifier \tag{4}$$

where $Qualifier$ (Q) and $Dependent$ (D) are predicates, and π is the probability of the constraint ($0 \leq \pi \leq 1$). When π is omitted, it is assumed to be equal to one. Logical variables in Q and D are existentially quantified over the entire constraint.

From an operational viewpoint the distributions of probabilistic variables in D are possibly altered as a result of the addition of Constraint 4 to the store. This is a local means of affecting distributions as opposed to the global way in which a probabilistic function affects its variable's distribution.

Let, $C \equiv D \ \mathbf{I}_\pi \ Q$ and $\mathcal{S}_C \equiv \mathcal{S} \cup \{C\}$ then, in probabilistic terms, Constraint 4 denotes two facts, (a) that $P_{\mathcal{S}_C}(D \mid Q) = \pi P_{\mathcal{S}}(D)$ with $P_{\mathcal{S}_C}(\neg D \mid Q) = (1-\pi) P_{\mathcal{S}}(D)$ and (b) that $P_{\mathcal{S}_C}(D \mid \neg Q) = P_{\mathcal{S}}(D)$ with $P_{\mathcal{S}_C}(\neg D \mid \neg Q) =$

$P_{\mathcal{S}}(\neg D)$. On one hand (a) states that in the cases where predicate Q holds, the probability of D is equal to π in this locality, while (b) states that when predicate Q does not hold, the degree with which D holds is not affected.

$$\begin{aligned}
P_{\mathcal{S}_C}(D) &= P_{\mathcal{S}_C}(D \cap Q) + P_{\mathcal{S}_C}(D \cap \neg Q) && \big(\text{from Prob. Theory}\big)\\
&= P_{\mathcal{S}_C}(D \mid Q) P_{\mathcal{S}_C}(Q) + P_{\mathcal{S}_C}(D \mid \neg Q) P_{\mathcal{S}_C}(\neg Q) && \big(\text{defn. 2.2}\big)\\
&= \pi P_{\mathcal{S}}(D) P_{\mathcal{S}}(Q) + P_{\mathcal{S}}(D) P_{\mathcal{S}}(\neg Q) && \big(\text{from a and b}\big)\\
P_{\mathcal{S}_C}(\neg D) &= P_{\mathcal{S}_C}(\neg D \mid Q) P_{\mathcal{S}_C}(Q) + P_{\mathcal{S}_C}(\neg D \mid \neg Q) && \big(\text{Prob.The. \& 2.2}\big)\\
&= (1-\pi) P_{\mathcal{S}}(\neg D) P_{\mathcal{S}}(Q) + P_{\mathcal{S}}(\neg D) P_{\mathcal{S}}(\neg Q) && \big(\text{from a and b}\big)
\end{aligned}$$

In this paper, we consider predicates D and Q which contain a single probabilistic variable. This is because such predicates are easier to marginalise. By marginalization we refer to the process of finding, from the probability of a composite event that of a variable. In particular, from $P_{\mathcal{S}_C}(D)$ we can compute the posterior probability of V, where $V \in pvars(D)$. Let $\mid pvars(D) \mid = 1$, $fd(V)$ be the finite domain of V, and $\phi_V(FD, v)$ the probability of element v according to the probability function of variable V applied to FD, with $FD \subset fd(V)$ and $v \in FD$. In what follows we shorten $\mathcal{S} \cup \mathcal{P} \vdash$ to $\mathcal{P} \vdash$.

Definition 5.2 Let $f_T^V(x) = \phi_V(\{v \cdot v \in fd(V) \wedge \mathcal{P} \vdash D_i/v, \forall i \cdot D_i \in T\}, x)$ The posterior distribution of variable V in relation to constraint C can be computed as follows

$$\begin{aligned}
P_{\mathcal{S}_C}(V = v) &\stackrel{def}{=} P_{\mathcal{S}_C}(D/v) + P_{\mathcal{S}_C}(\neg D/v)\\
&= \pi P_{\mathcal{S}}(D/v) P_{\mathcal{S}}(Q) \quad + (1-\pi) P_{\mathcal{S}}(\neg D/v) P_{\mathcal{S}}(Q)\\
&\qquad\qquad\qquad\qquad\quad + (P_{\mathcal{S}}(D/v) + P_{\mathcal{S}}(\neg D/v)) P_{\mathcal{S}}(\neg Q)\\
&= \sum_{\substack{y \cdot \\ \mathcal{P} \vdash Q/y}} f^V_{\{D\}}(v) \pi p(y) + \sum_{\substack{y \cdot \\ \mathcal{P} \vdash Q/y}} f^V_{\{\neg D\}}(v)(1-\pi) p(y)\\
&\qquad\qquad\qquad\qquad\quad + \sum_{\substack{y \cdot \\ \mathcal{P} \not\vdash Q/y}} f^V_{\{\}}(v) p(y)
\end{aligned}$$

In general, we need to consider more than one conditional constraint. In Pfd this is done by considering, all possible combinations of variables that appear in the *Qualifier* predicate of a constraint in which V appears in the *Dependent* predicate. More formally, let $C_j \equiv D_j \;\mathbf{I}_{\pi_j}\; Q_j$ (with $j = 1, \ldots, N$ and $C = \cup_i\{Ci\}$) such that $C_j \in \mathcal{S}$ and $V \in pvars(D_j)$, and vector $\bar{y} = \langle y_j \cdot y_j \in fd(Y_j) \wedge Y_j \in \cdot pvars(Q_j) \rangle$.

Definition 5.3 The posterior distribution of variable V in relation to C is computed as follows

```
cc_example( Die1, Die2 ) :-
    Die1 ∈p uniform([i, ii, iii, iv, v, vi]),
    Die2 ∈p uniform([i, ii, iii, iv, v, vi]),
    lucky(Die2) | Die1≠iv.

lucky( vi ).

doubles( X, X ).
```

Fig. 1. Conditional constraint example program.

$$P_{\mathcal{S}_C}(V=v) = \sum_{\substack{\bar{y}\cdot\forall y_j \\ \mathcal{P}\vdash Q_j/y_j}} f^V_{\cup_j\{D_j\}}(v) \prod_j \pi_j p(y_j) + \sum_{\substack{\bar{y}\cdot\forall y_j \\ \mathcal{P}\vdash Q_j/y_j}} f^V_{\{\neg D_j\}}(v) \prod_j (1-\pi_j) p(y_j) + \sum_{\substack{\bar{y}\cdot\forall y_j \\ \mathcal{P}\not\vdash Q_j/y_j}} f^V_{\{\}}(v) \prod_j p(y_j)$$

As an example consider query $?-cc\_example(Die1, Die2).$ against the program in Fig. 1. The prior and posterior (relative to the query) distributions are given in Table 2. The posterior probability for $Die2$ is given in the last row. Let $C \equiv lucky(Die2) \mid Die1 \neq iv$ then for the sixth face (vi) we have

$$P_{\mathcal{S}_C}(Die2 = vi) = \sum_{X\in\{i,ii,iii,v,vi\}} (P_{\mathcal{S}}(Die1 = X)\cdot\pi\cdot\frac{1}{\mid[vi]\mid}) + P_{\mathcal{S}}(Die1 = iv)\cdot\frac{1}{\mid[i,ii,iii,iv,v,vi]\mid}$$

$$= (\frac{1}{6}\cdot 1\cdot 1 + \frac{1}{6}\cdot 1\cdot 1 + \frac{1}{6}\cdot 1\cdot 1 + \frac{1}{6}\cdot 1\cdot 1 + \frac{1}{6}\cdot 1\cdot 1) + \frac{1}{6}\cdot\frac{1}{6} = \frac{31}{36}$$

while for the rest of the faces ($Y \in \{i, ii, iii, iv, v\}$) we have

$$P_{\mathcal{S}_C}(Die2 = Y) = P_{\mathcal{S}}(Die1 = iv)\frac{1}{\mid[i,\ldots,vi]\mid} = \frac{1}{6}\cdot\frac{1}{6} = \frac{1}{36}$$

Variable	Distribution
	prior
$Die1$	$[i-1/6, ii-1/6, iii-1/6, iv-1/6, v-1/6, vi-1/6]$
$Die2$	$[i-1/6, ii-1/6, iii-1/6, iv-1/6, v-1/6, vi-1/6]$
	posterior
$Die1$	$[i-1/6, ii-1/6, iii-1/6, iv-1/6, v-1/6, vi-1/6]$
$Die2$	$[i-1/36, ii-1/36, iii-1/36, iv-1/36, v-1/36, vi-31/36]$

Table 2
The effect of $lucky(Die2) \mid Die1 \neq iv$ on the distributions of $Die1$ and $Die2$.

6 Other Pfd predicates

Firstly, to illustrate the introduced syntax so far, consider the program in Fig. 1. Querying, in Pfd and on an empty store, with

$$? - cc\_example(Die1, Die2), Prb \text{ is } p(doubles(Die1, Die2)).$$

provides the answer

$$Prb = 7/36$$

which can be broken down to $1/6 + 1/36$, derived from $Die1 = Die2 = vi$ and $Die1 = Die2 = iv$ respectively.

Pfd defines, a convenient shorthand for a form of the conditional constraint. So,

$$DependentVar \ \mathbf{|}\!\neq\ QualifierVar \tag{5}$$

is defined to be equivalent to:

$$DependentVar \neq X \ \mathbf{|}\ QualifierVar = X \tag{6}$$

where DependentVar and QualifierVar are probabilistic variables and X is a logical variable, which is existentially quantified.

Also we provide means for the labelling single and multiple variables. The primary difference from standard Finite Domain labelling predicates, is that Pfd has a much richer knowledge about the domain elements, since each has an associated probability value. The single variable labelling is defined as :

$$\textbf{label}(Variable, Select, ElemProb, CurrentProb) \tag{7}$$

The overall behaviour of *label/4* is to instantiate $Variable$ to an element of its finite domain and $ElemProb$ will be instantiated to the element's probability. This is done by using $Select$ (an identification for the selection method to be used) for choosing between the different elements and their associated probabilities. Each time a different $Element$ is returned and $CurrentProb$ holds the summation of values in Probability encountered up to the current stage. This is a backtrackable predicate that succeeds once for each domain value. In a similar vein, but for a number of variables, we have

$$\textbf{label\_list}(Variables, Select, ValPrbs, CurrPrb) \tag{8}$$

This variation is particularly suited for specialised algorithms, which work better when considering more than one variable while labelling. The only difference from the single variable labelling is that $Variables$ is a list of probabilistic variables, and thus $ValPrbs$ is also a list of the corresponding probabilities.

At any stage in the computation, all current information associated with a single variable can be represented by a list of pairs, where each pair is formed by an element and its corresponding probability. This list of pairs is the current distribution of a variable.

Using the query *current_distribution(+PVariable, −ElsPrbs)* the programmer can access this information for a particular variable (*PVariable*). *ElsPrbs* include all current elements of *PVariable*'s finite domain, paired to their corresponding probabilities. The probabilities are calculated by taking into consideration all the current conditionals that involve *PVariable* at their left hand side (*Dependent* predicate).

7 Conclusions

In this paper we have examined Pfd for its statistical foundations by means of mapping its syntax to the concepts underlying modern Probability Theory. We believe this has helped to establish an argument for the intuitiveness of Pfd as a general purpose probabilistic programming formalism.

In attempting to achieve such an intuitiveness, Pfd employs two distinct features, (a) the separation of a variable's finite domain from its provability ascription function and (b) the use of Constraint technology to achieve integration of Probabilistic Theory into Logic Programming.

References

[Ang01] Nicos Angelopoulos. *Probabilistic Finite Domains.* PhD thesis, Department of Computing, City University, London, UK, 2001.

[Bal87] J. F. Baldwin. Evidential support logic programming. *Journal of Fuzzy Sets and Systems*, 24:1–26, 1987.

[CM82] K.L. Clark and F. G. McCabe. *PROLOG: a language for implementing expert systems*, volume 10, chapter 23, pages 455–470. Ellis Horwood, 1982.

[Cus00] James Cussens. Stochastic logic programs: Sampling, inference and applications. In *Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 115–122, San Francisco, USA, 2000. Morgan Kaufmann.

[KS00] Yoshitaka Kameya and Taisuke Sato. Efficient learning with tabulation for parameterized logic programs. In Yehoshua Sagiv, editor, *1st International Conference on Computational Logic (CL2000)*, number 1861 in Lecture Notes in Computer Science, pages 269–294. Springer, July 2000.

[Luk99] Thomas Lukasiewicz. Probabilistic logic programming. In *13th biennial European Conference on Artificial Intelligence*, pages 388–392, Brighton, Uk, August 1999.

[MR93] Ugo Montanari and Francesca Rossi. Finite domain constraint solving and constraint logic programming. In Alain Colmerauer and Frédéric Benhamou, editors, *Constraint Logic Programming: Selected Research*, chapter 11, pages 201–222. The MIT Press, Cambridge, Mass., 1993.

[NS92] R. Ng and V. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101:150–201, 1992.

[Rie98] Stefan Riezler. *Probabilistic Constraint Logic Programming.* PhD thesis, Neuphilologische Fakultät, Universität Tübingen, Tubingen, Germany, 1998.

[Sha83] Ehud Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proceedings of IJCAI'83*, pages 529–532. William Kauffman, 1983.