Integrating by Separating: Combining Probability and Logic with ICL, PRISM and SLPs

James Cussens
Department of Computer Science
University of York
Heslington, York, YO10 5DD, UK
jc@cs.york.ac.uk

January 25, 2005

Abstract

This report describes the close relationship that obtains between the ICL, PRISM and SLP frameworks. The common feature of these frameworks is that a purely probabilistic component and a purely logical component are connected to produce a hybrid model. A hidden Markov model (HMM) is used as a running example. The Uniqueness Condition, which allows these frameworks to represent statistical models is discussed, and the consequences of using a weakened version of the Uniqueness Condition briefly explored. 'Lazy' sampling, based on SLD-resolution, is discussed.

1 Introduction

The goal of this article is to describe the very close relationship that exists between three existing frameworks for combining probability and logic:

1. The independent choice logic (ICL) (Poole, 1993b, 1997)
2. Programming in statistical modelling (PRISM) (Sato, 1995; Sato & Kameya, 2001)
3. Stochastic logic programs (SLPs) (Muggleton, 1996; Cussens, 2001)

The basic idea in these three frameworks, which is explicit in ICL and PRISM, but implicit in SLPs, is nicely summed up by Sato (1995):

> When a joint distribution P_F is given to a set F of facts in a logic program $DB = F \cup R$ where R is a set of rules, we can further extend it to a joint distribution P_{DB} over the set of least models of DB.

Logical rules are used to *deterministically map* a simple probability distribution (P_F) to a more complex one (P_{DB}). In this article, these distributions will be referred to as the *base distribution* and the *induced distribution*, respectively. The connection between probability (a distribution over the facts F) and logic (the rules R) is effected by plugging the probabilistically chosen truth values of the facts into the rules and seeing what follows. There now follows a more detailed description of this modular approach to the integration of probability and logic.

2 Extending joint instantiations using logic

Recall that each fact (ground atomic formula) has a truth-value of either true or false. Specifying a joint distribution over the set of facts amounts to treating each fact as a binary random variable. The joint distribution gives a probability for each joint truth-value assignment to the facts. In ICL and PRISM, each rule in R is forbidden from having a head which unifies with a fact in F. There are thus two disjoint sets of ground atomic formula in the language: those in F and those that come from using the rules R.

A restriction to binary random variables can prove inconvenient, so Poole introduced the idea of *alternatives*: sets of atomic formulae where exactly one member can be true. This has the effect of allowing random variables with more than two values to be encoded into a logical representation.

We will use an example from Poole (1997). (Roughly speaking, Poole uses the symbol $\mathcal{C}$ where Sato uses F, and $\mathcal{F}$ where Sato uses R. We will stick with Poole's original notation for this example.) Let

$$\mathcal{C} = \{\{a_1, a_2, a_3\}, \{b_1, b_2\}\}$$

define two alternatives. In the first alternative exactly one a_i can be true and in the second exactly one b_i can be true. In this introductory example the atomic formulae are propositional formulae, but in general they will be first-order formulae.

Probabilistically speaking, $\mathcal{C}$ defines values for two random variables, the first taking values in the set $\{a_1, a_2, a_3\}$ and the second taking values in $\{b_1, b_2\}$. Specifying, for a particular alternative, which atomic formula is

true is called an *atomic choice*. Specifying atomic choices for all alternatives is called a *total choice*. A total choice is thus a joint instantiation. In our running example there are six possible total choices: $\{a_1, b_1\}$, $\{a_2, b_1\}$, $\{a_3, b_1\}$, $\{a_1, b_2\}$, $\{a_2, b_2\}$ and $\{a_3, b_2\}$. Each total choice can be thought of as a 'possible world'.

Continuing the example from (Poole, 1997), let $\mathcal{F}$ be a set of rules defined as follows:

$$\mathcal{F} = \{c \leftarrow a_1 \wedge b_1, c \leftarrow a_3 \wedge b_2, d \leftarrow a_1, d \leftarrow \neg a_2 \wedge b_1, e \leftarrow c, e \leftarrow \neg d\}$$

There is nothing probabilistic about the rules $\mathcal{F}$: any inferences made using rules in $\mathcal{F}$ are entirely logical. This is the case even though the truth-values of body literals can be probabilistic. To put it another way, the rules are true in all possible worlds.

At this point it is crucial to bring in some sort of Closed World Assumption (CWA) so that the truth value of *each* formulae in the language is determined by a total choice. Once the CWA is assumed, a total choice determines a *single* (minimal) model, and thus the truth-value for *any* formula, not just those involved in the alternatives. To get the truth-value for any formula given a total choice we just see whether the formula is true in the unique minimal model determined by the total choice. So we can now tabulate what is and isn't true in each possible world of our example:

$$\begin{array}{lcccccccc}
w_{a_1,b_1} & \models & a_1 & \neg a_2 & \neg a_3 & b_1 & \neg b_2 & c & d & e \\
w_{a_2,b_1} & \models & \neg a_1 & a_2 & \neg a_3 & b_1 & \neg b_2 & \neg c & \neg d & e \\
w_{a_3,b_1} & \models & \neg a_1 & \neg a_2 & a_3 & b_1 & \neg b_2 & \neg c & d & \neg e \\
w_{a_1,b_2} & \models & a_1 & \neg a_2 & \neg a_3 & \neg b_1 & b_2 & \neg c & d & \neg e \\
w_{a_2,b_2} & \models & \neg a_1 & a_2 & \neg a_3 & \neg b_1 & b_2 & \neg c & \neg d & e \\
w_{a_3,b_2} & \models & \neg a_1 & \neg a_2 & a_3 & \neg b_1 & b_2 & c & \neg d & e
\end{array}$$

3 The base distribution

In principle, any base distribution could be defined over the facts. However, the only distributions actually used are those where the atomic choices are mutually independent. This explains why Poole's independent choice logic is so called. However, for a given alternative the distribution over possible atomic choices for that alternative can be an arbitrary multinomial. The restriction to independent atomic choices is not severe since a suitable choice of rules can be used to effect desired dependencies in the induced distribution. Note that this independence restriction makes it particularly simple to *sample* from the base distribution, and that since the induced distribution

is a *function* of the base distribution, any sample from the base distribution can be extended to a sample over the induced distribution using logical inference.

To make this more concrete, consider the PRISM pseudo-code fragment in Figure 1 which implements a hidden Markov model (HMM). This fragment is an edited version of an example that comes with the PRISM distribution. (The PRISM pseudo-code in Figure 1 is more complex than the actual PRISM code for reasons which will be explained shortly.)

In this example, for any $n \in \mathsf{N}$, five alternatives are defined: $msw(init, n)$, $msw(tr(s0), n)$, $msw(out(s0), n)$, $msw(tr(s1), n)$ and $msw(out(s1), n)$, so that there is a countably infinite set of alternatives in total. The HMM being modelled has two states: $s0$ and $s1$. $msw(init, 1)$ defines the distribution over the initial state. $msw(tr(s0), n)$ defines the distribution for the nth transition from state $s0$. $msw(out(s0), n)$ is the distribution for the nth emission from state $s0$. $msw(tr(s1), n)$ and $msw(out(s1), n)$ define the analogous distributions for state $s1$.

Recall that an 'alternative' is basically a random variable, so we have a countably infinite set of random variables. We dictate, as always, that these random variables are mutually independent. We are still left with the task of defining distributions for each of these random variables. However, since this is an HMM, the distribution for, say, $msw(tr(s0), n)$, is the same for all n, and similarly for $msw(init, n), msw(out(s0), n), msw(tr(s1), n)$ and $msw(out(s1), n)$. Also, naturally, the *values* for these random variables are the same for each n.

Since the values of the random variables are independent of n, the second argument of the `values/3` predicate, defined by the first three non-comment lines of Fig 1, is the variable n (written as `N` in the pseudo-code since PRISM uses Prolog-style syntax, where variables start with upper-case letters). Note that the variable n is (implicitly) universally quantified. States are also denoted by a variable (written as `S`) in the definition of `values/3`, since the values (if not the probabilities) associated with the two states are the same. Probabilities are defined using the `set_sw/3` predicate. Since all the probability distributions are independent of n, the second argument of `set_sw/3` is a universally quantified variable (written again as `N`).

Now consider a particular instantiation of some of the base random variables given in Table 1 below. Recall that a (discrete) HMM is a probabilistic finite state machine. This instantiation corresponds to a particular 'run' of the HMM and has probability $0.9 \times 0.5 \times 0.2 \times 0.5 \times 0.8 \times 0.4 \times 0.2 \times 0.7 \times 0.2 \times 0.6 \times 0.8 = 0.000193536$.

```
% multi-valued switch declarations:
values(init,N,[s0,s1]).  % there are two initial states {s0,s1}.
values(out(S),N,[a,b]).  % output symbols are {a,b} in every state.
values(tr(S),N,[s0,s1]). % transition destinations are {s0,s1} in every state.

set_params :-          % set parameters manually to the model.
   set_sw(init,N,   0.9+0.1),
   set_sw(tr(s0),N, 0.2+0.8),
   set_sw(out(s0),N,0.5+0.5),
   set_sw(tr(s1),N, 0.8+0.2),
   set_sw(out(s1),N,0.6+0.4).

%-------------------------------------
% Modeling part:

hmm(L):-                % this three line program can simulate all hmms with
   str_length(M),       % appropriate modifications to the settings of msws.
   msw(init,1,S),       % random choice of an initial state
   hmm(1,M,S,L,[1,1]).      % start stochastic transition

hmm(T,M,_,[],_,_) :-T>M,!.
hmm(T,M,S,[Ob|Y],N0,N1) :-   % current state is S, current time is T.
   whichn(S,N0,N1,N,NewN0,NewN1),
   msw(out(S),N,Ob),     % output in the state S is Ob.
   msw(tr(S),N,Next),    % transition from S to Next.
   T1 is T+1,          % count up time
   hmm(T1,M,Next,Y,NewN0,NewN1). % recursion

str_length(5).        % string length is 5

whichn(s0,N0,N1,N0,NewN0,N1) :-
   NewN0 is N0 + 1.
whichn(s1,N0,N1,N1,N0,NewN1) :-
   NewN1 is N1 + 1.
```

Figure 1: PRISM pseudo-code fragment implementing a hidden Markov model

Instantiation				Probability	Comment
msw(init,	1)	=	s0	0.9	Start in state s0
msw(out(s0),	1)	=	a	0.5	Emit an 'a' from state s0
msw(tr(s0),	1)	=	s0	0.2	Stay in state s0
msw(out(s0),	2)	=	a	0.5	Emit an 'a' from state s0
msw(tr(s0),	2)	=	s1	0.8	Move to state s1
msw(out(s1),	1)	=	b	0.4	Emit an 'b' from state s1
msw(tr(s1),	1)	=	s1	0.2	Stay in state s1
msw(out(s1),	2)	=	a	0.6	Emit an 'a' from state s1
msw(tr(s1),	2)	=	s1	0.2	Stay in state s1
msw(out(s1),	3)	=	a	0.6	Emit an 'a' from state s1
msw(tr(s1),	3)	=	s0	0.8	Move to state s0

Table 1: A partial instantiation of the base distribution

3.1 Simplifying the syntax

Due to the iid assumptions, the variable `N` in Figure 1 is doing no useful work, and keeping track of its correct value (via the `whichn/6` predicate) is an unnecessary burden. Real PRISM syntax therefore drops this argument, thus rendering the variable `N` *implicit* rather than explicit. Actual PRISM code (as opposed to pseudo-code) for the HMM is given in Figure 2. It is important to remember that, because of the implicit extra argument, the code in Figure 2 defines not 5 random variables but a countably infinite number of random variables. It is this which allows us to 'run' the HMM for as long as desired by setting `str_length` to an arbitrarily high number.

4 The induced distribution

Continuing with the PRISM HMM example, and thus switching back to Sato's notation, let R denote the rules of the program in Figure 1. If the implicit argument of $msw/3$ is made explicit once again, we have that:

$$\begin{array}{llllll} msw(init(1),s0,0.9) & \wedge & msw(out(s0),1,a) & \wedge & msw(tr(s0),1,s0) & \wedge \\ msw(out(s0),2,a) & \wedge & msw(tr(s0),2,s1) & \wedge & msw(out(s1),1,b) & \wedge \\ msw(tr(s1),1,s1) & \wedge & msw(out(s1),2,a) & \wedge & msw(tr(s1),2,s1) & \wedge \\ msw(out(s1),3,a) & \wedge & msw(tr(s1),3,s0) & \wedge & R & \\ & \vdash & hmm([a,a,b,a,a]) & & & \end{array} \tag{1}$$

It follows that in P_{BD}, the *induced* distribution, $hmm([a,a,b,a,a])$ has

```
% multi-valued switch declarations:
values(init,[s0,s1]).  % there are two initial states {s0,s1}.
values(out(S),[a,b]).  % output symbols are {a,b} in every state.
values(tr(S),[s0,s1]). % transition destinations are {s0,s1} in every state.

set_params :-          % set parameters manually to the model.
   set_sw(init,   0.9+0.1),
   set_sw(tr(s0), 0.2+0.8),
   set_sw(out(s0),0.5+0.5),
   set_sw(tr(s1), 0.8+0.2),
   set_sw(out(s1),0.6+0.4).

%-------------------------------------
% Modeling part:

hmm(L):-                 % this three line program can simulate all hmms with
   str_length(M),        % appropriate modifications to the settings of msws.
   msw(init,S),        % random choice of an initial state
   hmm(1,M,S,L).       % start stochastic transition

hmm(T,M,_,[]) :-T>M,!.
hmm(T,M,S,[Ob|Y]) :-   % current state is S, current time is T.
   msw(out(S),Ob),     % output in the state S is Ob.
   msw(tr(S),Next),    % transition from S to Next.
   T1 is T+1,          % count up time
   hmm(T1,M,Next,Y).   % recursion

str_length(5).        % string length is 5
```

Figure 2: PRISM code fragment implementing a hidden Markov model

probability at least $0.9 \times 0.5 \times 0.2 \times 0.5 \times 0.8 \times 0.4 \times 0.2 \times 0.7 \times 0.2 \times 0.6 \times 0.8 = 0.000193536$, since it follows from the conjunction given in (1) which, as previously described, has this probability in the base distribution. To work out the precise value of $P_{DB}(hmm([a,a,b,a,a]))$ we need to consider the probabilities of all other instantiations which together with R imply $hmm([a,a,b,a,a])$. These *together with the closed world assumption* determine $P_{DB}(hmm([a,a,b,a,a]))$.

4.1 Sampling from success sets in the induced distribution using proof-theoretic notions

Consider the logic program in Fig 3. It defines a predicate `hmm/1` such that `hmm(out)` is true if `out` is a first-order term representing a possible output from the HMM that the PRISM program in Fig 2 models. The set of all such terms `out` is the *success set for `hmm/1`*.

It is not difficult to see, by inspecting the code in Figure 2, that for any full joint instantiation of the base distribution, exactly one atomic formulae from the success set for `hmm/1` is logically implied. Code of this sort is said to meet the *Uniqueness Condition* (with respect to atomic formulae with predicate symbol `hmm/1`) (Sato & Kameya, 2001). Due to the Uniqueness Condition, the induced distribution defined in Fig 2 provides a distribution over this success set. PRISM programs which satisfy the Uniqueness Condition are appropriate for defining (discrete) statistical models.

Note that only a finite subset of any full joint instantiation of the base distribution is required to determine the `hmm/1` formula which that full joint instantiation (uniquely) implies. If all the atomic formulae on the LHS of (1) are true then $hmm([a,a,b,a,a])$ is true, irrespective of the truth values of any other $msw/3$ atomic formulae.

Fortunately, if we wish to sample from the success set of $hmm/1$ it is easy to generate only a partial instantiation from the base distribution—in fact, exactly enough to imply a `hmm(out)` atomic formula for some term `out`. This is *lazy sampling*. The partial instantiation from the base distribution is produced sequentially by sampling from a sequence of $msw/3$ random variables. Because of the iid assumptions the distribution for each sample in this sequence is independent of previously sampled values. However, *which particular* $msw/3$ random variable is sampled next is not merely dependent on but *entirely determined* by previous sampled values. This 'lazy' approach was described by (Poole, 1993a) who goes further to present an anytime algorithm such that approximate probabilities are always available, with computations that will not improve the approximation much put to the

```
% multi-valued switch declarations:
values(init,[s0,s1]).  % there are two initial states {s0,s1}.
values(out(S),[a,b]).  % output symbols are {a,b} in every state.
values(tr(S),[s0,s1]). % transition destinations are {s0,s1} in every state.

msw(MSW,V) :-
   values(MSW,Values),
   member(V,Values).

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

%-----------------------------------
% Modeling part:

hmm(L):-                 % this three line program can simulate all hmms with
   str_length(M),        % appropriate modifications to the settings of msws.
   msw(init,S),        % random choice of an initial state
   hmm(1,M,S,L).       % start stochastic transition

hmm(T,M,_,[]) :-T>M,!.
hmm(T,M,S,[Ob|Y]) :-   % current state is S, current time is T.
   msw(out(S),Ob),     % output in the state S is Ob.
   msw(tr(S),Next),    % transition from S to Next.
   T1 is T+1,          % count up time
   hmm(T1,M,Next,Y).   % recursion

str_length(5).        % string length is 5
```

Figure 3: Prolog code defining possible outputs from a hidden Markov model

back of a prioriy queue.

It is easy to see how to do lazy sampling by considering SLD-resolution proofs of the formula $\exists Out : hmm(Out)$ in the logic program in Fig 3. (Strictly speaking such proofs are refutations of the negation of this formula, but here we gloss over that.) There are a number of such proofs and each one not only proves the existence statement, but constructs a particular term `out` such that `hmm(out)` is true.

There is a choice of proofs in Fig 3 because the SLD-resolution procedure will (often) have a choice of which `member/2` clause to use, leading to different instantiations for `V` in `msw(MSW,V)` and ultimately different instantiations for `Out` in `hmm(Out)`. It is not difficult to see that *there is an isomorphism between proofs and partial instantiations of the base distribution*, since each $msw/3$ corresponds to a proof choice.

SLD-resolution is a top-down goal-driven procedure, where the goal is to find a proof. Unless the prover is compelled to backtrack (more on this later), nothing is proved which does not contribute to proving the top-level query. Lazy sampling inherits this goal-directness: by specifying a top-level query (namely, does there $\exists Out$ such that $hmm(Out)$?) and replacing Prolog's deterministic search for a proof by a probabilistic one, just enough of the base distribution is sampled to sample a member of $hmm/1$'s success set. (Note that the laziness of goal-directed proof mechanisms is similarly exploited in probabilistic-logical approaches which build Bayes nets 'on the fly' from logical knowledge bases. PLP (Haddaway, 1994) and BLPs (Kersting & De Raedt, 2001) are examples of this approach.)

5 Stochastic logic programs

Stochastic logic programs (SLPs) can (at least in their 'non-extended' version) be seen as special cases of ICL/PRISM models which automatically guarantee (a weakened version of) the Uniqueness Condition. Fig 4 shows an SLP which implements the same HMM model as our previous PRISM examples.

The SLP in Fig 4 is an 'impure' SLP (Cussens, 2001) since not all clauses have attached probability labels. However, (roughly speaking) all the clauses that matter do have these labels. SLP syntax brings a proof-theoretic interpretation to the fore: whenever an SLD-resolution procedure has to choose between clauses, the choice is made according to probability labels. If all clauses are labelled, so that the SLP is 'pure', then this distribution over clause choices defines a distribution over instantiations of any top-level goal.

```
0.9 : msw_init(s0).
0.1 : msw_init(s1).

0.2 : msw_trs0(s0).
0.8 : msw_trs0(s1).

0.8 : msw_trs1(s0).
0.2 : msw_trs1(s1).

0.5 : msw_outs0(a).
0.5 : msw_outs0(b).

0.6 : msw_outs1(a).
0.4 : msw_outs1(b).



%-------------------------------------
% Modeling part:

hmm(L):-             % this three line program can simulate all hmms with
   str_length(M),
   msw_init(S),        % random choice of an initial state
   hmm(1,M,S,L).       % start stochastic transition

hmm(T,M,_,[]) :-T>M,!.
hmm(T,M,S,[Ob|Y]) :-
   hmm_more(T,M,S,[Ob|Y]).

hmm_more(T,M,S,[Ob|Y]) :-
   (
     S=s0
   ->
     msw_outs0(Ob),
     msw_trs0(Next)
   ;
     msw_outs1(Ob)
     msw_trs1(Next)
   ),
   T1 is T+1,          % count up time
   hmm(T1,M,Next,Y).   % recursion

str_length(5).        % string length is 5
```


Figure 4: SLP code fragment implementing a hidden Markov model

If the SLP is impure, then additional restriction are necessary for this to be the case (Cussens, 2001, Section 3.3). (Our current example meets these restrictions.)

6 Relaxing the Uniqueness Condition

The Uniqueness Condition—that exactly one atomic formulae representing observed data is derivable from any instantiation of the base distribution—can be rather severe. It essentially requires that the set of 'rules' R, i.e. the logical part of the model, is *failure-free*. As a result, the resulting distribution over observables is essentially that defined by a stochastic-context free grammar (SCFG).

The consequence of relaxing the uniqueness assumption so that *at most one* observable follows from any instantiation of the base distribution, is explored at some length in (Cussens, 2001). The resulting distributions over observables become log-linear models, where the partition function Z corresponds to the probability of choosing a clause which leads to a 'dead-end' in the search for a proof. If backtracking were not permitted then such dead-ends would lead to the SLD-resolution procedure failing. Efficient sampling from log-linear distributions is much harder than for the failure-free case. (Indeed this difficulty was the original motivation for pioneering work on MCMC approaches to sampling.) Parameter estimation also becomes more difficult. The EM algorithm is still usable (as in the failure-free case), but further 'hidden data' must be taken into consideration. The resulting version of EM (named *failure-adjusted maximisation (FAM)*) is presented in (Cussens, 2001) and has been incorporated into the recently released PRISM 1.8 system.

Fig 5 is an example of a *constrained* HMM model. This humorous example models a professor probabilistically choosing restaurants (states) and lunches (emissions), subject to the constraint that he does not consume too many (< 4000) calories. This model only satisfies the relaxed Uniqueness Condition, since some instantiations of the base distribution do not imply any $chmm/4$ atomic formulae; they correspond to failures. This example comes with the PRISM 1.8 distribution.

7 Backtrackable sampling

Removing the failure-free/strong Uniqueness Condition greatly extends what can be modelled with ICL/PRISM/SLP models, but the resulting log-linear

```
%-------------------------------------
% Directives:

target(failure,0).
data(user).

values(tr(s0),[s0,s1]).
values(tr(s1),[s1,s0]).

values(lunch(s0),[p,s]).  % pizza:900,  sandwich:400
values(lunch(s1),[h,s]).  % hanburger:400, sandwich:500

%-------------------------------------
% Model:

failure:- not(success).
success:- success(_).
success(L) :- chmm(L,s0,0,7).
failure(L):- not(success(L)).

chmm(L,S,C,N):-
  N>0,
  msw(tr(S),S2),
  msw(lunch(S),D),
  ( S == s0,
     ( D = p, C2 is C+900
     ; D = s, C2 is C+400 )
  ; S == s1,
     ( D = h, C2 is C+400
     ; D = s, C2 is C+500 )
  ),
  L=[D|L2],
  N2 is N-1,
  chmm(L2,S2,C2,N2).

chmm([],_,C,0):- C < 4000.
```

Figure 5: PRISM model fragment for a constrained HMM

distributions are difficult to sample form. Another distribution over observables arises if we allow the SLD-resolution procedure to backtrack. *Backtrackable sampling* is simply Prolog except that the order in which clauses are tried is probabilistic, leading to a probability distribution over observables defined by this sampling procedure. Due to the backtracking a rudimentary probabilistic search algorithm is defined by backtrackable sampling. However, unlike normal probabilistic search algorithms (such as simulated annealing) the purpose of backtrackable sampling is not to find some optimal answer, but to define a probability distribution over answers from which it is easy to sample. It has thus been used in Markov chain Monte Carlo applications of SLPs (Angelopoulos & Cussens, 2001).

Note that because SLD-resolution is sound and at least refutation-complete it is possible to switch between proof-theoretic and model-theoretic views of the probability distributions defined by these models. In the case of backtrackable sampling, the proof-theoretic (= sampling) view is the easiest way of defining the distribution, but in contrast to the log-linear distribution such a view does not provide an analytic description of the distribution. Returning to a more model-theoretic interpretation—such as is found in the original ICL—backtrackable sampling corresponds to re-sampling some of the random variables (= alternatives) whose original sampled values have defined a 'world' within which none of the atomic formulae representing observables are true. In backtrackable sampling, the random variables are implicitly ordered, so that it is always the most recently sampled random variable that is re-sampled. The re-sampling is constrained so that previously sampled values (which have lead to 'failure') cannot be sampled. This approach could be extended to re-sampling schemes which do not mimic Prolog so faithfully; for example, one could choose which random variable to re-sample according to some probability distribution.

Acknowledgements

This work is supported by UK EPSRC Grant Reference: GR/S30993/01 *Stochastic Logic Programs for MCMC* and also by *Application of Probabilistic Inductive Logic Programming II* funded by the European Commission under the Sixth Framework Programme (2002-2006).

This report came out out of work done in preparation for an invited talk at ILP04, so thanks also to that conference for providing the necessary stimulus. Thanks also to Nicos Angelopoulos, Taisuke Sato and David Poole for discussions.

References

Angelopoulos, N., & Cussens, J. (2001). Markov chain Monte Carlo using tree-based priors on model structure. In Breese, J., & Koller, D. (Eds.), *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–2001)* Seattle. Morgan Kaufmann.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning*, *44*(3), 245–271.

Haddaway, P. (1994). Generating Bayesian networks from probabilistic knowledge bases. In López de Mántaras, R., & Poole, D. (Eds.), *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, pp. 262–269 Seattle, USA. Morgan Kaufmann.

Kersting, K., & De Raedt, L. (2001). Bayesian logic programs. Tech. rep. 151, University of Freiburg.

Muggleton, S. (1996). Stochastic logic programs. In De Raedt, L. (Ed.), *Advances in Inductive Logic Programming*, Vol. 32 of *Frontiers in Artificial Intelligence and Applications*, pp. 254–264. IOS Press, Amsterdam.

Poole, D. (1993a). Logic programming, abduction and probability: A top-down anytime algorithm for computing prior and posterior probabilities. *New Generation Computing*, *11*(3–4), 377–400.

Poole, D. (1993b). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, *64*(1), 81–129.

Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, *94*(1–2), 5–56.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In Sterling, L. (Ed.), *Proc. Twelfth International Conference on Logic Programming*, pp. 715–729 Kanagawa, Japan. MIT Press.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, *15*, 391–454.