# 1 Introduction

## 1.1 Probability and Artificial Intelligence

The ability to perform reasoning with uncertainty is a prerequisite for intelligent behaviour. Consequently there has been considerable Artifical Intelligence (AI) research into representing and reasoning with uncertainty. Given that there have been several centuries of successful applications of probability to uncertain reasoning, it would seem a natural tool for uncertainty in AI. However, In 1969 McCarthy and Hayes produced an influential paper [16], which proclaimed that probabilities were "epistemoligically inadequate" and much early AI work on uncertainty accepted this argument.

More recently though, probabilistic approaches to AI uncertainty handling have been used extensively, indeed they are now dominant, as recent proceedings of the *Uncertainty in Artificial Intelligence* Conference demonstrate. The success of Bayesian networks as a computationally feasible way of combining structural and quantitative aspects of probability distributions has been instrumental in this probabilistic resurgence.

An important part of the work on probability and AI investigates the logical formalisation of probabilistic reasoning. The goal is to use a logical analysis to answer (or at least clarify) some of the following questions:

1. What are the semantics behind probabilistic reasoning?
2. How should we update probabilities in the light of new evidence (which may itself be uncertain)?
3. If we know that 90% of Swedes are Protestants, does it follow that the probability that a particular Swede of unknown relegion, say, Henrik Boström, has probability 0.9 of being Protestant?
4. Is the probability calculus alone sufficient for uncertain reasoning? If not what connections are there with other uncertainty formalisms, e.g. belief functions, fuzzy logic and nonmonotonic logic.

## 1.2 Halpern's logic of probability

Halpern's *An Analysis of First-Order Logics of Probability* [11] is an important contribution to the first question, where Halpern gives two approaches to giving semantics to (formalised) probability statements. The statement

> The probability that a randomly chosen bird flies is greater than 0.9

is formalised as

$$w_x(Flies(x)|Bird(x)) > 0.9 \tag{1}$$

The semantics of such a formula are defined in terms of a *type-1 probability structure* which is a standard first-order model with an associated probability function $\mu$ on its domain. Formula 1 is satisfied by any type-1 probability structure where the probability is greater than 0.9 that a domain element, constrained to be in the extension of $Bird$ and sampled according to $\mu$, is in the extension of $Flies$.

In contrast, the statement

> The probability that Tweety (a particular bird) flies is greater than 0.9.

is formalised as

$$w(Flies(Tweety)) > 0.9 \tag{2}$$

For such formulae, semantics are defined in terms of a probability distributions over first-order models ("possible worlds") rather than on some particular first-order model. Formula 2 is satisfied by any such distribution that gives probability mass greater than 0.9 to the set of those models in which $Flies(Tweety)$ is true.

These two semantics loosely correspond to the objective and subjective interpretations of probability found in the philosophical literature. Using this logical formalisation, Halpern and colleagues, in a series of papers, have clarified many of the issues involved in probabilistic reasoning. Most importantly, by connecting the two semantics above we can address the "Protestant Swedes" issue in a rigorous manner [3]. The clarity of such work is often in contrast to much philosophical work on probability which can be "dry, obscure, contrary to all ordinary ideas, and on top of that prolix"[1] See, for example, the Popper-Miller argument concerning the connections between deductive and probabilistic relations [25]. The Popper-Miller position is criticised in [6].

### 1.3 Probabilistic inference is deduction

In addition to such foundational work, considerable attention has been devoted to the practical issues of incoroporating probabilistic and logical inference into some overarching framework. The alleged inability of logic to represent uncertain information is often cited as a motivation for the augmentation of a logical calculus (e.g. first-order predicate calculus) with additional probabilistic capabilities.

First-order predicate calculus is concerned with valid logical inference: when can we be certain that the truth of one statement ensures the truth of another. This focus on certainty may seem at odds with probabilistic reasoning. However, standard probabilistic inference, by which we mean the inference of probabilities from probabilities, is also exclusively concerned with deductive, certain reasoning. It is not an "inductive logic"—whatever that might be. For example given that $P(b|a) = 0.3$ and $P(a) = 0.8$, we can infer with completely certainty that $P(a \wedge b) = 0.24$.

A logical formalisation of probabilistic inference can therefore be carried out in the same way as the formalisation of other branches of mathematics, such as arithmetic: relevant axioms are added to those of first-order predicate calculus. Unfortunately, as Gödel's celebrated incompleteness result shows, not even arithmetic has a *complete* axiomatisation. Naturally any axiomatisation for probability (such as supplied by Halpern [11]) can not hope to be complete. Indeed, Abadi and Halpern also show that it "takes very little to make reasoning about probability highly undecidable" [1].

Despite these rather discouraging results, a number of researchers have described systems which incorporate logic and probability in a restricted setting, and it to this work that we devote the remainder of the paper. In Section 2, we propose *Stochastic Logic Programs* as one method of integrating logical and probabilistic reasoning. A number of alternative approaches are then considered in Section 3; and in Section 4, we summarise and consider possible future work in this area.

[1]This is Kant's description of his own *Critique of Pure Reason.*

## 2 Loglinear models for first-order reasoning

Here we propose a conservative extension to the logic programming framework by defining probabilities directly on proofs and hence indirectly on goals and atomic formulae. (Most of this work has been presented previously in [7].) Our conservatism allows us to tie probabilistic and logical concepts very closely. Table 1 lists the linkages which the proposed approach establishes.

| **Logic** | **Probability** |
|---|---|
| logical variable | random variable |
| instantiation | instantiation |
| relations | joint distributions |
| queries | queries |
| ground definitions | probability tables |
| disjunctive definitions | mixture models |
| defining relations in terms of other relations | defining distributions in terms of other distributions |

Table 1: Linking logic and probability

This section is laid out as follows. Section 2.1 introduces Probabilistic Constraint Logic Programming and loglinear distributions. Section 2 is the core section where we describe Stochastic Logic Programs. Section 2.4 gives naive inference methods using a simple example, and Section 2.5 shows more complex examples.

### 2.1 Probabilistic Constraint Logic Programming

In [26], Riezler develops Abney's work [2] defining a loglinear model on the *proofs* of formulae with some constraint logic program. This requires defining features on these proofs (denoted $f_i$) and defining the model parameters (denoted $\lambda_i$).

The essentials of this approach can be given by using the logic programming framework. This is a special case of constraint logic programming where the only constraints allowed are equational constraints between terms. We will stay with the standard logic programming framework for simplicity. Consider the logic program $LP1$ given in Fig 1.

```
s(X) :- p(X,Y), q(Y).
p(a,b).  p(a,a).  p(a,c). p(d,b).
q(b). q(c).
```

Figure 1: $LP1$, a simple logic program

Fig 2 shows the SLD-tree generated by the query $\leftarrow s(X)$ (with empty goals omitted). This shows three refutations of $\leftarrow s(X)$ which amount to two proofs of $s(a)$ and one proof of $s(d)$.

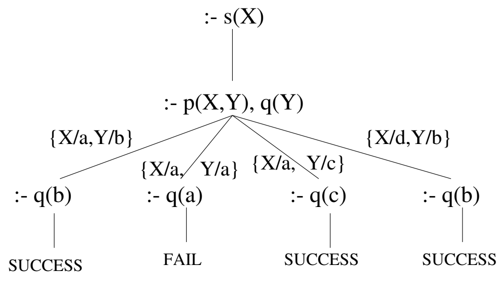


Figure 2: SLD-tree for $\leftarrow s(X)$ and $LP1$

We can define a *loglinear distribution* on refutations of $\leftarrow s(X)$. A loglinear probability distribution on a set $\Omega$ is of the following form. For $\omega \in \Omega$:

$$p(\omega) = Z^{-1} \exp\left(\sum_i \lambda_i f_i(\omega)\right) \tag{3}$$

where the $f_i$ are the features of the distribution, the $\lambda_i$ are the model parameters and $Z$ is a normalising constant.

Consider two features of refutations, $f_1$ and $f_2$, defined as follows. For any refutation $R$, $f_1(R) = n$ if the goal $\leftarrow q(b)$ appears $n$ times in $R$, and $f_2(R) = m$ if $\leftarrow q(c)$ appears $m$ times in $R$. Let $\lambda_1 = 0.2$ and $\lambda_2 = 0.4$, then the leftmost proof of $p(a)$ has probability $Z^{-1}\exp(0.2 \times 1 + 0.4 \times 0)$ and the one further to the right has probability $Z^{-1}\exp(0.2 \times 0 + 0.4 \times 1)$. The probability of the single proof of $p(d)$ is $Z^{-1}\exp(0.2\times 1+0.4\times 0)$. $Z$ is simply $\exp(0.2\times 1+0.4\times 0)+\exp(0.2\times 0+0.4\times 1)+\exp(0.2\times 1+0.4\times 0) = 2e^{0.2} + e^{0.4} \approx 3.9$, so the three probabilities are $0.31, 0.38, 0.31$ respectively. Having defined probabilities $p$ on the proofs of these atomic formulae, it is now trivial to define a probability $p'$ on the formulae themselves: $p'(Atom) = \sum_{R \text{ is a proof of } Atom} p(R)$. We have $p'(s(a)) = 0.69$ and $p'(s(b)) = 0.31$, which is a distribution on $SS(s/1)$, the success set for $s/1$.

This approach applies very naturally to natural language processing (NLP). In NLP, a proof that a sentence belongs to a language amounts to a parse of that sentence, and the loglinear model can be used to find the most likely parse of any particular sentence.

Riezler extends the improved iterative scaling algorithm of Della Pietra, Della Pietra and Lafferty [24] to induce features and parameters for a loglinear model from incomplete data. Incomplete data here consists of just atoms, rather than the proofs of those atoms. In an NLP context this means having a corpus of sentences rather than sentences annotated with their correct parses. The former is a considerably cheaper resource than the latter, so this sort of incomplete data handling is of considerable practical importance.

## 2.2 Stochastic Logic Programs

Riezler's framework allows arbitrary features of SLD-trees, and recent experiments have used features "indicating the number of argument-nodes or adjunct-nodes in the tree, and features indicating complexity, parallelism or branching-behaviour" (Stefan Riezler, personal communication).

Here we concentrate on a special case of Riezler's framework, where *the clauses used in a proof are the features defining the probability of that proof*, with clause labels denoting the parameters. Eisele [9] examined this approach from an NLP perspective. Muggleton [17] introduced *Stochastic Logic Programs*, approaching the issue from a general logic programming perspective, with a view to applications in Inductive Logic Programming.

In these cases, Stochastic Context-Free Grammars (SCFGs) were "lifted" to feature grammars (FGs) and logic programs (LPs) respectively. SCFGs are CFGs where each production is labelled, such that the labels for a particular non-terminal sum to one. The probability of a parse is then simply the product of the labels of all production rules used in that parse. Sentence probabilities are given by the sum of all parses of a sentence. The distributions so defined are special cases of loglinear models where the grammar rules define the features $f_i$ and their labels are the parameters $\lambda_i$. $Z$ is guaranteed to be one. This is because the labels for each non-terminal sum to one and because the context-freeness ensures that *we never fail* when generating a sentence from a SCFG—a production rule can always be applied to a nonterminal. Because of this a number of techniques (such as the inside-outside algorithm for parameter estimation [14]) can be applied to SCFGs, but cannot be lifted to FGs or LPs. (See Abney [2] for a demonstration of this.)

We define a stochastic logic program (SLP) as follows. An SLP is a logic program where some of the clauses are labelled with a non-negative number, and which satisfies the following constraints:

**Constraint 1** If there is a refutation of the most general goal for a predicate that uses a labelled clause, then the predicate is *distribution-defining*. It is required that the computed answer substitutions for any unit goal where the predicate is distribution-defining be ground.

**Constraint 2** The *potential* $\psi(R)$ of any refutation $R$ is the product of all the clause labels of the clauses used in $R$. If none of the clauses used in $R$ have labels, then $\psi(R)$ is undefined. The potential $\psi(G)$ of a goal is $\sum_{R\in ref(G)}\psi(R)$, where $ref(G)$ is the set of all refutations $R$ of $G$ such that $\psi(R)$ is defined. If $\psi(R)$ is undefined for all refutations $R$ of a goal $G$, then $\psi(G)$ is also undefined. We require that all goal potentials be finite.

**Constraint 3** For every distribution-defining predicate, the potential of its most general goal must be positive.

Constraint 1 can be met by requiring SLPs to be range-restricted: every variable appearing in the head of a clause must also appear in the body. The second condition is trivially met by any SLP where there is a bound on the depth of any refutation, e.g. non-recursive SLPs, and can also be met by requiring the clause labels for the clauses defining any given predicate to sum to at most one. Our definition generalises that found in [17], where Muggleton requires SLPs to be range-restricted and with labels for the same predicate summing exactly to one. Also, Muggleton does not use SLPs to define a loglinear model as we do here.

An SLP defines a distribution *for every distribution-defining predicate* in the SLP. Suppose $r/3$ were a distribution-defining predicate, then we have a loglinear distribution over refutations $R$ of the most general goal for this predicate $\leftarrow r(X_1, X_2, X_3)$, as follows:

$$p_r(R) = Z_r^{-1} \exp\left(\sum_i \log(\lambda_i) f(R,i)\right) \quad (4)$$

$$= Z_r^{-1} \prod_i \lambda_i^{f(R,i)} \quad (5)$$

where $\lambda_i$ is the label of clause $C_i$ and $f(R,i)$ is the number of times the clause $C_i$ is used in $R$. So we have a loglinear model where the labelled clauses define features and the logs of the clause labels are the model parameters.

$Z_r$ is simply the appropriate normalising constant, which can be found by simply summing the potentials of all refutations of $\leftarrow r(X_1, X_2, X_3)$. By definition, this sum is the potential of the goal $\leftarrow r(X_1, X_2, X_3)$. We have that $Z_r = \psi(\leftarrow r(X_1, X_2, X_3)) = \sum_{Atom \in SS(r/3)} \psi(\leftarrow Atom)$.

This last equation holds because Constraint 1 ensures that every refutation of $\leftarrow r(X_1, X_2, X_3)$ finds a member of $SS(r/3)$, and all elements of $SS(r/3)$ can be found this way. Constraints 2 and 3 ensure that $0 < Z_r < \infty$, so $Z_r^{-1}$ is always defined.

We get a marginal distribution over $SS(r/3)$: any ground atom $A$ has probability $p'_r(A) = \sum_{R \in ref(\leftarrow A)} p_r(R)$. Now consider the $X_i$ in $\leftarrow r(X_1, X_2, X_3)$. Each atom in $SS(r/3)$ defines a joint instantiation of the $X_i$ and therefore the distribution on atoms defines a three-dimensional joint probability distribution over $(D_1, D_2, D_3)$, where $D_i$ is the domain of $X_i$ which is both a logical and random variable. $D_i$ is just the set of values found for $X_i$ in $SS(r/3)$. In a standard logic program the $D_i$ will be finite or countably infinite.

We have used an example predicate $r/3$ to concreteness, but all of the above applies to predicates of any arity. We can use the logical structure of SLPs to define complex multi-dimensional joint distributions. Now we show how very simple SLPs can represent the joint distributions defined by Markov nets, and then give naive methods of probabilistic inference using SLPs.

## 2.3 SLPs and Markov nets

Markov nets (or undirected Bayes nets) are representations of *graphical* models, a special case of loglinear models. Fig 3 shows a version of the "Asia" Markov net used as a running example in [15], and which is also implemented as an example in the BUGS system [29].

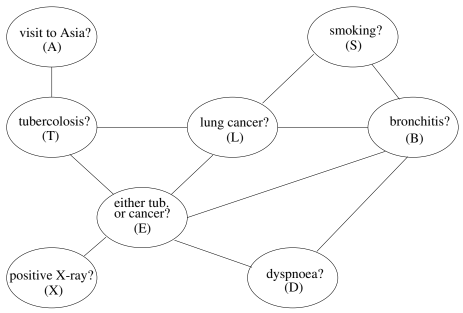


Figure 3: "Asia" Markov net

In general, let $V_C$ be the set of cliques of a Markov net $G$. A potential representation consists of *evidence potentials* $\psi_A$, defined on the cliques. Potentials are real-valued non-negative functions depending only on the states of the variables in each clique. The evidence potentials define a joint distribution on the nodes $V$ of the net as follows:

$$p(V) = Z^{-1} \prod_{A \in V_C} \psi_A \tag{6}$$

where

$$Z = \sum_V \prod_{A \in V_C} \psi_A \tag{7}$$

is a normalising constant. If $Z = 0$ then $p$ is undefined. ($Z$ will always be finite.) Consider the Markov net in Fig 3, which has six cliques. Each of the random variables in this net is binary, taking values $t$ or $f$. Table 2 gives a potential function defined on the clique $\{A, T\}$.

| Instantiation | | $\psi_{\{A,T\}}$ |
|---|---|---|
| $A = t$ | $T = t$ | 0.0005 |
| $A = t$ | $T = f$ | 0.0095 |
| $A = f$ | $T = t$ | 0.0099 |
| $A = f$ | $T = f$ | 0.9801 |

Table 2: An evidence potential on the clique $\{A, T\}$

Markov nets consist of a structure with associated parameters. Both can be represented easily using SLPs. Clique potentials are represented as tables of SLP ground facts. Figure 4 gives an SLP representation of the clique potential defined in Table 2.

```
0.0005 : c1(t,t).
0.0095 : c1(t,f).
0.0099 : c1(f,t)
0.9801 : c1(f,f).
```

Figure 4: Ground SLP representation of clique potential on $\{A, T\}$

We can then use a single unlabelled clause to represent the structure of a Markov net. The net in Fig 3 is represented by the unlabelled clause shown in Fig 5. Let us call SLPs that represent Markov nets in this fashion *Markov net SLPs*

## 2.4 Naive use of SLPs

Markov net SLPs are a very simple class of SLPs, which make them suitable for explaining basic probabilistic inference using SLPs. (In Section 2.5, we will look at more complex SLPs.) We can easily implement SLPs in Prolog, and we will use these implementations to demonstrate naive use of SLPs to process probabilistic queries. This naive use of the SLP corresponds to naive methods of using Markov nets.

```
asia(A,B,D,E,L,S,T,X) :-
   c6(E,X),
   c5(E,B,D),
   c4(L,B,X),
   c3(L,E,B),
   c2(E,L,T),
   c1(A,T).
```

Figure 5: Clausal representation of the "Asia" Markov net

Part of a Prolog implementation of the SLP which represents the "Asia" Markov net can be found in Fig 6. The `c/3` predicate looks like a probabilistic database such as those analysed in [31], but, for example, `c1(t,t,0.0005)` does *not* signify that the tuple $(t, t)$ belongs to the relation $c1/2$ with probability 0.0005. This tuple belongs to the relation $c1/2$ with certainty, we just have a distribution over tuples that belong to that relation.

```
asia(A,B,D,E,L,S,T,X,Prob) :-
        c6(E,X,P6),
        c5(E,B,D,P5),
        c4(L,B,S,P4),
        c3(L,E,B,P3),
        c2(E,L,T,P2),
        c1(A,T,P1),
        Prob is P1*P2*P3*P4*P5*P6.

c1(t,t,0.0005).
c1(t,f,0.0095).
c1(f,t,0.0099).
c1(f,f,0.9801).

%definitions for c2, c3, c4, c5, c6
```

Figure 6: "Asia" SLP implemented in Prolog

#### 2.4.1 Joint distribution

The unnormalised probability of, e.g. ($A = t$, $B = f$, $D = t$, $E = t$, $L = f$, $S = t$, $T = t$, $X = f$) can be found with the Prolog query `:- findall(P,asia(t,f,t,t,f,t,t,f,P),PS), sum_list(PS,Prob).` `findall/3` is a meta-logical Prolog built-in such that `findall(TEMPLATE,GOAL,BAG)` is true if `BAG` is a list of instances of `TEMPLATE` in all proofs of `GOAL` found by Prolog. `sum_list/2` just adds up the elements of a list. In this particular case there is only one proof of `asia(t,f,t,t,f,t,t,f,P)` (= refutation of `:- asia(t,f,t,t,f,t,t,f,P)`). The probability happens to be $1.26 \times 10^{-6}$.

#### 2.4.2 Normalisation

The normalising constant $Z(asia/8)$ can be found by putting the most general goal for `asia/8` inside the `findall/3`:

```
:- findall(P,asia(A,B,D,E,L,S,T,X,P),PS),
   sum_list(PS,Z).
```

giving the answer $Z = 0.946144$. It clearly makes sense to assert this value for future use:

```
:- assert(z(0.946144)).
```

### 2.4.3 Marginalisation

For the "Asia" case, marginal distributions are also easily found using `findall/3`. The following query returns $P(A = t, B = t)$:

```
:- findall(P, asia(t,t,D,E,L,S,T,X,P),PS),
sum_list(PS,ZATBT), z(Z), Prob is ZATBT/Z.
```

We can also use 'relationally defined' marginals, for example the following returns $P(A = t, B = t, E = T)$, the probability that $A = t$, $B = t$ and $E$ and $T$ have the same value $VV$ (whatever that might be).

```
:- findall(P,asia(t,t,D,VV,L,S,VV,X,P),PS),
sum_list(PS,ZATBT1), z(Z), Prob is ZATBT1/Z.
```

### 2.4.4 Conditional distributions

Conditional distributions are defined by the quotient of the relevant two marginal distributions. All that is required is to find the relevant normalising constant. For example,

```
:- findall(P, asia(A,B,D,VV,L,S,VV,X,P),PS),
sum_list(PS,ZES), assert(zes(ZES)).
```

finds the normalising constant needed for computations involving the conditional distribution $P(.|E = T)$. $P(A = t, B = t|E = T)$ can then be found as follows:

```
:- findall(P, asia(t,t,D,VV,L,S,VV,X,P),PS),
sum_list(PS,ZATBT), zes(ZES), Prob is ZATBT/ZES.
```

The methods we have just shown demonstrate the basic ideas needed for probabilistic inference in SLPs, but we do not propose them as serious computational methods, except for small examples. Finding $Z$ is expensive, so should be avoided unless necessary.

## 2.5 Non Markov net SLPs

Fig 7 shows an SLP defining a distribution over the non-context-free language $\{a^n b^n c^n\}$. Note that we can define distributions using structured terms, not just constants. Naively using `findall/3` to find $Z_{anbncn/1}$ with a Prolog implementation will not work since $SS(anbncn/1)$ is countably infinite, although $Z_{anbncn/1} = 1$. In general, it makes sense to implement a procedure which updates a lower bound for $Z$ as refutations are found, and produces an output when this lower bound converges.

Consider the SLP in Figure 8 which represents a simple linear Markov net. We have that $A$ is independent of $C$ given $B$ ($A \perp B|C$). For instance $P(A = foo|B = foo) = P(A = foo|B = foo, C = foo)$. This conditional independence phenomenon is central to probabilistic graphical models such

```
anbncn(A) :- build(A-B,B-C,C-[]).

0.3: build(A-A,B-B,C-C).
0.7: build([a|A]-Ap,[b|B]-Bp,[c|C]-Cp) :-
        build(A-Ap,B-Bp,C-Cp).
```

Figure 7: Stochastic non context-free grammar defined with an SLP

```
linear(A,B,C) :-
      c1(A,B),
      c2(B,C).

0.2 : c1(foo,foo).
0.1 : c1(foo,bar).
0.3 : c1(bar,foo).
0.4 : c1(bar,bar).

0.3 : c2(foo,foo).
0.2 : c2(foo,bar).
0.3 : c2(bar,foo).
0.2 : c2(bar,bar).
```

Figure 8: Linear Markov net SLP

as Markov nets. But note that $A$ is independent of $C$ given *any* value of $B$. Sometimes we may not be justified in making such a strong assumption. It may be that $A$ is only independent of $C$ given *particular values of* $B$. This *conditional conditional independence*[2] or *context-specific* independence between $A$ and $C$ crop up often in applications and has been investigated by Boutilier *et al* [4].

To represent context-sensitive independence, we need to be able to differentiate between these two sorts of values of $B$. Let us assume we have two predicates, `strong/1` and `weak/1` defined to be mutually exclusive which achieves this. We can then define the SLP in Fig 9 that defines an appropriate mixture model. A neater alternative might be to use negation to differentiate, using `strong(B)` and `\+ strong(B)`[3], but the use of negation in SLPs has yet to be properly investigated, hence our current restriction to definite clauses.

## 2.6 Representing degrees of belief

We do not use a labelled rule $\lambda : p(X,Y) \leftarrow q(X,Y), r(Y)$ to define the probability that some ground atom $p(a,b)$ is *true* (as in KBMC, see Section 3.2.1), or to provide bounds on the probability that $p(a,b)$ is true as in [28, 18]. Instead, we have a binary distribution associated with $p(X,Y)$ which defines the probability of instantiations such as $\{X/a, Y/b\}$. In order to reason about the probability of the truth of atoms, we simply augment

[2]conditional on a variable *and* conditional on values of that variable

[3]\+ is ISO Prolog notation for `not`.

```
mixlin(A,B,C) :-
      strong(B),
      c1(A,B),
      c2(B,C).

mixlin(A,B,C) :-
      weak(B),
      c3(A,B,C).

%labelled defs of c1, c2 and c3 omitted
```

Figure 9: Mixture model SLP defining context-specific independence

atoms by introducing an extra logical-random variable to represent the truth value of unaugmented atoms, and then treat this logical-random variable exactly as any other. This is in keeping with our conservative approach—if we are interested in the truth value of an atom as it varies across different "possible worlds"—then we model this variation in the standard way: with a random variable.

Consider `genotype(P,G) :- (0.5) parent(P,Q), genotype(Q,G).` an example from [13], where the "rule says that when a person's parent has a gene, the person will inherit it with probability 0.5". We would encode such "degree of belief" probability in an SLP with an boolean truth-value variable as in Fig 10. However, this labelled clause is only guaranteed to define the desired probability if it is the only clause for `genotype/3`

```
genotype(P,G,T) :- parent(P,Q),
   genotype(Q,G,1), half(T).

0.5 : half(1).
0.5 : half(0).
```

Figure 10: Representing degree of belief with an extra variable

To find the probability that $genotype(bob, big\_ears)$ is true we are required to use the SLP to compute the probabilities of $genotype(bob, big\_ears, 1)$ *and* $genotype(bob, big\_ears, 0)$. (In fact, all we need are unnormalised potentials for these, which is simpler.) This amounts to demanding arguments (=proofs) for the truth of $genotype(bob, big\_ears)$ *and* for its falsity. We then effectively balance the strength of these proofs when deciding on the probability of truth.

# 3 Research on logic and probability: a sample

Here we consider a number of approaches to integrating logic and probability. We do not pretend to have compiled a complete survey; but hope, in the spirit of statistical inference, to have collected a sufficiently representative sample to allow suitably qualified inferences.

## 3.1 Probabilities in logic programming

### 3.1.1 Clark and McCabe

Adding an extra argument to atoms to represent the degree of belief associated with such atoms, as in Section 2.6, has a long history in logic programming, although the mechanisms for propogating probabilities through Prolog rules have often been *ad hoc*.

In [5], the extra argument is a probability so that if the query $\leftarrow r(a, b, P)$ renders an answer substitution $P = 0.2$, this means that $r(a, b)$ holds with probability 0.2. Extra literals are added to the clause which compute the probability in the head atom as a function of probabilities of the body atoms: they embody a *combination rule*. Clark and McCabe do not actually give examples of such rules, but it is clear that there are links with Prolog implementations of SLPs as given in Appendix A. There the combination rule is simple multiplication (which represents the indepedence assumptions in SLPs). Clark and McCabe also propose more complex combination rules where a query matches more than one clause head. Again it is up to the user to devise these rules. With SLPs we are constrained to use mixing.

Obtaining a degree-of-belief probability (in, say, $r(a, b)$ being true) is both more direct and more flexible in [5] than with SLPs. It is more direct, because there is a single extra argument representing degree of belief. In our approach we find all refutations of $\leftarrow r(a, b, T)$ and compare the ratio of $T = 0$ to $T = 1$ answer substitutions. (We assume that the SLP has been so written so that other substitutions are not possible.) When implementing an SLP in Prolog, we have two extra arguments, one for the truth-value and one for the clause label. In [5], it is up to the "expert" to define how probabilities propogate up to the query, whereas SLPs are representations of particular sorts of distributions.

### 3.1.2 Shapiro's *logic programs with uncertainties (LPUs)*

Shapiro's approach is close to ours: the calculation of probabilities is carried out by a meta-interpreter and if all of Shapiro's *certainty factors* are 1, then operational and declarative semantics degenerate to those of standard logic programs.

LPUs involve certainty factors and certainty functions mapping multisets of certainty factors (eg the multiset of certainty factors from the body of a clause) to a single certainty factor. This is close to the Prolog implementation of SLPs, except that our certainty function (simple multiplication) is not necessarily monotonic increasing as Shapiro's have to be.

With an LPU, deducing $p(a)$ with certainty $\alpha$ is tantamount to proving that $Pr(p(a)) \geq \alpha$. Shapiro defines a minimal model for a LPU where each atom is associated with the minimal probability with which that atom is true. Consequently, the first clause below is redundant. In our approach, we do not directly represent degrees of belief and we compute exact probabilities, rather than bounds on probabilities. Also the potentials associated with different refutations of a goal add up, so that neither clause would be redundant.

```
< p(a) :- true, 0.2>
< p(a) :- true, 0.3>
```

Shapiro introduces an interpreter which essentially searches for a proof that a query is true with probability above a certain threshold. It is possible to prune this search in many cases—this would be harder in our case since we have to add up a large (possibly infinite) number of refutations of a goal to get its potential.

### 3.1.3 Probabilistic logic programming

Probabilistic Logic Programming (hereafter PLP) is an extension of Logic Programming developed by Ng and Subrahmanian in a series of papers [18, 19, 20]. In [18], the basic ideas of PLP are presented, in [19] variables ranging over probabilities are introduced and in [20] non-monotonic negation is added. As with Shapiro's LPUs, PLPs are designed for the propagation of constraints on probability values in the form of intervals, rather than the computaion of exact probabilities which we find in SLPs.

In PLP, conjunctions and disjunctions of literals are annotated with probability intervals, allowing clauses such as (8–11).

$$\mathit{eastbound}(\mathit{train}1) : [0.7, 0.9] \leftarrow \tag{8}$$

$$\mathit{bark}(X) : [0.95, 1] \leftarrow \mathit{dog}(X) : [1, 1] \tag{9}$$

$$(\mathit{bark}(X) \wedge \mathit{dog}(X)) : [0.95 * V, 0.95 * V] \leftarrow \mathit{dog}(X) : [V, V] \tag{10}$$

$$\mathit{not\_dog}(X) : [1 - V_2, 1 - V_1] \leftarrow \mathit{dog}(X) : [V_1, V_2] \tag{11}$$

The probabilities here are possible-world probabilities, and can be translated into Halpern's logic as follows:

$$w(\mathit{eastbound}(\mathit{train}1)) \in [0.7, 0.9] \leftarrow \tag{12}$$

$$\forall x : w(\mathit{bark}(x)) \in [0.95, 1] \leftarrow w(\mathit{dog}(x)) = 1 \tag{13}$$

$$\forall x, v : w(\mathit{bark}(x) \wedge \mathit{dog}(x)) = 0.95 * v \leftarrow w(\mathit{dog}(x)) = v \tag{14}$$

$$\forall x, v : \mathit{not\_dog}(x) \in [1 - v_2, 1 - v_1] \leftarrow w(\mathit{dog}(X)) \in [v_1, v_2] \tag{15}$$

Ng and Subrahmanian introduce probability intervals, since point probability values for conjuncts and disjuncts are not derivable from their constituents. In general only intervals are derivable, as shown in Equations 16 and 17.

$$P(\alpha) \in [\rho_\alpha, \delta_\alpha], P(\beta) \in [\rho_\beta, \delta_\beta] \models P(\alpha \wedge \beta) \in [\max\{0, \rho_\alpha + \rho_\beta - 1\}, \min\{\delta_\alpha, \delta_\beta\}] \tag{16}$$

$$P(\alpha) \in [\rho_\alpha, \delta_\alpha], P(\beta) \in [\rho_\beta, \delta_\beta] \models P(\alpha \vee \beta) \in [\max\{\rho_\alpha, \rho_\beta\}, \min\{1, \delta_\alpha + \delta_\beta\}] \tag{17}$$

It is worth stressing that PLP can not express domain probabilities directly. A PLP statement such as

$$\mathit{bronchitic}(X) : [0.4, 0.4] \leftarrow \mathit{smoker}(X) : [1.0, 1.0] \tag{18}$$

which translates to

$$\forall x : w(\mathit{bronchitic}(x)) = 0.4 \leftarrow w(\mathit{smoker}(x)) = 1 \tag{19}$$

does *not* state that 40% of smokers are bronchitic, but that for each particular smoker (i.e. someone who is a smoker in all worlds) there is exactly a 40% chance that that individual is bronchitic, a much stronger statement. Hence (19) would be contradicted by a single smoker **a** who is known definitely to be or not to be bronchitic, since, respectively, this would translate as $w(\mathit{bronchitic}(\mathbf{a})) = 1$ or $w(\mathit{bronchitic}(\mathbf{a})) = 0$.

### 3.1.4 Probabilistic databases

This translation of the clique functions of a Markov net to a generalised relational database is essentially the same as that of Wong *et al* [30]. Wong *et al* translate many of the graphical operations used with Markov nets to database operations: product distributions are constructed using joins, conditional distributions by projection, and marginals by database operations which mimic the standard approach in the Markov net literature. Wong *et al*'s argument is that since the operations required for effective use of Markov nets are defined on tables—for example, tables defining marginal probability distributions—one should use optimised methods developed by the database community for manipulating tables. The work on SLPs seeks to extend that of Wong *et al* by moving from a relational database setting to the logic programming setting.

## 3.2 Logic and Bayes nets

### 3.2.1 Knowledge-based model construction

*Knowledge-based model construction (KBMC)* [22] is described by Koller and Pfeffer [13]:

> *Knowledge-based model construction (KBMC)* goes a considerable way towards bridging this gap [between logic and probability] by allowing a set of *first-order probabilistic logic (FOPL)* rules (first-order rules with associated probabilistic uncertainty parameters) to be used as a basis for generating Bayesian networks tailored to particular problem instances.

Normally, one constructs a fixed Bayesian network such as the "Asia" net, so that it is 'ready' to process any query we might have. In KBMC, a Bayesian network is constructed *for a particular query*, and then that network is used to answer the query.

This query-dependence is built into logic programming with SLD-resolution—for each query we have a particular SLD-tree. Also, when using Prolog, we search for a refutation in the tree, rather than constructing the complete tree and then looking for refutations. The latter is not even an option when the tree is infinite.

In *Knowledge-based model construction (KBMC)* [21, 13, 10] first-order rules with associated probabilities are used to generate Bayesian networks for particular queries. As in SLD-resolution queries are matched to the heads of rules, but in KBMC this results in nodes representing ground facts being added to a growing (directed) Bayesian network. A *context* is defined using normal first-order rules, perhaps explicitly as a logic program [21], which specifies logical conditions for labelled rules to be used. The ground facts are seen as boolean variables (either *true* or *false*). Once the Bayesian network is built it is then used to compute the probability that the query is true.

In KBMC, as in much of the work connecting logic and probability, parameterised first-order rules $\alpha : c(X) \leftarrow a(X)$ are connected to conditional probability statements such as $p(c(b)|a(b)) = \alpha$. Also the the objective is to compute the probability that an atom is true. In the current paper, we focus on undirected representations, so that $\lambda : p(X,Y) \leftarrow q(X,Y), r(Y)$ forms part of the definition of a binary distribution associated with $p/2$ defined in terms of distributions associated with $q/2$ and $r/1$. We make *no* attempt to model causality. Despite these differences in approach there are clear similarities between KBMC query-specific Bayes net construction and the query-specific exploration of an SLD-tree by Prolog which deserve further investigation.

### 3.2.2 Ngo and Haddaway

In [10], Haddaway states that " ... a Bayesian net is essentially a propositional representation of the domain: each node represents a multi-valued propositional variable. Thus it is not possible to express general relationships among concepts without enumerating all the potential instances in advance". The existence of nodes representing random variables with countably and uncountably inifinite domains would seem to contradict this view. Such nodes are commonplace in many applications (see, for example, the documentation accompanying the BUGS system [29]). In any case, we take a different view of the connection between logical and random variables, where the nodes (random variables) in a Bayes net are identified with logical variables, ranging over a possibly infinite domain.

Let us compare Haddaway's approach, as found in [22] to that presented here. Haddaway uses a normal logic program as a *context base*. Since random variables are functions (measurable ones from some probabilities space to some other space), Haddaway takes the natural step of representing them as such. Haddaway has $(n+1)$-ary probabilistic p-predicates which map $n$-tuples to values. The value of the argument is constrained to be last argument of the relevant atom. With each p-predicate $p$ there must be a statement of the form $VAL(p) = \{v_1, v_2, \dots v_n\}$ where the $v_i$ are constants enumerating the set of possible values of the random variable represented by the p-predicate. If $A = p(t_1, \dots, t_n, t_{n+1})$ is a p-atom then $obj(A)$ is used to denote the tuple $(t_1, \dots, t_n)$ and $VAL(A)$ denotes the value $t_n$. There is thus a demarcation between individuals (such as $obj(A)$) and attributes of those individuals: $VAL(A)$.

The probabilistic base of a knowledge base is a set of probabilistic sentences of the form $(P(A_0|A_1, \dots, A_n) = \alpha) \leftarrow L_1, \dots L_n$ where the $A_i$ are p-atoms and the $L_i$ are context literals. For example:

$$P(burglary(X, yes)|nbrhd(X, good)) = .3 \leftarrow in\_CALI(X)$$

To this we add various type declarations, and rules for combining rules. Haddaway distinguishes evidence (ground p-atoms) and contextual information, which is a set of context atoms. This is in contrast to our approach where evidence is supplied by instantiating variables in the top-level query to the SLP. Contextual information is represented in the same way in both approaches. Queries take the form of $P(Q) =?$ where the last argument of $Q$ is a variable in $Q$ - the query is asking for the posterior distribution of $obj(Q)$

Query answering uses an algorithm ($Q$-procedure), which builds a "necessary portion of the Bayesian network" where each node of the net corresponds to an $obj(A)$, where $A$ is a *relevant* ground p-atom. The Bayesian net is updated using the evidence and the probabilities associated with the query are computed as per normal.

We will not give the BUILD-NET procedure for constructing the Bayesian net from a query, instead just noting that there are similarities with query-answering in SLPs. In BUILD-NET all sentences that have the input atom as consequent are collected and the associated probabilities combined according to a combination rule. This is analogous to the SLP approach of finding all refutations of a goal—except we, unlike Haddaway, only allow one form of combination. Haddaway's dynamic query-driven construction of a Bayes net is analogous to the query-driven construction of an SLD-tree.

### 3.2.3 Jaeger: first-order Bayesian nets

Jaeger [12] notes that standard Bayesian nets model a multivariate distribution over a fixed number of random variables. These apparently have to be "attributes" of a single "random event"—the Bayes net can not adequately model relations between different random events.

Jaeger's basic idea is to have Bayesian networks where the nodes are relation symbols, $r$ whose values are possible interpretations of $r$. So the random variables are set-valued. In such a set-up, normal first-order formulae setup hard-constraints on possible values for the relation symbols in our language.

In contrast, the SLP approach does not make any fundamental distinction between attributes and the individuals (Jaeger's "random events") that possess them. In this we follow standard first-order logic semantics and also standard approaches in statistics for modelling related multivariate distributions. For example, the domain of a model for the first-order theory

```
dist(london,york,333).
has(james,job,family).
social_unit(family).
father(james,rob).
```

contains elements corresponding to the constants `york,london, 333, james, job, family` and `rob` and there is no distinction between individuals and attributes. Indeed something which (intuitively) appears as an attribute in one formula can represent an individual in another, eg `family`.

## 3.3 Using ILP to construct a loglinear model

In [8], Dehaspe presents the MACCENT algorithm which constructs a loglinear model using boolean clausal constraints as features. This work is motivated by the possibility of incorporating first-order statstical modelling with other statistical methods. Dehaspe uses the "learning from interpretations" ILP setting where *each example is a Prolog database*. This approach to ILP gathers information relevant to a particular example in an example-specific Prolog program. In normal ILP, where each example is an instance of a predicate, such information would be scattered throughout the background knowledge. The learning from interpretations approach also allows example-nonspecific background knowledge. It is somewhat akin to throwing away the name of each example. A rule covers an example if the relevant query (body of the rule) succeeds in the database which represents the example.

Dehapse considers examples in different classes and has boolean features defined in terms of a class and query which is true for a (instance,class) pair iff the class matches that in the constraint and the query succeeds for the instance.

To bring out the connections between Dehaspe's approach and those others presented here, we can re-write Dehaspe's clausal constraints as labelled clauses for some dummy target predicate, re-instating the name of the example (which is the first argument in the literals in Fig 11).

Dehaspe associates (modulo our rewrite) boolean features with each labelled clause, defined on $(I, Class)$ pairs. ($B$ is background knowledge.)

```
\lambda_{1} : p(X,fish) :- \+ has_legs(X), habitat(X,water).
\lambda_{2} : p(X,reptile) :- \+ has_covering(X,hair), \+ has_legs(X).
```

Figure 11: Dehaspe's clausal constraints as labelled clauses

$$f_j(I, Class) = \begin{cases} 1 & \text{if } B, C_j \vdash p(I, Class) \\ 0 & \text{otherwise} \end{cases}$$

This defines a distribution over pairs $(I, Class)$, and hence, with suitable nomalisation, conditional distributions $Pr(Class|I)$. +

$$Pr(I, Class) = Z^{-1} \exp\left(\sum_j \lambda_j f_j(I, Class)\right)$$

We have a bijection between proofs of $p/2$ atoms and $p/2$ clauses, since each proof uses exactly one $p/2$ clause. This allows Dehaspe to treat *each proof as a feature*, where the parameter associated with each feature (=proof) is the label of the $p/2$ clause used in that proof. These features are then used to define a probability on atoms directly.

This contrasts with the SLP and PFG approach where each proof *has* features (e.g. the set of labelled clauses used in the proof), and these are used to define a *probability over proofs*. To get an (unnormalised) probability on an atom we have to sum up the probabilities of the proofs of that atom.

Dehapse's approach allows a more direct definition of a distribution over atoms, but relies on each proof passing through exactly one labelled clause. With SLPs, the probability of an atom always increases with the number of proofs of that atom. Following Dehaspe's approach this may not always be the case.

Dehapse exploits the lattice structure of clauses and applies ILP techniques to guide the search for suitable constraints. He searches for clauses with a general-to-specific beam search using the DLAB declarative bias language formalism. Dehaspe, like Riezler, keeps all the old parameters fixed when searching for the next constraint.

# 4 Conclusions and possible future directions

We have seen in Sections 2 and 3 a number of approaches to integrating logical and probabilistic reasoning. Generally, they are restricted to computing the probabilities of atoms. With the exception of SLPs, the atom-probabilities are probabilities that the atom is *true*. Although this is a natural extension of logical reasoning, it does not allow the identification of logical and random variables which the SLP does. Such an identification is likely to be of practical importance. Probabilistic inference relying on SLD-resolution, as our naive methods do, is highly inefficient. Inference methods which involve storing and transmitting probabilistic information such as Bayes net algorithms based on join trees are consderably more efficient. We conjecture that writing a meta-logical interpreter to perform such inference will be able to exploit the identification of logical and random variables using the sort of "level-crossing" tricks we see in Earley deduction interpreters [23].

In a number of cases bounds on probabilities rather than exact probabilities are computed. In PLP, considerable work has been done on propagating such constraints, and checking for consistency. The SLP approach is more traditional in that we have a fully defined probabilistic (loglinear) model from which it is possible to compute all relevant probabilities. It would be interesting to see to what extent these two approaches could be unified. One possibility would be to extend SLPs to define constraints on probabilities rather than exact values: constraint logic programming over reals (CLP(R)) is an obvious choice here.

On a more fundamental level, there are connections between the 'tree-shaped' distributions which an SLP defines via SLD-resolution and the probability trees that Shafer uses as "a language for talking about the structure of contingency" [27]. Shafer's aim is to provide a more dynamic basis for probability than the usual static set-theoretic Kolmogorovian axioms of probability theory. Rather than concern ourselves merely with the probability that an event happens, Shafer seeks to connect that probability with a probabilistic history of those events that led up to it. Shafer refers to the final event, as a *Moivrean* event—this is the event of standard probability theory. The events leading up to a Moivrean event— the changes "in some object or circumstance at a particular time" are called *Humean* events by Shafer. SLPs also have this division: there is the Humean event of choosing a particular clause head, and there is the Moivrean event of getting a particular instantiation for the variables in a top-level goal.

Although we have argued that the SLP approach allows an elegant and usable integration of logic and probability: the case has yet to be fully proven. Our preliminary exploration of the properties of SLPs leaves many questions unanswered. We conclude by listing them in order of precedence:

1. Can we adapt algorithms from the Markov net literature to provide efficient algorithms for inference in SLPs?
2. What are the analogies between transformations of Markov nets and logic program transformations?
3. How do we estimate clause labels from data—to what extent can we use the parameter estimation techniques from the Bayes net literature?
4. Can ILP techniques be used to learn the structure of Markov nets?
5. To what extent do SLPs bridge the gap between logical and probabilistic reasoning?
6. What are the necessary and sufficient conditions for clause labels to define an SLP?

## Acknowledgements

Many thanks to Sara-Jayne Farmer for weeding out various errors and omissions. Thanks to Stephen Muggleton for useful discussions on the role of normalisation in SLPs and to Stefan Riezler for clarifying his method. Thanks also Gillian, Jane and Robert Higgins for putting up with me. Finally thanks to Luc de Raedt for encouraging me to investigate first-order Bayesian nets.

## References


[1] Martín Abadi and Joseph Y. Halpern. Decidability and expressiveness for first-order logics of probability. *Information and Computation*, 112(1):1–36, 1994.

[2] Steven Abney. Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618, 1997.

[3] Fahiem Bacchus, Adam Grove, Joseph Y. Halpern, and Daphne Koller. From statistics to belief. *Artificial Intelligence*, 87:75–143, 1996.

[4] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pages 115–123, Portland, Oregon, 1996.

[5] K.L. Clark and F.G. McCabe. PROLOG: a language for implementing expert systems. In J.E. Hayes, Donald Michie, and Y-H Pao, editors, *Machine Intelligence*, volume 10, chapter 23, pages 455–470. Ellis Horwood, Chichester, 1982.

[6] James Cussens. Deduction, induction and probabilistic support. *Synthese*, 108(1):1–10, July 1996.

[7] James Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pages 126–133, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[8] Luc Dehaspe. Maximum entropy modeling with clausal constraints. In *Inductive Logic Programming: Proceedings of the 7th International Workshop (ILP-97). LNAI 1297*, pages 109–124. Springer, 1997.

[9] Andreas Eisele. Towards probabilistic extensions of constraint-based grammars. Contribution to DYANA-2 Deliverable R1.2B, DYANA-2 project, 1994.

[10] Peter Haddaway. An overview of some recent developments in Bayesian problem solving techniques. *AI Magazine*, Spring 1999.

[11] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990.

[12] Manfred Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 266–273, San Francisco, CA, 1997. Morgan Kaufmann Publishers.

[13] Daphne Koller and Avi Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, August 1997.

[14] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[15] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society A*, 50(2):157–224, 1988.

[16] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[17] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, page 29. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[18] Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[19] Raymond Ng and V.S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *Journal of Automated Reasoning*, 10(2):191–235, 1993.

[20] Raymond Ng and V.S. Subrahmanian. Stable semantics for probabilistic databases. *Information and Computation*, 110(1):42–83, 1994.

[21] L. Ngo and Peter Haddaway. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–171, 1997. Special issue on uncertainty in databases and deductive systems.

[22] Liem Ngo, Peter Haddawy, and James Helwig. A theorectical framework for context-sensitive temporal probability model construction with application to plan projection. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 419–426, Montreal, Quebec, Canada, 1995.

[23] Fernando C. N. Pereira and Stuart M. Shieber. *Prolog and Natural-Language Analysis.* CLSI, Stanford, 1987.

[24] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, April 1997.

[25] Karl R. Popper and David Miller. Why probabilistic support is not inductive. *Philosophical Transactions of the Royal Society of London*, 321:569–591, 1987.

[26] Stefan Riezler. Probabilistic constraint logic programming. Arbeitsberichte des SFB 340 Bericht Nr. 117, Universität Tübingen, 1997.

[27] Glenn Shafer. *The Art of Causal Conjecture.* MIT Press, Cambridge, Mass., 1996.

[28] Ehud Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In *Proc. IJCAI-83*, pages 529–532, 1983.

[29] D. Spiegelhalter, A. Thomas, N. Best, and W. Gilks. *BUGS 0.5 Bayesian inference using Gibbs Sampling Manual.* MRC Biostatistics Unit, Cambridge, 1996.

[30] S. K. M. Wong, C. J. Butz, and Y. Xiang. A method for implementing a probabilistic model as a relational database. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 556–564, Montreal, Quebec, Canada, 1995.

[31] Esteban Zimányi. Query evaluation in probabilistic relational databases. *Theoretical Computer Science*, 171:179–219, 1997.

# A Code

```
asia(A,B,D,E,L,S,T,X,Prob) :-
        c6(E,X,P6),
        c5(E,B,D,P5),
        c4(L,B,S,P4),
        c3(L,E,B,P3),
        c2(E,L,T,P2),
        c1(A,T,P1),
        Prob is P1*P2*P3*P4*P5*P6.

c1(t,t,0.0005).
c1(t,f,0.0095).
c1(f,t,0.0099).
c1(f,f,0.9801).

c2(t,t,t,1).
c2(t,t,f,0).
c2(t,f,t,1).
c2(t,f,f,0).
c2(f,t,t,1).
c2(f,t,f,0).
c2(f,f,t,0).
c2(f,f,f,1).

c3(t,t,t,1).
c3(t,t,f,1).
c3(t,f,t,1).
c3(t,f,f,1).
c3(f,t,t,1).
c3(f,t,f,1).
c3(f,f,t,1).
c3(f,f,f,1).

c4(t,t,t,0.0300).
c4(t,t,f,0.0015).
c4(t,f,t,0.0200).
c4(t,f,f,0.0035).
c4(f,t,t,0.2700).
c4(f,t,f,0.1485).
c4(f,f,t,0.1800).
c4(f,f,f,0.3465).

c5(t,t,t,0.9).
c5(t,t,f,0.1).
c5(t,f,t,0.7).
c5(t,f,f,0.3).
c5(f,t,t,0.8).
c5(f,t,f,0.2).
c5(f,f,t,0.1).
c5(f,f,f,0.9).

c6(t,t,0.98).
c6(t,f,0.02).
c6(f,t,0.05).
c6(f,f,0.95).
```

Figure 12: “Asia” SLP implemented in Prolog

```
anbncn(A,Prob) :- build(A-B,B-C,C-[],Prob).

build(A-A,B-B,C-C,0.3).
build([a|A]-Ap,[b|B]-Bp,[c|C]-Cp,Prob) :-
        build(A-Ap,B-Bp,C-Cp,Prob2),
        Prob is Prob2*0.7.
```

Figure 13: Non context-free grammar SLP implemented in Prolog

```
n(A,B,C,Prob) :-
        net(A,B,C,Prob).
n(A,d,C,Prob) :-
        c1(A,d,P1),
        c2(d,C,P2),
        Prob is P1*P2.


net(A,B,C,P) :-
        member(B,[a,c,d,x]),
        c1(A,C,P).
net(A,B,C,P) :-
        member(A,[a,c,d,x]),
        member(C,[a,c,d,x]),
        c2(B,B,P).

c1(a,d,3).
c1(x,c,4).

c2(x,c,1).
c2(d,X,5) :- member(X,[a,c,d,x]).

member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

Figure 14: Non Markov Net, non-recursive SLP