Probabilistic space partitioning in Constraint Logic Programming

Nicos Angelopoulos

Department of Computer Science,
University of York,
Heslington, York, YO1 5DD, UK
nicos@cs.york.ac.uk
http://www.cs.york.ac.uk/~nicos

Abstract. We present a language for integrating probabilistic reasoning and logic programming. The key idea is to use constraints based techniques such as the constraints store and finite domain variables. First we show how these techniques can be used to integrate a number of probabilistic inference algorithms with logic programming. We then proceed to detail a language which effects conditioning by probabilistically partitioning the constraint store. We elucidate the kinds of reasoning effected by the introduced language by means of two well known probabilistic problems: the three prisoners and Monty Hall. In particular we show how the syntax of the language can be used to avoid the pitfalls normally associated with the two problems. An elimination algorithm for computing the probability of a query in a given store is presented.

1 Introduction

Probabilistic reasoning is an integral part of areas such as AI and decision making. As our understanding of non-probabilistic, crisp, reasoning is improving, researchers are increasingly interested in furthering support for probabilistic aspects to a variety of formalisms. On the other hand, formalisms such as graphical models and in particularly Bayesian networks (BNs), are being extended to incorporate reasoning about other aspects of the real world. For instance dynamic BNs have been proposed to cope with time dependent change. However, BNs, by nature a propositional formalism, has had limited success in being incorporated with a high level general purpose programming language.

This paper presents a high level language which a extends a successful crisp formalism, logic programming, with probabilistic constructs. Our approach builds on work originally presented in (Angelopoulos, 2001). Although there exist a number of approaches integrating logic programming (LP) and probabilistic reasoning, they have all, but one, been based on the pure, clausal form of LP. Here we propose a framework which is different in nature and scope. Instead of attempting to integrate the two parts into a single inference engine, as most approaches do, CLP(pfd(Y)), based on ideas introduced by constraint logic programming, CLP(X) (Jaffar & Lassez, 1987), keeps reasoning in the two constituents separate, but interfaced seamlessly via the constraint store. The constraint store was also used in (Costa, Page, Qazi, & Cussens, 2002) for storing probabilistic information. However, the main thrust in Costa et.al. was on learning BNs. Their means for achieving this, is by fitting probability distributions on skolem functions. Their approach is thus much more focused on integration with learning of BNs than our work, which is about integration of inference with generic probabilistic engines.

From the other approaches, the ones that seem to lie closest to our work, can be broadly put into two categories. On one hand there are the fully compositional approaches, which attach probabilities to relations in a manner that renders them full compositional probabilistic units. Such approaches include, (Baldwin, 1987; Ng & Subrahmanian, 1992; Lukasiewicz, 1999). On the other hand generative formalisms, (Riezler, 1998; Cussens, 2000; Kameya & Sato, 2000; Poole, 1993) that sprung from areas such as probabilistic context free grammars and machine learning, which are stronger in representing distributions over objects/terms generated by the underlying program, specifically over terms generated from a single top-level query.

Fully compositional approaches tend to have problems with defining the appropriate *reference class* (H. E. Kyburg, 1983) for objects to which we seek to find a probability measure. Whereas, generative approaches have difficulty with making assertions about the probability of general statements. In the terms introduced by (Pollock, 1992), the former are capable of defining *indefinite* probabilities whereas the latter are better at defining *definite* probabilities. Compositional approaches are handicapped in that no universally acceptable procedures exist for reducing indefinite to definite probabilities, which are normally what we are interested in. Generative formalisms on the other hand, cannot be readily used for modelling indefinite probabilities which is the the kind of knowledge that is more compactly representable.

Interestingly researchers from both directions have tried to bridge the gap. For instance (Lukasiewicz, 2001) attempts to enhance a compositional approach, while (Cussens, 2000) devised an alternative semantics for a generative formalism. However, there has been less effort in reducing programming constructs to basic probability theory concepts, particularly, interpretation-free concepts, which would improve understanding over computed results.

The remainder of the paper is structured as follows. Section 2 contains basic terminology definitions. Section 3 introduces the generic framework. Section 4 presents an instance of the framework based on probability ascribing functions and conditional constraints over arbitrary predicates. Section 5 presents the three prisoners puzzle and Section 6 shows a solution for the Monty Hall problem. Section 7 briefly discusses some computational issues. The concluding remarks are in Section 8.

2 Preliminaries

We briefly clarify our use of the LP terminology. We follow (Hogger, 1990) and use Prolog conventions throughout. Predicates, or atomic formulae, are of the form $r(t, A)$. In our example r is a predicate name and t with A are first-order terms (t a constant and A a logical variable). We extend first-order terms to also include probabilistic variables. We follow the convention of denoting both logical and probabilistic variables with names starting with a capital letter. A logic program, such as $\mathcal{P}$, is a set of definite

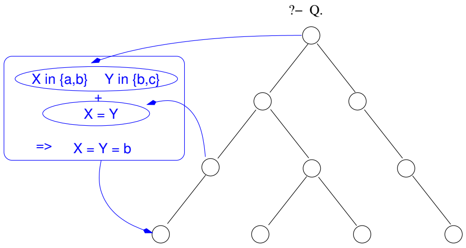

Fig. 1. Interleaving SLD-resolution and constraints inference.

clauses. A substitution θ, e.g. $\{A/a\}$, is a set of variables to term pairs. Applying θ to a formula will replace every occurrence of a variable in θ with its associated term. In our example $\theta r(t, A)$ is equivalent to $r(t, a)$.

Queries are of the form $?E_1, E_2, \ldots$ where each E_i is a predicate. Each query interrogates particular program and store. $?E$ succeeds iff $\mathcal{P} \cup \mathcal{S} \vdash E$ ($\vdash$ is the derives operator). Each successful derivation often provides a substitution θ, denoting that $\mathcal{P} \cup \mathcal{S} \vdash \theta E$. A Prolog engine will backtrack and find all θs. We take $\mathcal{P} \cup \mathcal{S} \vdash \neg E$ to be equivalent to $\mathcal{P} \cup \mathcal{S} \nvdash E$. Note that in this case there is no associated θ ($\nexists \theta \cdot \mathcal{P} \cup \mathcal{S} \vdash \theta E$).

We view the store, $\mathcal{S}$, in the usual constraint logic programming fashion as a set of constraints. In this paper $\mathcal{S}$ holds CLP(fd) and probabilistic constraints. We will often abbreviate $\mathcal{P} \cup \mathcal{S} \vdash$ to $\mathcal{S} \vdash$. Furthermore we will use $P_{\mathcal{S}}(X)$ to note the probability of X in the context of $\mathcal{P}$ and $\mathcal{S}$. Set definitions are given as $T = \{t \cdot B_1, B_2, \ldots, B_n\}$. Set T is defined as the collection of all instances t such that all B_i statements are true. We will also denote collections of unique objects as lists of the form $[B_1, B_2, \ldots, B_n]$. We extend the usual set operations to apply to lists. A list is similar to a set except that the order is not the natural sort order but a predefined arbitrary order in which the elements appear in the list.

3 Probabilistic Finite Domains for Probabilistic Reasoning

The key ideas of CLP(X) is that SLD resolution, the deduction mechanism of LP, is interleaved with constraints inference and that communication between the two components is facilitated by constraints posted by the LP engine and present in the constraints store. Fig. 1 shows an example of an interaction in diagrammatic form. The undirected part of graph corresponds to standard LP inference, while constraints posted, and information derived from the constraints store, are shown as directed arrows.

Constraints assist LP to overcome two of its computational drawbacks; the inflexibility of its computational model and the inability to use state information. Linear equation solving and finite domain propagation (CLP(fd)) are examples of commonly used CLP instances. In both, backtracking is replaced by forward pruning of the search space of special variables.

Based on these principles we can introduce variables which will carry probabilistic information. Instead of pruning the space, we are interested in evaluating the probabilities assigned to composite parts of the space. The pfd(Y) framework, short for CLP(pfd(Y)), allows graphical models to be built dynamically in the constraints store by logic programs. Probabilistic inference can then be seen as constraints inference with its results accessible to the logic engine, thus being able to influence the computation. The basis of pfd(Y) is quite generic and a number of different probabilistic inference engines can be used. In the remaining of this section we will examine the generic features of the framework while subsequent sections detail a particular instance. Another instance of the framework, pfd(bn) allowing integration of BN inference, was outlined in (Angelopoulos, 2003).

3.1 Probabilistic Variables

In CLP(fd), a finite domain variable V is introduced to the store with an associated set of values, $fd(X)$. These are all the allowed values for the variable. Constraints posted to the store may subsequently reduce this set. The basic ideas behind constraints between finite domain variables were introduced in (Montanari, 1974). In arbitrary store $\mathcal{S}$ the consistency algorithm computes $fd_{\mathcal{S}}(V) \subseteq fd(V)$ the set of allowed values given $\mathcal{S}$. Elaborating on these ideas, a probabilistic finite domain variable V, or probabilistic variable for short, in pfd(Y), defines a probability distribution over its finite domain. When introduced to the store not only is its domain declared, but the probabilistic behaviour of the variable is also described. The constraint store, in addition to CLP(fd) constraints, may also hold probabilistic information about such variables in the form of probabilistic constraints.

Definition 1. *For any variable, V present in a pfd(Y) store $\mathcal{S}$, probabilistic inference must be able to compute $\psi_{\mathcal{S}}(V) = \{(v_1, \pi_1), (v_2, \pi_2), \ldots, (v_n, \pi_n)\}$. Each v_i is an element of $fd_{\mathcal{S}}(V)$ and set $\cup_{i=1}^{n}\{\pi_i\}$ should define a probability distributions, i.e. $0 \leq \pi_i \leq 1$ and $\sum_i \pi_i = 1$. We let $P_{\mathcal{S}}(V = v_i)$ be the probability attached to element v_i under store $\mathcal{S}$.*

Note that by the above definition $P_{\mathcal{S}}(V = v_i) = \pi_i$. Also that pfd(Y) constraints are orthogonal to CLP(fd) constraints, as long as the probabilistic inference engine can recover an appropriate distribution (i.e. set of π_is). In what follows we will assume $fd_{\mathcal{S}}(V)$ to be a list of distinct objects rather than a set.

3.2 Predicates as Events

In the theory of probability, an event is a subset of the space of all possible outputs of an experiment. By treating the assignment of all possible values to probabilistic variables as our space of possible outcomes, we can view predicates containing such variables as events in this space. The main intuition is that the probability assigned to an event is proportional to the space covered by combinations of its probabilistic variables which lead to successful derivations.

For example, consider predicate $lucky(X, Y)$ in the context of program $\mathcal{P}_1$

```
       lucky(iv, hd).
P1:    lucky(v, hd).
       lucky(vi, hd).
```

The space of all possible experiments corresponds to the Cartesian product of all possible values for X and Y. The event which corresponds to $lucky(X, Y)$ is the subset of the Cartesian product that makes the predicate true. In this example $[(iv, hd), (v, hd), (vi, hd)]$. For uniformity we use square brackets throughout since we view these sets as lists of unique elements.

Definition 2. *Let $pvars(E)$ be the vector of probabilistic variables in predicate E, $\mathcal{P}$ be the program defining E and $\mathcal{S}$ a constraint store. Let E_i index the variables in $pvars(E)$, e be a vector collecting one element from the finite domain of each variable in $pvars(E)$ and $P_{\mathcal{S}}(E_i = e_i)$ the probability attached to value e_i of variable E_i as defined above. E/e denotes predicate E with its probabilistic variables replaced by their respective elements in e. The probability of predicate E with respect to store $\mathcal{S}$ and program $\mathcal{P}$ is given by:*

$$
\begin{aligned}
P_{\mathcal{S}}(E) &= P(E \mid \mathcal{P} \cup \mathcal{S}) \\
&= \sum_{\substack{\forall e \cdot \\ \mathcal{P} \cup \mathcal{S} \vdash E/e}} P_{\mathcal{S}}(E/e) \\
&= \sum_{\substack{\forall e \cdot \\ \mathcal{P} \cup \mathcal{S} \vdash E/e}} \prod_i P_{\mathcal{S}}(E_i = e_i)
\end{aligned}
$$

For instance, reconsider $\mathcal{P}_1$ in conjunction with the store $\mathcal{S}_1$ holding variables D and C, with $\psi_{\mathcal{S}_1}(D) = [(i, 1/6), (ii, 1/6), (iii, 1/6), (iv, 1/6), (v, 1/6), (vi, 1/6)]$ and $\psi_{\mathcal{S}_1}(C) = [(hd, 1/2), (tl, 1/2)]$. The probability of a lucky combination given by the probability assigned to event $lucky(D, C)$ is $P_{\mathcal{S}_1}(lucky(D, C)) = 1/4$.

In terms of programming constructs, in pfd(Y) the probability of a predicate E can be assigned to a logical variable, L with the following syntax: L is $p(E)$. For convenience L is $p(E \mid G)$ with $p(E \mid G) = p(E \wedge G)/p(G)$, is also provided.

4 Partitioning the Store

In this section we present an instance of the pfd(Y) framework. In pfd(c) probabilistic variables are coupled with probability functions and constraints between variables mark partitions to the store. The former provides means for better integration with CLP(fd) and the latter enables arbitrary events to act as conditionals on the store.

4.1 Probability Ascribing Functions

When first introduced to the store, a variable in pfd(c) is paired with a probability ascribing function. The basic intuition is that the function will provide the unconstrained distribution at any stage of the computation. This means that it should be well defined over all possible sublists of the associated finite domain.

Definition 3. *Let Fd be a list of distinct objects representing a finite domain, ϕ_V be a probability function defined over all sublists of Fd and $Args$ a list of ground terms. Probabilistic variable V is declared with $V \sim \phi_V(Fd, Args)$.*

For a sublist T of Fd ($T \subseteq Fd$) the result of $\phi_V(T, Args)$ is a list of pairs, each pair coupling an element of T to its associated probability. Each element of T should be given a probability value and the sum of all given values should be equal to one. $Args$ parameterise aspects of ϕ_V and if it is the empty list we shorten $\phi_V(Fd, [])$ to $\phi_V(Fd)$. We will use this shorter form when ϕ_V is applied, since $Args$ only play a role in variable declarations. We will also drop the subscript from ϕ_V when the context clearly identifies a particular V. In relation to the ψ notation from Definition 1, the following holds for the empty store, $\emptyset$: $\phi_V(Fd) = \psi_\emptyset(V)$. We let $\phi_V^v(Fd)$ be the probability attached to domain element v ($v \in Fd$) when applying ϕ_V to Fd.

For example, $Heat \sim \texttt{finite\_geometric}([l, m, h], [2])$ declares a finite geometric distribution for variable $Heat$. In this case the deterioration factor is 2. The distribution in the absence of other information is $\psi_\emptyset(Heat) = [(l, 4/7)(m, 2/7), (h, 1/7)]$. A distinct feature of pfd(c) is that the two constituents of a probabilistic variable are kept separate. As a result, the variable is still capable of participating in finite domain constraints. pfd(c) is therefore orthogonal to CLP(fd) while at the same time share information. A second consequence of the separation is that probabilistic functions capture statistical behaviour of variables, in a manner which is to a large extent, independent of specific domain values. Adding CLP(fd) constraint $Heat \neq m$ to the empty store, changes the distribution of $Heat$ to $[(l, 2/3), (h, 1/3)]$. This is due to the removal of m from the list of possible values for the variable. The distribution is readjusted over the remaining elements.

4.2 Conditional Constraints

In pfd(c) the probabilistic information added to the store is the conditional constraint. Its main intuition is that of defining probability subspaces in which we know that certain events hold.

Definition 4. *Let $1 \leq i \leq m$, and $\cup_i\{D_i\}$ be a set of predicates sharing a single probabilistic variable V. Let Q be a predicate not containing V, and set $\cup_i\{\pi_i\}$ define a distribution, i.e. $0 \leq \pi_i \leq 1, \sum_i \pi_i = 1$. Conditional constraint C is declared with:*

$$D_1 : \pi_1 \oplus \ldots \oplus D_m : \pi_m \;\mathbf{|}\; Q$$

We say V is the conditioned variable in C. V's distribution is altered as a result of C being added to the store. To see how the distribution changes, let q be the vector collecting one element from the finite domain of each variable in $pvars(Q)$. The conditional states that each q for which $\exists\theta$ such that $\mathcal{P} \cup \mathcal{S} \vdash \theta Q/q$ divides the current store to a number of partitions. Each partition has an associated probability, π_i, and in that subspace the values of V are constrained to be the sublist of its current finite domain that satisfy θD_i ($\mathcal{P} \cup \mathcal{S} \vdash \theta D_i$). No evidence is provided when $\nexists\theta$. Each subspace is thus weighed by $\prod_{Q_i} P_\mathcal{S}(Q_i = q_i)$ for $Q_i \in pvars(Q)$. The distribution of V according to C, $P_{\{C\}}(V = v)$, can then be computed by considering all possible subspaces and by applying ϕ_V within each subspace. The precise form is a special case of the formula given in Eq. 1.

For example, consider $A \sim \texttt{uniform}([y, n])$ and $B \sim \texttt{uniform}([y, n])$ having $\psi_\emptyset(A) = \psi_\emptyset(B) = [(y, 1/2), (n, 1/2)]$. That is, the $uniform$ function gives equal probability to all elements of a finite domain. Adding constraint C_1, $B = y : .8 \oplus B = n : .2 \mid A = y$ to the empty store, changes the probability of B to $\psi_{\{C_1\}}(B) = [(y, .65), (n, .35)]$. For $B = y$ this is derived as follows

$$\begin{aligned} P_{\{C_1\}}(B = y) = \quad & P_{\emptyset}(A = y).8\phi_B^y([y]) \\ + \quad & P_{\emptyset}(A = n)\phi_B^y([y, n]) \\ = {} & .5 \cdot .8 \cdot 1 + .5 \cdot .5 \\ = {} & .65 \end{aligned}$$

$P_{\{C_1\}}(B = n)$ is computed in a similar way by replacing .2 for .8 and $[n]$ for $[y]$. In an arbitrary store, a variable may be conditioned by more than one constraint. In this case, all different groups of constraints in which qualifying Qs are satisfied and within each such group all combinations of D_i need to be considered. It is also the case, that in an arbitrary store cyclic dependencies of variables are not allowed, i.e. a variable V conditioned by constraint with qualifying predicate Q, should not appear in a qualifying predicate of another constraint that conditions a variable in $pvars(Q)$.

Intuitively, the conditional states that when we know Q to be the case then we can divide the store, i.e. the space of all possible continuations, to probabilistic partitions. As we will see later, Q might be always true. What is interesting in this case, is that each satisfying instantiation θ can be used to partition the space accordingly.

4.3 Computing the probability of events

For query E and store $\mathcal{S}$ we use an elimination algorithm for computing the query's probability $P_{\mathcal{S}}(E)$. Elimination algorithms generalise nonserial dynamic programming. Their use as a unifying framework for probabilistic reasoning in BNs was advocated in (Dechter, 1996). Given an ordering over a set of variables and a number of buckets each holding functions over subsets of the variables, elimination algorithms process and eliminate each bucket in turn. The main idea is that the effect of each bucket eliminated is summarised in functions placed in the remaining buckets. When the last bucket is considered the overall effect of the network to the asked question is computed. Dechter (1996) presents elimination algorithms for computing answers to four kinds of probabilistic questions posed against BNs.

In our algorithm, each bucket B_i, contains a variable and all the variables it depends on, while the ordering is some topological ordering of the dependency graph. The buckets are eliminated backwards. At each stage the updated distribution of the dependant variable in the bucket is calculated and it is then passed to the remaining buckets that mention this variable. The algorithm is shown in Fig. 2, its crux is the formula for computing the marginal probability of variables, which is given in Eq. 1.

Let $dep(V)$ be the set of variables which condition V, i.e. for each constraint conditioning V add $pvars(Q)$ to $dep(V)$. To construct the dependency graph for a set of variables start with adding to the empty graph the variables of this set as nodes. For each variable V added to the graph, recursive add each $X \in dep(V)$ as a node to the graph, if it is not already present, add the arc $V \to X$. The constructed graph is directed and acyclic since cyclic dependencies of variables are not allowed. It is thus guaranteed that a topological ordering O exists. Let O_i be the ith variable in O with $1 \leq i \leq n$.

Algorithm: Compute probability of event

Input: Query Q and store S.

Output: $P_{\mathcal{S}}(Q)$

Initialise:

- Construct dependency graph G for $pvars(Q)$.
- Find a topological ordering O of G.
- Place $pvars(Q)$ to B_0. Place each O_i and $dep(O_i)$ in B_i.

Iterate:

For i = n to 1

compute $P_{\mathcal{S}}(O_i)$ according to (1)

add $P_{\mathcal{S}}(O_i)$ to each remaining bucket that mentions O_i

Compute:

updated $P_{\mathcal{S}}(Q)$ based on probabilities of $pvars(Q)$ in B_0

Fig. 2. Elimination algorithm for $P_{\mathcal{S}}(Q)$

The marginal distribution of V in store S can be found given the distributions for $dep(V)$. Intuitively, each distinct combination of values for $dep(V)$ need to be considered. For each combination the set of constraints for which the qualifying constraint is satisfied form a probabilistic subspace. The weight of this subspace is equal to the product of probabilities of the elements in the combination. The subspace is further partitioned to all possible combinations of dependent partitions in the qualified constraints. A diagrammatic example is presented later in Fig. 5.

More formally, let n be the number of constraints conditioning variable V and C_i be the ith constraint, such that $D_{i,1} : \pi_{i,1} \oplus \ldots \oplus D_{i,l_i} : \pi_{i,l_i}$ ❙ Q_i $(1 \leq i \leq n$, and l_i number of Ds in $C_i)$. Let R be a non empty subset of $\{1, \ldots, n\}$ and T_R be a set of pairs $T_R = \{(r, t_r) \cdot r \in R, 1 \leq t_r \leq l_r\}$. Conjunctions of qualifying Qs are in $\hat{Q}^R \equiv \wedge_{r \in R} Q_r$, possible combinations of D_is from qualified conditionals are in $\hat{D}^{T_R} \equiv \wedge_{(r,t_r) \in T_R} D_{r,t_r}$ and $\neg\hat{Q} \equiv \neg\hat{Q}^{\{1 \ldots n\}}$ is the case where none of the conditionals is qualified. We assume that each element in D_{r,t_r} is independent from the others. Let $\Psi_{\mathcal{S}}(v, \hat{D}^{T_R})$ be the probability value of element v when ϕ_V is applied to the subset of V's domain which satisfies $\hat{D}^{T_R}$. $\hat{q}$ is a vector of values for $pvars(Q)$ and $\hat{q}^R$ is the sub-vector of $\hat{q}$ for conjuction of predicates $\hat{Q}^R$. θ is an instantiation of the non probabilistic variables in $\hat{Q}^R$ such that $\mathcal{P} \cup \mathcal{S} \vdash \theta\hat{Q}^R/\hat{q}^R$ and for a given $\hat{q}$, R is the maximal subset of $\{1, \ldots, n\}$ for which such θ exists.

Definition 5. *The marginal distribution of variable V conditioned by n constraints in arbitrary S is computed with:*

$$\begin{aligned}
P_{\mathcal{S}}(V=v) &= \sum_{\hat{q}\cdot\exists R} P_{\mathcal{S}}(V{=}v \wedge \hat{Q}^R/\hat{q}^R) + \sum_{\hat{q}\cdot\nexists R} P_{\mathcal{S}}(V{=}v \wedge \neg\hat{Q}/\hat{q}) \\
&= \sum_{\hat{q}\cdot\exists R} P_{\mathcal{S}}(V{=}v \,|\, \hat{Q}^R/\hat{q}^R) P_{\mathcal{S}}(\hat{Q}^R/\hat{q}^R) + \sum_{\hat{q}\cdot\nexists R} P_{\mathcal{S}}(V{=}v \,|\, \neg\hat{Q}/\hat{q}) P_{\mathcal{S}}(\neg\hat{Q}/\hat{q}) \\
&= \sum_{\hat{q}\cdot\exists R} (\sum_{T_R} (\prod_{\substack{r \in R \\ (r,t) \in T_R}} (\pi_{r,t}) \Psi_{\mathcal{S}}(v, \theta\hat{D}^{T_R})) P_{\mathcal{S}}(\theta\hat{Q}^R/\hat{q}^R)) \\
&\qquad + \sum_{\hat{q}\cdot\nexists R} P_{\emptyset}(V{=}v) P_{\mathcal{S}}(\neg\hat{Q}/\hat{q}^R))
\end{aligned} \tag{1}$$

There are two special cases of the conditional that are of particular interest. Let $D \mid_{\pi} Q$ be a shorthand for $D : \pi \oplus \neg D : (1-\pi) \mid Q$. Note that $\neg D$ is equivalent to $\mathcal{P} \cup \mathcal{S} \not\models D$. Also, let $DepV \mid\!\ddagger\, \pi\, QlfV$ be a shorthand for

$$DepV \neq X : \pi \oplus DepV = X : 1 - \pi \mid QlfV = X$$

Predicates = and $\neq$ are the standard equality and inequality predicates respectively. $DepV$ and $QlfV$ are probabilistic variables and X is a logical variable. Furthermore, in what follows, we drop π when it is equal to 1. For example, $DepV \mid\!\ddagger\, 1\, QlfV$ is shortened to $DepV \mid\!\ddagger\, QlfV$ and it is equivalent to $DepV \neq X : 1 \mid Qlf = X$.

To recapitulate, pfd(c) instantiates the proposed framework as follows. Probabilistic information at declaration time is given by ϕ_V, which describes the probabilistic behaviour of the variable after all finite domain pruning. Conditionals add probabilistic information to the store. This information partitions the space to weighed subspaces within which different events may hold. Inference uses these partitions and the application of functions to compute updated probability distributions for the conditioned variables.

5 Three Prisoners

We use the three prisoners puzzle to illustrate that even in simple settings human intuition can lead to incorrect conclusions. This problem is a well known cautionary example, e.g. see (Mosteller, 1965; Grünwald & Halpern, 2003). Grünwald and Halpern (2003) state the problem as follows:

> Of three prisoners a, b, and c, two are to be executed, but a does not know which. Thus, a thinks that the probability that i will be executed is 2/3 for $i \in \{a, b, c\}$. He says to the jailer, "Since either b or c is certainly going to be executed, you will give me no information about my own chances if you give the name of one man, either b or c, who is going to be executed." But then, no matter what the jailer says, naive conditioning leads a to believe that his chance of execution went down from 2/3 to 1/2.

The objective in Grünwald and Halpern (2003) is to contrast and compare solutions provided by conditioning on the "naive space" to those given by conditioning in the "sophisticated space". The objective of this paper is to show how the use of syntax describing the problem in pfd(c) can lead to computations that always occur in the sophisticated space. Provided this space is built correctly, mathematically correct answers are guaranteed.

In the three prisoners puzzle, let w_i be the win of i, i.e. prisoner i is not executed then the naive space is $W = \{w_a, w_b, w_c\}$. Let o_i be the case when i is observed, i.e. when the jailer names i as one of the prisoners to be executed, then the set of all possible observations is $O = \{o_b, o_c\}$. The sophisticated space is then a product of the two set reflecting all the possible interactions. That is $I = \{(w_a, o_c), (w_a, o_b), (w_b, o_c), (w_c, o_b)\}$. Note that these are no longer equiprobable due to luck of symmetry. Each of the two (w_a, o_i) has half the probability of the other two elements in the set.

tp(Obs, Prb) :-
  $W \sim \texttt{uniform}([a, b, c])$,
  $O \sim \texttt{uniform}([b, c])$,
  $O \mathrel{|\!\ddagger} W$,
  $Prb \;\; is \;\; \mathbf{p}(W = a \mid O = Obs)$.

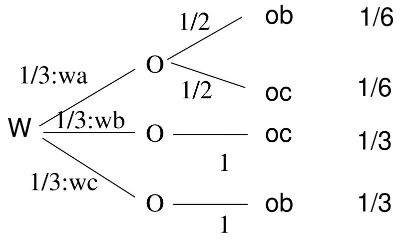

Fig. 3. Left: Clause for the three prisoners puzzle. Right: Graph for $Prb \;\; is \;\; \mathbf{p}(W = a \mid O = Obs)$

The wrong answer is computed when one conditions in the naive space. Once w_b, or equivalently w_c, is removed because of observation o_b then $W = \{w_a, w_c\}$, which leads to the belief that $P(w_a) = 1/2$. The correct answer when o_c is observed can be computed from $\{(w_a, o_c), (w_b, o_c)\}$ and the equation $P((w_a, o_c)) = \frac{1}{2}P((w_b, o_c))$. It is then apparent that $P(w_a) = P((w_a, o_c)) = \frac{1}{3}$ while $P(w_c) = P((w_b, o_c)) = \frac{2}{3}$.

To model this puzzle in pfd(c) we need two variables and one constraint. Variable W holds the name of the prisoner avoiding execution. Since we do not have any evidence to support preferences amongst the three prisoners we attach function `uniform` . The variable definition is thus: $W \sim \texttt{uniform}([a, b, c])$. Variable O holds the name of the prisoner named by the jailor. The elements of this domain are b and c, since a can never be named. Whenever the jailor can choose either of the two names, i.e. when a is to be spared, we assume that the choice is unbiased. Thus we define: $O \sim \texttt{uniform}([b, c])$. The constraint between these two variables states, that any value for W has to be distinct from the value of O. Equivalently, when the informed jailor makes his choice between b and c, he must avoid naming the wrong prisoner, i.e. the one who is not going to be executed. Thus we have $O \mathrel{|\!\ddagger} W$.

A predicate clause for the three prisoners puzzle is given in Figure 5. To ask the probability of w_a given observation b, we pose the query $tp(b, P)$. The answer will be provided by instantiating P to $1/3$. The algorithm computes $Prb \;\; is \;\; \mathbf{p}(W = a \mid O = Obs)$ by traversing the graph in Figure 5 adding the probabilities of branches for which $W = a \wedge O = b$ is true, and those for which $O = b$ holds. Finally, $p(W = a \mid O = b) = p(W = a \wedge O = b)/p(O = b)$. Here, $\frac{1}{6}/\frac{1}{2} = \frac{1}{3}$.

Note that the order of the variables in the conditional is crucial. If instead of the correct $O \mathrel{|\!\ddagger} W$ one uses $W \mathrel{|\!\ddagger} O$ the answer will be $1/2$. The correct order is in accord with the jailor knowing the winner. Thus, observations, O are conditioned on W not the other way around.

6 Monty Hall

Here we examine another problem which is often used to demonstrate that reasoning with probability is hard and often leads to wrong answers. In the Monty Hall TV show, (vos Savant, 1992; Grinstead & Snell, 1997, p.137) there are three curtains. Behind these a car and two goats are concealed. A contestant chooses an initial curtain. One of

curtains(alpha, Prb) :-
$Gift \sim$ `uniform`$([a, b, c])$,
$First \sim$ `uniform`$([a, b, c])$,
$Reveal \sim$ `uniform`$([a, b, c])$,
$Reveal \neq Gift$,
$Reveal \neq First$,
Second = First,
`Prb` is $\mathbf{p}(Second{=}Gift)$.

curtains(gamma, Switch, Prb) :-
$Gift \sim$ `uniform`$([a, b, c])$,
$First \sim$ `uniform`$([a, b, c])$,
$Reveal \sim$ `uniform`$([a, b, c])$,
$Second \sim$ `uniform`$([a, b, c])$,
$Reveal \neq Gift$,
$Reveal \neq First$,
$Second \neq Reveal$,
$Second \neq_{Switch} First$
`Prb` is $\mathbf{p}(Second{=}Gift)$.

Fig. 4. pfd(c) clauses for strategies α and γ.

the remaining curtains is then drawn revealing one of the goats. The contestant is then asked to make a final choice. Whatever is behind this final curtain is the contestant's prize. We will say that the contestant wins if the final choice is the curtain concealing the car. A more thorough analysis of the show in the context of logic and probability can be found in (Grünwald & Halpern, 2003).

What we would like to establish is whether the contestant can develop a strategy for playing the game which will increase the chances of a win. By far, most people believe that this is not possible, and that both options available to the contestant at the final choice of curtain (strategy α :stay-with-same, against strategy β switch-to-other curtain) are equiprobable. However, a closer look reveals that the latter choice is twice as likely to lead to a win. (For a justification see Grinstead and Snell (1997), for evidence of the counter-intuitiveness of the example see vos Savant (1992).) Furthermore, both choices can be seen as part of a single strategy; that of switching with probability Swt. The first, stay-with-same is equivalent to $Swt = 0$ and switch-to-other to $Swt = 1$. Lets call this unifying strategy γ . Then probability of win is $P(\gamma) = \frac{1+Swt}{3}$.

An intelligent decision making system would, ideally, be able to construct the analytical formula from a basic description of the problem. However, this is a very ambitious target. Instead, we require that the system, given a description, should perform computations that provide the same answers to those computed by the formula. In pfd(c) the program modelling the game, is used to extensionally compute the answers given by the analytical formula, from the constituents parts of the problem. The pfd(c) clauses for strategies α and γ are shown in Fig. 4. The declarative reading for the latter clause is that Prb is the probability of a win given that the player switched with probability Swt. Probabilistic variables $First$, $Gift$, $Reveal$ and $Second$ are defined as having no prior information to distinguish the three possible values in their domains. Constraint $\neq$ is used to declare the pair of variables that must be different (for example the revealed curtain cannot be the player's first choice $Reveal \neq Gift$). Variable $Second$ is defined to be different than $First$ with probability Swt and the same as $First$ with probability $1 - Swt$. $P(\gamma)$ is assigned to Prb with the call `Prb` is $\mathbf{p}(Second{=}Gift)$.

Querying this program with `curtains(gamma, 1/2, Prb)` provides the answer $Prb = 1/2$. In the meta-interpreter we have constructed, when trying to reduce/prove the goal `Prb` is $\mathbf{p}(Second{=}Gift)$ all possible concrete values to $Gift$

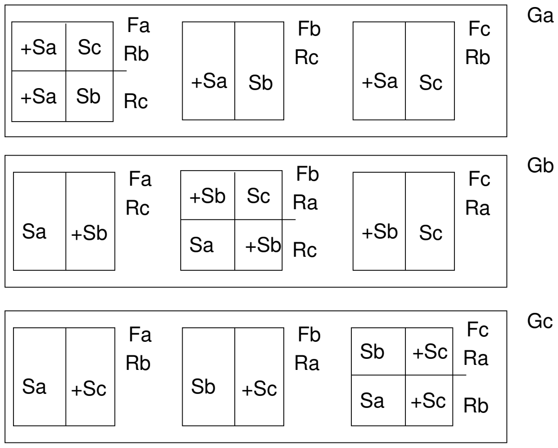

Fig. 5. Weighed subspaces for Monty Hall program. Probability of a win under strategy γ with $Swt = \frac{1}{2}$ is proportional to the area covered by squares marked with $+S_s$ labels.

, $First$ and $Reveal$ are considered. Thesse variables are chosen because they do not appear on the right hand side of any conditional constraints while at the same time, variables in $Second = Gift$, the goal for which we wish to find the probability, depend on them. The distribution of $Second$ can then be established for each set of values from the final conditional. Note that each such set has an associated probability. Finally the total distribution of $Second$ is equal to the total of the weighed sub-distributions.

A diagrammatic form of the partitioning performed by pfd(c) is in Fig. 5. V_v stands for variable V taking value v. We have abbreviated variables from Fig. 4 to their first letter. Inner squares containing S_s values are drawn to scale. The probability of a win, under strategy γ with $Swt = \frac{1}{2}$ is equal to the area covered by $+S_s$ labelled squares, where $s \in \{a, b, c\}$ and the $+$ prefixes assignments S_s which coincide with respective assignments to G. The current implementation of pfd(c) explores this space naively in order to find all combinations that satisfy $G = S$.

7 Discussion

With respect to inference, pfd(c) follows very closely work in probabilistic inference with graphical models such as BNs. We have not analysed the exact relationship here, since pfd(Y) can be instantiated to languages that provide discrete BN inference with any algorithm from the BN literature. pfd(bn) was outlined in (Angelopoulos, 2003).

With regard to computational power required, pfd(c) when all variables are conditioned on a single constraint, is comparable to inference in BNs. Specifically, to variable elimination algorithms, such as (Dechter, 1996). The main extra work pfd(c) needs to do is in deriving each event.

In the case of multiple constraints conditioning a single variable each summation step in the elimination algorithm becomes exponential to the number of elements in corresponding T_R. The extra computations reflect the gain in expressivity pfd(Y) offers on top of static propositional formalisms such as BNs.

Potential improvements include use of a limited set of predicates (particularly = and $\neq$) in the conditional constraints. Also, use of tabling or even tabulation for eliminating re-proving of known events. Finally, the use of single algorithm for constraint propagation and probabilistic inference. The similarities of certain algorithms for both these tasks have already be noted, for instance see (Dechter & Larkin, 2001).

8 Conclusions

In this paper we presented a generic framework for integrating logic programming with probabilistic reasoning. We have used constraints techniques to accomplish this. In particular, finite domains used as basic probabilistic objects, the constraint store for holding probabilistic information and the solver for probabilistic inference. Interaction between crisp and uncertain reasoning was furthered by treating predicates as statistical events. In this regard, the use of probability theory was free of a particular interpretation.

The main characteristic of pfd(Y) is its ability to transparently mix crisp and uncertain computation. As it has been argued by many researchers, e.g. (H. E. Kyburg, 1996), probabilistic reasoning is computationally more expensive than its crisp counterpart. By allowing crisp and uncertain reasoning within one framework we gain representational power. Whereas, by keeping them separate, probabilistic reasoning within the constraint solver and crisp reasoning in the logic engine, we contain the algorithmic behaviour of each part within distinct limits. Altogether, we anticipate that the ability to add well studied algorithms for probabilistic inference to logic programming will be of benefit to AI research and applications.

We have detailed an instance of the generic formalism and showed how it can be used by means of two well studied statistical puzzles. Of particular significance is the fact that the solutions to these puzzles are often different than those obtained by non-experts.

A prototype implementing pfd(c) has been built to provide initial feedback. This can be downloaded from `http://www.cs.york.ac.uk/~nicos/sware/pfds/` or tested on-line at `http://www.doc.ic.ac.uk/~nicos/cgi/pfds/pfds.cgi`. The prototype system has been used to demonstrate both the soundness of probabilistic inference from intuitive first principles and the exploitation of probabilistic information for improvements in performance within a declarative framework.

In the future, we expect to built a similar system for discrete BNs, and we would be keen to experiment with methods to incorporate finite domain propagation interaction, i.e. the ability to recover a meaningful distribution once the set of possible values for a BN variable was pruned. Finally we hope to use this framework for combining constraint propagation and probabilistic inference into a single algorithm. The two kinds of algorithms have a number of similarities and can be viewed as variable elimination algorithms, as noted by (Dechter & Larkin, 2001). Finally, we hope to refine the definition of the pfd(Y) constraint domain along the lines of existing formal definitions for constraints systems such as (Saraswat, 1992).

Acknowledgements

Thanks to James Cussens for his helpful comments, to one of the reviewers for the meticulous corrections they provided, and to another reviewer for their encouraging comments.

References

Angelopoulos, N. (2001). *Probabilistic Finite Domains*. Ph.D. thesis, Department of Computing, City University, London, UK.

Angelopoulos, N. (2003). clp(pfd(Y)): Constraints for probabilistic reasoning in logic programming. In *Ninth International Conference on Principles and Practice of Constraint Programming* Kinsale, Ireland.

Baldwin, J. F. (1987). Evidential support logic programming. *Journal of Fuzzy Sets and Systems*, *24*, 1–26.

Costa, V., Page, D., Qazi, M., & Cussens, J. (2002). CLP($\mathcal{BN}$) constraint logic programming for probabilistic knowledge. In *19th Annual Conference on Uncertainty in AI*, pp. 517–524 Acapulco, Mexico.

Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications.. In *16th Annual Conference on Uncertainty in AI*, pp. 115–122 San Francisco, USA.

Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of Uncertainty in AI (UAI-96)* Portland, USA. Extended version in Dechter99.

Dechter, R., & Larkin, D. (2001). Hybrid processing of beliefs and constraints. In *Proceedings of Uncertainty in AI (UAI-2001)*. Extended version in Dechter2001a.

Grinstead, C. M., & Snell, J. L. (1997). *Introduction to Probability* (Second Revised Edition edition). American Mathematical Society.

Grünwald, P., & Halpern, J. (2003). Updating probabilities. *Journal of AI Research*, *19*, 243–278.

H. E. Kyburg, J. (1983). The reference class. *Philosophy of Science*, *50*, 374–397.

H. E. Kyburg, J. (1996). Uncertain inferences and uncertain conclusions. In *12th Annual Conference on Uncertainty in AI*, pp. 365–372.

Hogger, C. J. (1990). *Essentials of Logic Programming*. Oxford University Press, Oxford.

Jaffar, J., & Lassez, J.-L. (1987). Constraint logic programming. In *POPL'87: Proceedings 14th ACM Symposium on Principles of Programming Languages*, pp. 111–119 Munich. ACM.

Kameya, Y., & Sato, T. (2000). Efficient learning with tabulation for parameterized logic programs. In Sagiv, Y. (Ed.), *1st International Conference on Computational Logic (CL2000)*, pp. 269–294. Springer.

Lukasiewicz, T. (1999). Probabilistic logic programming. In *13th Biennial European Conference on Artificial Intelligence*, pp. 388–392 Brighton, UK.

Lukasiewicz, T. (2001). Probabilistic logic programming under inheritance with overriding. In *17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, pp. 329–336 Seattle, Washington, USA.

Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, *7*, 95–132.

Mosteller, F. (1965). *Fifty challenging problems in probability, with solutions*. Addison-Wesley.

Ng, R., & Subrahmanian, V. (1992). Probabilistic logic programming. *Information and Computation*, *101*, 150–201.

Pollock, J. L. (1992). The theory of nomic probability. *Synthese*, *90*, 263–300.

Poole, D. (1993). Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, *64*, 81–129.

Riezler, S. (1998). *Probabilistic Constraint Logic Programming*. Ph.D. thesis, Neuphilologische Fakultät, Universität Tübingen, Tubingen, Germany.

Saraswat, V. (1992). The category of constraint systems is Cartesian-closed. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pp. 341–345 Santa Cruz, California. IEEE Computer Society Press.

vos Savant, M. (1992). *Ask Marilyn*. St. Martins, New York.