Probabilistic Logic Programming and Bayesian Networks [*]

Liem Ngo and Peter Haddawy
Department of Electrical Engineering and Computer Science
University of Wisconsin-Milwaukee
Milwaukee, WI 53201
{*liem, haddawy*}*@cs.uwm.edu*

Abstract

We present a probabilistic logic programming framework that allows the representation of conditional probabilities. While conditional probabilities are the most commonly used method for representing uncertainty in probabilistic expert systems, they have been largely neglected by work in quantitative logic programming. We define a fixpoint theory, declarative semantics, and proof procedure for the new class of probabilistic logic programs. Compared to other approaches to quantitative logic programming, we provide a true probabilistic framework with potential applications in probabilistic expert systems and decision support systems. We also discuss the relationship between such programs and Bayesian networks, thus moving toward a unification of two major approaches to automated reasoning.

To appear in Proceedings of the 1995 Asian Computing Science Conference, Pathumthani, Thailand, December 1995.

[*]This work was partially supported by NSF grant IRI-9509165.

1 Introduction

Reasoning under uncertainty is a topic of great importance to many areas of Computer Science. Of all approaches to reasoning under uncertainty, probability theory has the strongest theoretical foundations. In the quest to extend the framwork of logic programming to represent and reason with uncertain knowledge, there have been several attempts to add numeric representations of uncertainty to logic programming languages [21, 6, 3, 11, 14, 15, 13]. Of these attempts, the only one to use probability is the work of Ng and Subrahmanian [15]. In their framework, a probabilistic logic program is an annotated Horn program. A typical example clause in a probabilistic logic program, taken from [15], is $path(X, Y) : [0.85, 0.95] \leftarrow a(X, Y) : [1, 1]$, which says that if the probability that a type A connection is used lies in the interval [1,1] then the reliability of the path is between 0.85 and 0.95. As this example illustrates, their framework does not employ conditional probability, which is the most common way to quantify degrees of influence in probabilistic reasoning and probabilistic expert systems [18]. In [13] the authors allow clauses to be interpreted as conditional probability statements, but they consider only the consistency of such programs, and do not provide a query answering procedure.

Bayesian networks [18] have become the most popular method for representing and reasoning with probabilistic information [10]. An extended form of Bayesian networks, influence diagrams, are widely used in decision analysis [18]. The strengths of causal relationships in Bayesian networks and influence diagrams are specified with conditional probabilities. A prominent feature of Bayesian networks is that they allow computation of posterior probabilities and performance of systematic sensitivity analysis, which is important when the exact probability values are hard to obtain. Bayesian networks are used as the main representation and reasoning device in probabilistic diagnostic systems and expert systems.

Bayesian networks were originally presented as static graphical models: for a problem domain the relevant random variables are identified, a Bayesian network representing the relationships between the random variables is sketched, and probability values are assessed. Inference is then performed using the entire domain model even if only a portion is relevant to a given inference problem. Recently the approach known as knowledge-based model construction [22] has attempted to address this limitation by representing probabilistic information in a knowledge base using schematic variables and indexing schemes and constructing a network model tailored to each specific problem. The constructed network is a subset of the domain model represented by the collection of sentences in the knowledge base. Approaches to this area of research have either focused on practical model construction algorithms, neglecting formal aspects of the problem [8, 4], or focused on formal aspects of the knowledge base representation language, without presenting practical algorithms for constructing networks [19, 2]. In [16, 9], we propose both a theoretical framework and a procedure for constructing Bayesian networks from a set of conditional probabilistic sentences.

The purpose of this paper is two-fold. First, we propose an extension of logic programming which allows the representation of conditional probabilities and hence can be used to write probabilistic expert systems. Second, we investigate the relationship between probabilistic logic programs and Bayesian networks. While Poole [19] shows how to represent a discrete Bayesian network in his Probabilistic Horn Abduction framework, in this paper we address both sides of this relationship. First, we show how Bayesian networks can be represented easily and intuitively by our probabilistic logic programs. Second, we present a method for answering queries on the probabilistic logic programs by constructing Bayesian networks and then propogating probabilities on the networks. We provide a declarative semantics for probabilistic logic programs and prove that the constructed Bayesian networks faithfully reflect the declarative semantics.

2 Syntax

Throughout this paper, we use Pr and sometimes Pr_P, Pr_P^C to denote a probability distribution; $A, B, \ldots$ with possible subscripts to denote atoms; names with leading capital characters to denote domain variables; names with leading small characters to denote constants and $p, q, \ldots$ with possible subscripts to denote predicates. We use a first order language containing infinitely many variable symbols and finitely many constant, function and predicate symbols. We use HB to denote the Herbrand base of the language, which can be infinite. For convenience, we use comma instead of logical AND and semicolons to seperate the sentences in a list of sentences.

Each predicate represents a class of similar random variables. In the probability models we consider, each random variable can take values from a finite set and in each possible realization of the world, that variable can have one and only one value. For example, the variable *neighborhood* of a person X can have value *bad, average, good* and, in each possible realization of the world, one and only one of these three values can be true; the others must be false. We capture this property by requiring that each predicate have at least one attribute representing the *value* of the corresponding random variable. By convention we take this to be the last attribute. For example, the variable *neighborhood* of a person X can be represented by a two-position predicate $neighborhood(X, V)$—the first position indicates the person and the second indicates the type of that person's neighborhood (bad, average or good). We associate with each predicate a *value integrity constraint* statement.

Definition 1 *The* **value integrity constraint** *statement associated with an m-ary predicate p consists of the following first order sentences (1)* $p(X_1, \ldots, X_{m-1}, V) \rightarrow V = v_1 \vee \ldots \vee V = v_n$; *(2)* $\leftarrow p(X_1, \ldots, X_{m-1}, v_i), p(X_1, \ldots, X_{m-1}, v_j), \forall i, j : 1 \leq i \neq j \leq n$; *where* $n > 1, m \geq 1$ *are two integers,* $v_1, \ldots, v_n$ *are different constants, called the value constants, denoting the possible values of the random variables corresponding to p,* $X_1, \ldots, X_{m-1}$ *are different variable names and each sentence is universally quantified over the entire sentence. For convenience, we use* $EXCLUSIVE(p, v_1, \ldots, v_n)$ *to denote the above set of sentences.*

We use $=$ as the identity relation on HB and always assume our theories include Clark's Equality Theory [12]. We denote by $VAL(p)$ the set $\{v_1, \ldots, v_n\}$. If A is an atom of predicate p, we also use $VAL(A)$ as equivalent to $VAL(p)$. If A is the ground atom $p(t_1, \ldots, t_{m-1}, v)$ then $val(A)$ denotes the value v and $obj(A)$ denotes the random variable corresponding to $(p, t_1, \ldots, t_{m-1})$.

We require such a value integrity constraint for each predicate. The set of all the integrity constraints is denoted by IC.

Example 1 *The value integrity constraint for the predicate neighborhood is*
$EXCLUSIVE(neighborhood, bad, average, good) =$
$\{\leftarrow neighborhood(X, bad), neighborhood(X, average);$
$\leftarrow neighborhood(X, bad), neighborhood(X, good);$
$\leftarrow neighborhood(X, average), neighborhood(X, good);$
$neighborhood(X, V) \rightarrow V = bad \vee V = average \vee V = good\}.$
For a person, say named John, $neighborhood(john, good)$ means the random variable neighborhood of John, indicated in the language by $obj(neighborhood(john, good))$, is good, *indicated in the language by $val(neighborhood(john, good)) = good$. In any possible world, one and only one of the following atoms is true: $neighborhood(john, bad)$, $neighborhood(john, average)$, or $neighborhood(john, good)$. $VAL(neighborhood)$, or $VAL(neighborhood(john, bad))$, is the set $\{bad, average, good\}$.*

We have two kinds of constants. The value constants are declared by EXCLUSIVE clauses and used as the last arguments of predicates. The non-value constants are used for the other predicate arguments.

Definition 2 *Let A be the ground atom $p(t_1, \ldots, t_m)$. We define $Ext(A)$, the* **extension** *of A, to be the set $\{p(t_1, \ldots, t_{m-1}, v) | v \in VAL(p)\}$.*

Example 2 *In the burglary example, $Ext(neighborhood(john, bad)) =$ $\{neighborhood(john, bad),\ neighborhood(john, average),\ neighborhood(john, good)\}$.*

Let A be an atom. We define $ground(A)$ to be the set of all ground instances of A. A set of ground atoms $\{A_i | 1 \leq i \leq n\}$ is called *coherent* if there do not exist any A_j and $A_{j'}$ such that $j \neq j'$ and $obj(A_j) = obj(A_{j'})$ (and $val(A_j) \neq val(A_{j'})$).

Definition 3 *A* **probabilistic sentence** *has the form $Pr(A_0 | A_1, \ldots, A_n) = \alpha$ where $n \geq 0, 0 \leq \alpha \leq 1$, and A_i are atoms. The sentence can have free variables and each free variable is universally quantified over its entire scope. The meaning of such a sentence is: If $Pr(B_0 | B_1, \ldots, B_n) = \alpha$ is a ground instance of it then the conditional probability of $obj(B_0)$ achieving the value $val(B_0)$ given $obj(B_i)$ having value $val(B_i), \forall i : 1 \leq i \leq n$, is α.*

Let S be the sentence $Pr(A_0 | A_1, \ldots, A_n) = \alpha$. We use $ante(S)$, the antecedent of S, to denote the conjunction $A_1 \wedge \ldots \wedge A_n$ and $cons(S)$, the consequent of S, to denote A_0. Sometimes, we use $ante(S)$ as the set of conjuncts.

An alternative representation of the probability sentence $Pr(A_0 | A_1, \ldots, A_n) = \alpha$ is $A_0 \leftarrow A_1, \ldots, A_n : \alpha$, where α is a value associated with the entire sentence. We will stick with the form in the definition but mention the alternative representation to highlight the resemblance to quantitative logic program clauses. Notice that by using predicates with value attribute and integrity constraints, we can explicitly represent 'negative facts'.

$$\begin{array}{lll}
IC & = & EXCLUSIVE(neighborhood, bad, average, good) \cup \\
 & & EXCLUSIVE(burglary, yes, no) \cup EXCLUSIVE(alarm, yes, no) \cup \\
 & & EXCLUSIVE(tornado, yes, no) \\
PB & = \{ & Pr(neighborhood(john, average)) = .4; \\
 & & Pr(neighborhood(john, bad)) = .2; Pr(neighborhood(john, good)) = .4; \\
 & & Pr(burglary(X, yes)|neighborhood(X, average)) = .4; \\
 & & Pr(burglary(X, yes)|neighborhood(X, good)) = .2; \\
 & & Pr(burglary(X, yes)|neighborhood(X, bad)) = .4; \\
 & & Pr(alarm(X, yes)|burglary(X, yes)) = .98; Pr(alarm(X, yes)|burglary(X, no)) = .05; \\
 & & Pr(alarm(X, yes)|tornado(X, yes)) = .99; Pr(alarm(X, yes)|tornado(X, no)) = .15\}
\end{array}$$

Figure 1: A Basic Probabilistic Logic Program.

2.1 Basic Probabilistic Logic Programs

Definition 4 *A* **basic (probabilistic logic) program** *consists of two parts: the probabilistic base PB is a finite set of probabilistic sentences and the set IC of value integrity constraints for the predicates in the language.*

Consider the following motivating example, which will be referred to throughout the remainder of the paper. A burglary alarm could be triggered by a burglary or a tornado. The likelihood of a burglary is influenced by the type of neighborhood one resides in. Figure 1 shows a possible basic probabilistic logic program for representing this example. We have the following predicates: *neighborhood*, *burglary*, *alarm*, and *tornado*. The interpretation of statements in IC is similar to that of $EXCLUSIVE(neighborhood, bad, average, good)$ shown in a previous example.

2.2 Acyclic Probabilistic Bases

In this paper, major results are achieved for a class of programs characterized by acyclicity. A probabilistic base PB is called *acyclic* if there is a mapping $ord_{PB}()$ from the set of ground instances of atoms into the set of natural numbers such that (1) For any ground instance $Pr(A_0|A_1, \ldots, A_n) = \alpha$ of some sentence in PB, $ord_{PB}(A_0) > ord_{PB}(A_i), \forall i : 1 \leq i \leq n$. *(2) If A and A' are two ground atoms such that $A' \in Ext(A)$ then $ord_{PB}(A) = ord_{PB}(A')$.*

The expressiveness of acyclic logic programs is demonstrated in [1]. We expect that probabilistic logic programs with acyclic probabilistic bases will prove to have equal importance. To the best of our knowledge, knowledge bases of conditional probabilisties containing loops are considered problematic and all those considered in the literature are acyclic.

3 Fixpoint Semantics

3.1 The Relevant Atom Set

In this section, we consider the implications of the structure of basic probabilistic logic programs, ignoring the probability values associated with the sentences. We view the probabilistic sentence $Pr(A_0|A_1, \ldots, A_n) = \alpha$ as the Horn clause $A_0 \leftarrow A_1, \ldots, A_n$. Our purpose is to determine the set of *relevant atoms* implied by a program. For normal logic programs, fixpoint theory characterizes the semantics of a program by a *'minimal'* set of literals which is the fixpoint of a transformation constructed from its syntactic structure. That set consists of ground atoms that are considered true (and their negations false), and ground atoms that are considered false (and their negations true). Usually, there are other atoms whose truth values are undefined [7]. Similarly, from a basic probabilistic logic program, we can obtain (sometimes partial) probabilistic information about some ground atoms.

Example 3 *Consider the following basic program:*

$$\begin{array}{lll} IC & = & EXCLUSIVE(p, true, false) \cup EXCLUSIVE(q, bad, average, good) \cup \\ & & EXCLUSIVE(r, true, false) \cup EXCLUSIVE(s, true, false) \\ PB & = \{ & Pr(p(true)) = .4; \\ & & Pr(q(good)|p(true)) = .3; Pr(q(good)|p(false)) = .5; \\ & & Pr(r(true)|s(true)) = .6; Pr(r(true)|r(true)) = 1\} \end{array}$$

Using Bayes' rule, we can derive $Pr(q(good)) = Pr(q(good) \wedge p(true)) + Pr(q(good) \wedge p(false)) = Pr(q(good)|p(true)) * Pr(p(true)) + Pr(q(good)|p(false)) * Pr(p(false)) = .3 * .4 + .5 * .6 = .42$. *We know partial information about* $Pr(q(bad))$ *and* $Pr(q(average))$ *because* $Pr(q(bad)) + Pr(q(average)) = 1 - Pr(q(good)) = .58$. *But we do not know any probabilistic information about* $r(true), r(false), s(true)$ *and* $s(false)$ *(independently).*

Definition 5 *Given a basic program P. The* **fixpoint operator $\mathbf{T_P}$** *is defined as a mapping from* 2^{HB} *into* 2^{HB} *such that for all* $I \in 2^{HB}$, $T_P(I)$ *is the smallest set in* 2^{HB} *satisfying the following properties: (1) if* S *is a ground instance of a sentence in* P *such that* $ante(S)$ *is a subset of* I *then* $cons(S) \in T_P(I)$*; (2) if* $A \in T_P(I)$ *then* $Ext(A) \subseteq T_P(I)$.

The transformation T_P produces only *reflexive* subsets of HB. Such subsets are important to us because when we know (partial) probabilistic information about an atom A, we also know (partial) probabilistic information about each other atom in $Ext(A)$. A subset I of HB is a **reflexive subset** if $\forall A \in I, Ext(A) \subseteq I$. We consider the *space of reflexive subsets* of HB, denoted by RHB.

Proposition 1 *(1)* RHB *is a complete lattice w.r.t. the normal* $\subseteq$ *relation. (2)* T_P *is monotonic on* RHB, *i.e.* $\forall I, I' \in RHB$: *whenever* $I \subseteq I', T_P(I) \subseteq T_P(I')$.

We define a simple iterative process for applying T_P.

Definition 6 *Let* ι *range over the set of all countable ordinals. The upward sequences* $\{I_\iota\}$ *and* I_* *are defined recursively by: (1)* $I_0 = \{\}$. *(2) If* τ *is a limit ordinal,* $I_\tau = \cup_{\iota<\tau} I_\iota$. *(3) If* $\tau = \iota + 1, I_\tau = T_P(I_\iota)$. *(4) Finally,* $I_* = \cup_\iota T_P(I_\iota)$.

Example 4 *Continuing the example 3, the upward sequence $\{I_\iota\}$ is: $I_0 = \{\}$, $I_1 = \{p(true), p(false)\}, I_2 = I_1 \cup \{q(bad), q(average), q(good)\}, I_\iota = I_2, \forall \iota > 2$. I_2 is the set of all ground atoms whose (partial) probability information can be obtained from the program.*

The upward sequence $\{I_\iota\}$ is a monotonic sequence of elements in RHB. It follows by classical results of Tarski that the upward sequence converges to the least fixpoint.

Theorem 1 *The upward sequence $\{I_\iota\}$ converges to $lfp(T_P) = I_*$, the least fixpoint in RHB. Furthermore, if there are no function symbols in the language then the convergence occurs after a finite number of steps.*

We call $lfp(T_P)$ the *relevant set of atoms (RAS)*. RAS plays a similar role to well-founded partial models [7]. We use RAS to formalize the concept of possible worlds implied by a program. Let ι be a countable ordinal. An **ι-macro-world** of the logic program P is a maximal coherent subset of I_ι. A **possible world** is a maximal coherent subset of RAS. We use PW to denote the set of possible worlds. We can see that there always exist possible worlds for a program P.

Example 5 *Continuing the previous example, there are two 1-macro-worlds: $w_{11} = \{p(true)\}$ and $w_{12} = \{p(false)\}$. The possible worlds and also 2-macro-worlds are $w_{21} = \{p(true), q(good)\}; w_{22} = \{p(true), q(average)\}; w_{23} = \{p(true), q(bad)\}$; $w_{24} = \{p(false), q(good)\}; w_{25} = \{p(false), q(average)\}; w_{26} = \{p(false), q(bad)\}$.*

Let W be a possible world and $A \in W$. Then $W \cup IC$ derives $\neg A', \forall A' \in Ext(A)$ and $A' \neq A$. So, $W \cup IC$ represent a coherent assignment of values to the relevant random variables.

4 Combining Rules and Probabilistic Logic Programs

A basic probabilistic logic program will typically not be a complete specification of a probability distribution over the random variables represented by the atoms. One type of information which may be lacking is the specification of the probability of a variable given combinations of values of two or more variables which influence it. For real-world applications, this type of information can be difficult to obtain. For example, for two diseases D_1 and D_2 and a symptom S we may know $Pr(S|D_1)$ and $Pr(S|D_2)$ but not $Pr(S|D_1, D_2)$. Combining rules such as generalized noisy-OR [5, 20] are commonly used to construct such combined influences.

We define *a combining rule* as any algorithm that takes as input a (possibly infinite) set of ground probabilistic sentences with the same consequent $\{Pr(A_0|A_{i1}, \ldots, A_{in_i}) = \alpha_i | 1 \leq i \leq m(\text{m may be infinite})\}$ such that $\cup_{i=1}^{m}\{A_{i1}, \ldots, A_{in_i}\}$ is coherent and produces as output $Pr(A_0|A_1, \ldots, A_n) = \alpha$, where $A_1, \ldots$, and A_n are all different and n is a finite integer. In addition to the standard purpose of combining rules, we also use them as one kind of default rule to augment missing causes (a cause is an atom in the antecedent). In this case, the antecedents of the output contain atoms not in the input sentences. The set of output causes can be a proper subset of the set of input causes, in which case the combining rule is performing a filtering and summarizing task.

Example 6 *Assume two diseases D_1, D_2 and one symptom S, which are represented by predicates d_1, d_2, and s, respectively. Also assume D_1, D_2 and S have values normal and abnormal. A program might contain only the following sentences:*
$Pr(s(abnormal)|d1(abnormal)) = .9$, $Pr(s(abnormal)|d1(normal)) = .15$,
and $Pr(s(abnormal)|d2(normal)) = .2$. *We can provide combining rules to construct from the first and third sentences a new sentence of the form*
$Pr(s(abnormal)|d1(abnormal), d2(normal)) = \alpha$ *and from the second and third another new sentence of the form* $Pr(s(abnormal)|d1(normal), d2(normal)) = \beta$, *where α and β are two numbers determined by the combining rule. The combining rules may also act as* default rules *in augmenting the first and second sentences to achieve*
$Pr(s(abnormal)|d1(abnormal), d2(abnormal)) = \alpha'$
and $Pr(s(abnormal)|d1(normal), d2(abnormal)) = \beta'$, *for some values α' and β'.*

Definition 7 *A* **(probabilistic logic) program** *is a triple $\langle IC, PB, CR\rangle$, where $\langle IC, PB\rangle$ is a basic probabilisitic logic program and CR is a set of combining rules. We assume that for each predicate, there exists one corresponding combining rule in CR.*

The combining rules usually depend on the meaning of the program. In [17], we discuss the combining rules for interaction between effects of actions and persistence rules in planning problems.

5 The Combined Relevant Probabilistic Base

With the addition of combining rules, the real structure of a program changes. In this section, we consider the effect of combining rules on the relationships prescribed by the program.

Definition 8 *Given a program P, let ι be a countable ordinal. The* **set of ι-relevant probabilistic sentences** *(**ι-RPB**) is defined as the set of all ground instances S of some probabilistic sentence in PB, such that all atoms in S are in I_ι.*

The ι-RPB contains the basic relationships between atoms in I_ι. In the case of multiple influences represented by multiple sentences, we need combining rules to construct the combined probabilistic influence.

Definition 9 *Given a program P. Let ι be a countable ordinal. The* **combined ι-RPB (ι-CRPB)** *is constructed by applying the appropriate combining rules to each maximal set of sentences $\{S_i | i \in I\}$ (I maybe an infinite index set) in ι-RPB which have the same consequent and such that $\cup_{i \in I} ante(S_i)$ is coherent.*

Combined ι-RPB's play a similar role to completed logic programs. We assume that each sentence in ι-CRPB describes all random variables which directly influence the random variable in the consequent. We define a syntactic property of ι-CRPB which characterizes the completeness of probability specification.

Definition 10 *An ι-CRPB is* **completely quantified** *if*

(1) for all ground atoms A in I_ι, there exists at least one sentence in ι-CRPB with A in the consequent; and

(2) for all ground sentences S in ι-CRPB we have the following property: Let S have the form $Pr(A_0|A_1,\ldots,A_n)=\alpha$, then for all $i=0,..,n$, if $val(A_i)=v$ and $v'\in VAL(A_i), v\neq v'$, there exists another ground sentence S' in ι-CRPB such that S' can be constructed from S by replacing $val(A_i)$ by v' and α by some α'.

Definition 10 says that for each ground atom A we have a complete specification of the probability of all possible values $val(A)$ given all possible combinations of values of the atoms that directly influence A. If we think of each $obj(A)$ as representing a random variable in a Bayesian network model then the definition implies that we can construct a link matrix for each random variable in the model.

We call $*$-RPB the Relevant Probabilistic Base (RPB) and we call $*$-CRPB the Combined Relevant Probabilistic Base (CRPB).

Example 7 *Consider our burglary example and assume that the language contains only one non-value constant john.* RAS={ neighborhood(john,bad), neighborhood(john,average), neighborhood(john,good), burglary(john,yes), burglary(john,no), alarm(john,true), alarm(john,false), tornado(john,yes), tornado(john,no)}, and
RPB = {Pr(neighborhood(john,average))=.4; Pr(neighborhood(john,bad))=.2; Pr(neighborhood(john,good))=.4; Pr(burglary(john,yes)|neighborhood(john,average))=.4; Pr(burglary(john,yes)|neighborhood(john,good))=.2; Pr(burglary(john,yes)|neighborhood(john,bad))=.4; Pr(alarm(john,yes)|tornado(john,yes))=.99; Pr(alarm(john,yes)|burglary(john,yes))=.98; Pr(alarm(john,yes)|burglary(john,no))=.05; Pr(alarm(john,yes)|tornado(john,no))=.1}.
In the CRPB, the sentences in RPB with alarm as the consequent are transformed into sentences specifying the probability of alarm conditioned on both burglary and tornado. The other sentences in RPB remain the same in CRPB.

In conjunction with acyclicity property of probabilistic bases, we are interested in a class of combining rules which is capable of transferring the acyclicity property of a PB to the correspoding CRPB. Given a program P, we say a combining rule in CR is *self-contained* if the generated sentence $Pr(A|A_1,\ldots,A_n)=\alpha$ from the input set

$\{Pr(A|A_{i1},\ldots,A_{in_i})=\alpha_i|1\leq i\leq m(\text{m may be infinite})\}$

satisfies one additional property:

$\{A_1,\ldots,A_n\}\subseteq\cup_{A'\in Ext(A)}\{\{B_{i1},\ldots,B_{n_i}\}|\ Pr(A'|B_{i1},\ldots,B_{in_i})=\alpha_i \text{ is in RPB}\}$.

Self-containedness seems to be a reasonable assumption on a combining rule: it does not allow the generation of new atoms in the antecedent which are not 'related' to any atom in the extension of the consequent. In order to generate a sentence with consequent A, a self-contained combining rule may need to collect all the sentences which have an atom in $Ext(A)$ as consequent.

Example 8 *The combining rule in the example 6 is not self-contained because the sentence $Pr(s(abnormal)|d1(abnormal),d2(abnormal))=\alpha'$ is constructed from a set of sentences* *which do not contain the atom $d2(abnormal)$. For this kind of diagnosis problem, generalized noisy-OR rule [20] always assume that if a disease is in the abnormal state then there is a probability 1 that the symptom is abnormal, that means $Pr(s(abnormal)|d2(abnormal)) = 1$. In order to use self-contained combining rules, we need to write explicitly those sentences.*

6 Model Theory

The semantics of a probabilistic program is characterized by the probability weights assigned the ground atoms. That annotated approach is widely used in the related work [14, 15, 13]. Because of space limitation, we only show how to assign weights to ground atoms. Details on annotated models can be found in the full paper.

6.1 Probabilistic Independence Assumption

In addition to the probabilitistic quantities given in the program, we assume some probabilistic independence relationships specified by the structure of probabilistic sentences. Probabilistic independence assumptions are used in all probability model construction work [22, 4, 8, 19, 9] as the main device to construct a probability distribution from local conditional probabilities. Unlike Poole [19] who assumes independence on the set of consistent *"assumable"* atoms, we formulate the independence assumption in our framework by using the structure of the sentences in ι-CRPB. We find this approach more natural since the structure of the ι-CRPB tends to reflect the causal structure of the domain and independencies are naturally thought of causally.

Definition 11 *Given a set P of ground probabilistic sentences, let A and B be two ground atoms. We say A is* **influenced by** *B in P if (1) there exists a sentence S, an atom A' in $Ext(A)$ and an atom B' in $Ext(B)$ such that $A' = cons(S)$ and $B' \in ante(S)$ or (2) there exists another ground p-atom C such that A is influenced by C in P and C is influenced by B in P.*

Assumption *We assume that if $Pr(A|A_1, \ldots, A_n) = \alpha$ is in ι-CRPB then for all ground atoms B which are not in $Ext(A)$ and not* influenced by *A in ι-CRPB, A and B are probabilistically independent given $A_1, \ldots, A_n$.*

Example 9 *Continuing the burglary example, $alarm(john, yes)$ is probabilistically independent of $neighborhood(john, good)$ and $neighborhood(john, bad)$ given $burglary(john, yes)$ and $tornado(john, no)$.*

Definition 12 (Consistent ι-CRPB) *A completely quantified ι-CRPB is consistent if*
(1) there is no atom in I_ι which is influenced by itself in ι-CRPB and
(2) for all $Pr(A_0|A_1, \ldots, A_n) = \alpha$ in ι-CRPB, $\sum\{\alpha_i | Pr(A'_0|A_1, \ldots, A_n) = \alpha_i \in \iota\text{-CRPB and } obj(A'_0) = obj(A_0)\} = 1$.

6.2 Possible World Semantics

In this section, we allow the language to contain function symbols. There are, in general, infinitely many possible worlds, infinitely many ι-macro -worlds. We use an approach similar to that of Poole [19] by assigning weights to only certain subsets of worlds.

Definition 13 (Rank of an atom) *Let A be a ground atom in RAS. We define $rank(A)$, the rank of A, recursively by: (1) If A is not influenced (in CRPB) by any atom then $rank(A) = 0$, otherwise (2) $rank(A) = sup\{rank(B) | Pr(A| \ldots, B, \ldots)$ is in CRPB$\} + 1$.*

Example 10 *In the burglary example, $rank(tornado(.,.)) = rank(neighborhood(.,.)) = 0$, $rank(burglary(.,.)) = 1$ and $rank(alarm(.,.) = 2$.*

The program with the following CRPB has an atom which cannot be assigned a finite rank: $CRPB = \{Pr(q(true)|p(X, true)) = 1; Pr(p(X+1, true)|p(X, true)) = 1\}$. We cannot assign any finite rank to $q(true)$ because $rank(q(true)) > rank(p(X, true)), \forall X$.

We can see that if CRPB has no cycles then *rank* is a well-defined mapping. The following lemma will be useful in working with acyclic probabilistic bases.

Lemma 1 *Given a program P with an acyclic probabilistic base. If the combining rules are self-contained then the rank() function is well-defined.*

In defining the sample space, we will not consider individual possible world but sets of possible worlds characterized by formulae of specific forms.

Definition 14 *Given a program P, we can determine the set of all possible worlds PW. Assume that the rank function is well-defined. Let A be a ground atom in RAS. We denote the set of all possible worlds containing A by $W(A)$. We define the* **sample space $\omega_{\mathbf{P}}$** *to be the smallest set consisting of (1) $PW \in \omega_P$; (2) $\forall A \in RAS$ such that $rank(A)$ is finite, $W(A) \in \omega_P$; (3) if $W \in \omega_P$ then $PW - W \in \omega_P$; (4) if W_1, W_2 are in ω_P, then $W_1 \cap W_2$ is in ω_P.*

We consider the probability functions on the sample space ω_P. Let Pr be a probability function on the sample space, we define $Pr(A_1, \ldots, A_n)$, where $A_1, \ldots, A_n$ are atoms in RAS with finite ranks, as $Pr(\cap_{i=1}^{n} W(A_i))$. We take a sentence of the form $Pr(A_0|A_1, \ldots, A_n) = \alpha$ as shorthand for $Pr(A_0, A_1, \ldots, A_n) = \alpha \times Pr(A_1, \ldots, A_n)$. We say Pr() satisfies a sentence $Pr(A_0|A_1, \ldots, A_n) = \alpha$ if $Pr(A_0, A_1, \ldots, A_n) = \alpha \times Pr(A_1, \ldots, A_n)$ and $Pr()$ satisfies $CRPB$ if it satisfies every sentence in $CRPB$.

Definition 15 *A* **probability distribution induced by a program P** *is a probability distribution on ω_P satisfying CRPB and the independence assumption implied by CRPB.*

Example 11 *Consider the following program:*

$$\begin{array}{lll} IC & = & EXCLUSIVE(p, true, false) \cup EXCLUSIVE(q, bad, average, good) \\ PB & = \{ & Pr(p(0, true)) = .4; Pr(p(0, false)) = .6; \\ & & Pr(q(good)|p(T, true)) = .3; Pr(q(good)|p(T, false)) = .5; \\ & & Pr(p(T+1, true)|p(T, true)) = .999; Pr(p(T+1, false)|p(T, true)) = .001 \\ & & Pr(p(T+1, true)|p(T, false)) = .002; Pr(p(T+1, false)|p(T, false)) = .998\} \\ CR & = \{ & Generalized-Noisy-OR\} \end{array}$$

We can imagine that p is a timed predicate with the first attribute indicating time. The last four sentences represent persistence rules. We have $Pr(W(p(0, true))) = .4$, $Pr(W(p(0, false))) = .6$, $Pr(W(p(1, true))) = (.999 \times .4 + .002 \times .6) = .4008, \ldots$

Theorem 2 *Given a program P, if the CRPB is completely quantified and consistent then there exists one and only one induced probability distribution.*

The following theorem allows us to handle probability of conjunctions and disjunctions in our framework.

Theorem 3 *Given a program P. Any probability function on* ω_P *satisfying CRPB assigns a weight to any formula of the form* $\vee_{i=1}^{n} \wedge_{j=1}^{m} A_{ij}$, *where* n *and* m *are finite integers and* $rank(A_{ij})$ *is finite,* $\forall i, j$.

7 Fixpoint Theory Revisited

We now extend the fixpoint theory to include the quantatitive information given in a program. We have constructed in a previous section the transformation T_P and the upward sequence $\{I_\iota\}$. We associate with each I_ι a sample space and a probability distribution.

Definition 16 *Given a program P, we can determine the set of possible worlds PW. Assume that the rank function is well-defined and* ι *is a finite ordinal. We define the sample space* ω_P^ι *to be the smallest set consisting of (1)* $PW \in \omega_P$; *(2)* $\forall A \in I_\iota$ *such that* $rank(A) \leq \iota$, $W(A) \in \omega_P^\iota$; *(3) if* $W \in \omega_P^\iota$ *then* $PW - W \in \omega_P^\iota$; *(4) if* W_1, W_2 *are in* ω_P^ι *then* $W_1 \cap W_2$ *is in* ω_P^ι.

Proposition 2 *If* $\iota < \tau$ *are two finite ordinals then* $\omega_P^\iota \subseteq \omega_P^\tau \subseteq \omega_P$.

We define the probability functions on the sample space ω_P^ι induced by a program P by replacing RAS by I_ι and CRPB by ι-CRPB in the definitions of the previous section. We call the corresponding induced probability function Pr_ι.

Theorem 4 *If* $\iota < \tau$ *are two finite ordinals and* $W \in \omega_P^\iota$ *then* $Pr_\iota(W) = Pr_\tau(W) = Pr_P(W)$, *where* $Pr_P()$ *is the probability distribution induced by P and RAS.*

So, as the upward sequence $\{I_\iota\}$ "converges" to $lfp(T_P)$, $\{\omega_P^\iota\}$ converges to ω_P and $Pr_\iota()$ "converges" to $Pr_P()$. Here, we use a "loose" definition of convergence: for any finite first order formula F of ground finite rank atoms, there exists an integer n such that for all $\iota > n$, the set $W(F)$ of possible worlds satisfying F is an element of ω_P^ι and $Pr_\iota(W(F)) = Pr_P(W(F))$.

8 Proof Theory

In this section, we define a proof theory which can be used to derive the probability of a ground atom given a program. We will use G, with possible subscripts, to denote a goal atom. We will use a process similar to the SLD proof procedure with the only real difference being in the handling of combining rules. We call this proof procedure *probabilistic SLD* (p-SLD).

A query is a sentence of the form $Pr(G_1, \ldots, G_n) =?$, where G_i are atoms. The query is a request to find all ground instances $G'_1 \wedge \ldots \wedge G'_n$ of $G_1 \wedge \ldots \wedge G_n$ such that $Pr(G'_1 \wedge \ldots \wedge G'_n)$ can be determined from the program P and to return those probability values.

Definition 17 *Suppose $Pr(G_1, \ldots, G_n) =?$ is a query, called Q, and S is the sentence $Pr(A_0|A_1, \ldots, A_n) = \alpha$ in the program, and that the variables in Q and S are standardized apart. Let G_i be the* selected atom *in Q. Assume that θ is the most general unifier of A_0 and G_i. The* **resolvent** *of Q and S using (mgu) θ on G_i is the sentence*
$Pr(\ldots, G_{i-1}, A_1, \ldots, A_n, G_{i+1}, \ldots)\theta =?$

A p-SLD **derivation** *of the initial query Q_1 from a program P is a sequence $\langle Q_1, S_1, G_1, \theta_1 \rangle$, $\ldots$, $\langle Q_r, S_r, G_r, \theta_r \rangle, \ldots$, where $\forall i \geq 1, S_i$ is a renamed version of a sentence in P and Q_{i+1} is the resolvent of Q_i and S_i using θ_i on the selected atom G_i.*

An p-SLD refutation *of the query Q is an n-step p-SLD derivation of the initial query Q such that the resolvent of Q_n and S_n using θ_n is the empty query. The combined substitution $\theta_1 \ldots \theta_n$ is called the* computed answer substitution.

The p-SLD refutation tree *of the query Q is the set of all p-SLD refutations of the initial query Q.*

We need the concept of p-SLD refutation tree because before we can use the combining rules to construct the sentences with an atom A as consequent in CRPB, all sentences with consequent matching A in P need to be collected. Furthermore, we need to instantiate those sentences to ground before applying the combining rules.

Example 12 *For the sentence $Pr(q(true)|r(X, true)) = .1$, a combining rule needs to consider all ground sentences $Pr(q(true)|r(a_1, true)) = .1, Pr(q(true)|r(a_2, true)) = .1, \ldots$, where $a_1, a_2, \ldots$ are all constants in the language.*

Definition 18 *The* **ground p-SLD refutation tree** *of the query Q is the set of all ground p-SLD refutations of the initial query Q. A ground p-SLD refutation is obtained from a p-SLD refutation by first applying the associated computed answer substitution to each formula in the derivation and finally instantiating it to a possible ground instance.*

Let Q be the query $Pr(G) =?$. The ground p-SLD refutation tree of Q contains all the necessary ground probabilistic sentences to construct the combined sentences in CRPB whose consequents are ground instances of G or of the selected atoms in the original refutation trees. We apply the combining rules to it.

Definition 19 *Let Q be the query $Pr(G) =?$. The* **combined supporting set** *of Q is the set of ground probabilistic sentences constructed from the ground p-SLD refutation tree of Q by the following procedure: for each A, a (ground) selected atom or ground instance of G appearing in the tree, collect all (ground) sentences in it which have A as consequent and apply the appropriate combining rule to construct the combined sentence.*

The combining rules may generate new atoms which did not occur in the ground refutation tree, as in example 6. We need to apply the same process to these new atoms.

Definition 20 *Let Q be the query $Pr(G) =?$. The* **augmented combined supporting set** *of Q is constructed by augmenting the combined supporting set of Q in the following recursive way: starting from the combined supporting set of Q, for each atom A appearing in that set, if there is no sentence (in that set) with A as consequent then augment it with the augmented combined supporting set of the query $Pr(A) =?$.*

Example 13 *Continuing the example 11 with the query $Q : Pr(p(2, true)) =?$. The p-SLD refutation tree of Q is the following set of p-SLD refutations:*

$$\begin{array}{lll}
\{ \; \{ & \langle Pr(p(2,true)) =?, Pr(p(t+1,true)|p(t,true)) = .999, p(2,true), \{t|1\}\rangle, & \\
 & \langle Pr(p(1,true)) =?, Pr(p(t+1,true)|p(t,true)) = .999, p(1,true), \{t|0\}\rangle, & \\
 & \langle Pr(p(0,true)) =?, Pr(p(0,true)) = .4, p(0,true), \{\}\rangle & \}; \\
\quad \{ & \langle Pr(p(2,true)) =?, Pr(p(t+1,true)|p(t,true)) = .999, p(2,true), \{t|1\}\rangle, & \\
 & \langle Pr(p(1,true)) =?, Pr(p(t+1,true)|p(t,false)) = .002, p(1,true), \{t|0\}\rangle, & \\
 & \langle Pr(p(0,false)) =?, Pr(p(0,false)) = .6, p(0,false), \{\}\rangle & \}; \\
\quad \{ & \langle Pr(p(2,true)) =?, Pr(p(t+1,true)|p(t,false)) = .002, p(2,true), \{t|1\}\rangle, & \\
 & \langle Pr(p(1,false)) =?, Pr(p(t+1,false)|p(t,false)) = .998, p(1,false), \{t|0\}\rangle, & \\
 & \langle Pr(p(0,false)) =?, Pr(p(0,false)) = .6, p(0,false), \{\}\rangle & \}; \\
\quad \{ & \langle Pr(p(2,true)) =?, Pr(p(t+1,true)|p(t,false)) = .002, p(2,true), \{t|1\}\rangle, & \\
 & \langle Pr(p(1,false)) =?, Pr(p(t+1,false)|p(t,true)) = .001, p(1,false), \{t|0\}\rangle, & \\
 & \langle Pr(p(0,true)) =?, Pr(p(0,true)) = .4, p(0,true), \{\}\rangle & \}\}.
\end{array}$$

The ground p-SLD refutation tree, the combined supporting set, and the augmented supporting set of Q are also equal to the above set.

Proposition 3 *Given a program P and an atom G, we can construct the augmented combined supporting set PS of the query $Pr(G) =?$. If the $rank()$ function is well-defined then the rank of each atom in PS can be determined by a simple recursive procedure:* If $Pr(G) = \alpha \in PS$ then $rank(G) = 0$
else $rank(G) = sup\{rank(A)|(Pr(G|\ldots, A, \ldots) = \alpha) \in PS\} + 1$.

The **probability of a ground atom G computed from the program P, $Pr^C_P(G)$,** can be calculated from the augmented combined supporting set PS of the query $Pr(G) =?$ recursively:

If $Pr(G) = \alpha \in PS$ then return $Pr^C_P(G) = \alpha$
else if $\{G_1, \ldots, G_n\}$ is coherent, and let G_i be the atom with highest rank, $Pr^C_P(G_1, \ldots, G_n) = \sum\{\alpha \times Pr^C_P(G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_n, A_1, \ldots, A_m)|(Pr(G_i|A_1, \ldots, A_m) = \alpha) \in PS\}$
else $Pr^C_P(G_1, \ldots, G_n) = 0$.

Example 14 *Continuing the example 11 with the query $Q : Pr(p(2, true)) =?$. $Pr^C_P(p(0, true)) = .4$; $Pr^C_P(p(1, true)) = Pr(p(1, true)|p(0, true)) \times Pr^C_P(p(0, true)) + Pr(p(1, true)|p(0, false)) \times Pr^C_P(p(0, false)) = .999 \times .4 + .002 \times .6 = .4008; \ldots$.*

Theorem 5 *Given a program P with a well-defined $rank()$ and a ground atom G. If $rank(G)$ is finite and CRPB is completely quantified and consistent then (1) the probability of G computed from the program P, $Pr^C_P(G)$, is equal to $Pr_P(G)$, where $Pr_P()$ is the probability function induced by the logic program P. (2) the p-SLD procedure will return the value $Pr^C_P(G)$ which is equal to $Pr_P(G)$.*

The condition that $rank(G)$ be finite can be assured by the acyclicity property of probabilistic logic programs. We have soundness and completeness of p-SLD for acyclic programs.

Theorem 6 *Given a program P with an acyclic probabilistic base and self-contained combining rules. If CRPB is completely quantified and consistent then p-SLD procedure is sound and complete wrt finite rank ground atoms.*

As can be seen in the recursive definition of $Pr^C_P()$, p-SLD can be easily extended to evaluate the probability of a finite conjunction of atoms. In fact, we can evaluate any finite formula of the form $\vee_i \wedge_j A_{ij}$ or $\wedge_i \vee_j A_{ij}$, where the A_{ij} are atoms of finite rank by a simple extension of p-SLD.

9 Local Maximum Entropy: Negation As Failure For Probabilistic Logic Programs

Negation-as-failure is used as a default rule in the SLDNF proof procedure. It allows us to conclude that a ground atom A has the truth value false if all attempts to prove A fail. In probabilistic logic programs, such default rules are desirable both to shorten the programs and to facilitate reasoning on incompletely specified programs. For these reasons we would like to define a probabilistic analogue to negation-as-failure.

Example 15 *Consider the example 3. It is obvious that we should be able to infer $Pr(p(false)) = .6$. Furthermore, we only know that $Pr(q(average)|p(true)) + Pr(q(bad)|p(true)) = .7$ and want to have some default rule to temporarily assign a probability value to each of $Pr(q(bad)|p(true))$ and $Pr(q(average)|p(true))$.*

A popular principle for assigning missing probability values is the *maximum entropy principle.* We propose the *local maximum entropy rule* as a form of negation-as-failure for probabilistic logic programs.

Definition 21 *Given a program P and its corresponding CRPB. Let A be an atom in RAS and $VAL(A) = \{v_1, \ldots, v_n\}$. The* **local maximum entropy rule (LME)** *can be applied to A in the following situation: If $Pr(A|A_1, \ldots, A_k) = \alpha$ is a sentence in CRPB such that the set*

$V = \{val(A')|Pr(A'|A_1, \ldots, A_k) = \alpha \in CRPB$ *and* $obj(A') = obj(A)\}$ *has* $m, 0 < m < n$, *elements and* β*, the sum of all* α *in those sentences, is* ≤ 1 *then augment CRPB with the following set of sentences* $\{Pr(A'|A_1, \ldots, A_k) = (1-\beta)/(n-m)|A' \in Ext(A)$ *and* $val(A') \in \neg V\}$.

Example 16 *Continuing the previous example, the LME rule would assign .6 to* $Pr(p(false))$ *and .35 to* $Pr(q(average)|p(true))$ *and* $Pr(q(bad)|p(true))$.

We incorporate LME into the p-SLD proof procedure and call the new procedure p-SLDLME by generalizing the concept of derivation and other dependent concepts. Details are given in the full paper.

10 Bayesian Networks Construction

10.1 Baysian networks

Bayesian networks [18] are finite directed acyclic graphs. Each node in a Bayesian network represents a random variable, which can be assigned values from a fixed finite set. A link represents the relationship, either causal or relevance, between random variables at both ends. Usually, a link from random variable A to a random variable B says that A causes B. Associated with each node A is a link matrix, which contains the conditional probabilities of random variable A receiving specific values given each combination of values of A's parents. The Bayesian network formalism is an efficient approach to representing probabilistic structures and calculating posterior probabilities of random variables given a set of evidence.

In our query procedure, we will not only find the probability value of, say, a random variable but its posterior probability after observing a set of evidence.

Definition 22 *A set of* **evidence** *E is a set of atoms s.t.* $ground(E)$ *is coherent.*

We do that by first constructing from the program the portion of the Bayesian network related to the query. On the constructed network, we can use available propogation procedures to update the probabilities taking into account the set of evidence.

Notice that, in our framework, an atom A represents the fact that the random variable denoted by $obj(A)$ receiving the value $val(A)$ and $VAL(A)$ is the set of all possible values of that random variable.

Definition 23 *Given a program P and a set of evidence E. A* **complete ground query** *is a query of the form* $Pr(G) =?$*, where G is an atom, the last argument of G is a variable and it is the only variable in G. The meaning of such a query is: find the posterior probability distribution of* $obj(G)$*. If* $VAL(G) = \{v_1, \ldots, v_n\}$ *then the answer to such a query is a vector* $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$*, where* $0 \leq \alpha_i \leq 1, \sum_{i=1}^{n} \alpha_i = 1$ *and* α_i *is the posterior probability of* $obj(G)$ *receiving the value* v_i.

A complete query *is a query of the form* $Pr(G) =?$*, where the last argument of G is a variable and the other arguments may also contain variables. The meaning of such a query is: find all ground instances G' of G such that the complete ground query* $Pr(G') =?$ *has an answer and return those answers.*

```
Q*-PROCEDURE

BEGIN
    { Build the network that supports the evidence }
    NET:= {};
    FOR i:=1 TO number_of_elements_in_E DO
        temp := BUILD-NET(the $i^{th}$ element $E_i$ of E, NET);
    { Extend the network to support the ground instances of the query }
    SUBSS := BUILD-NET ($G$, NET);
    UPDATE(NET,$E$);
    { Output posterior probabilities }
    FOR each $\theta$ in SUBSS output the probability values at node $obj(G\theta)$;
END.
```

Figure 2: Query processing procedure.

10.2 Bayesian Network Construction Procedure

Because of space limitation, we drop the details which can be found in the full paper. In this section, we present a query answering algorithm, Q^*-procedure, for answering complete queries. We only consider self-contained combining rules. This assumption allows us to omit the augmentation step, which was presented in the p-SLD proof procedure.

Assume that we are given a program P, a set of evidence E and a complete query $Pr(G) =$?. Q^*-procedure has the following two main steps: build the supporting Bayesian network for $\{A\} \cup (ground(E) \cap RAS)$ by a backward chaining process similar to Prolog engine and calculate the posterior probability using the set of evidence E and any available procedure [18]. Q^*-procedure is more complex than SLDNF because it needs to collect all relevant sentences before combining rules can be used.

The pseudo code for Q^*-procedure is shown in Figure 2. It simply makes calls to BUILD-NET function, which builds the supporting network of an atom, with atoms in E and G as successive arguments. The FOR loop constructs the supporting network for the set of evidence and the final BUILD-NET call augments the constructed network with supporting network of ground instances of the query. UPDATE() is any probability propogation algorithm on Bayesian networks. BUILD-NET receives as input an atom, whose supporting network needs to be explored. It updates the NET, which might have been partially built. The returning value of the function is the set of substitutions for input atom to get all corresponding ground instances in the resulting network.

10.3 The soundness and completeness of Q^*-procedure

In Q^*-procedure, we do not address the problem of termination. We expect that the techniques for assuring termination of Prolog programs could be applied to Q^*-procedure.

Theorem 7 (Soundness) *Given a program P. If CRPB is completely quantified then Q^*-procedure is sound wrt complete queries.*

Theorem 8 (Soundness and Completeness) *Given a program P with an allowed and acyclic PB. If CRPB is completely quantified then Q^*-procedure is sound and complete wrt complete ground queries and ground finite set of evidence.*

11 Related Work

In a related paper [16] we present a temporal variant of our logic. We describe the application of this framework to representing probabilistic temporal processes and projecting probabilistic plans.

Poole [19] expresses an intention similar to ours: "there has not been a mapping between logical specifications of knowledge and Bayesian network representations ..". He provides such a mapping using probabilistic Horn abduction theory, in which knowledge is represented by Horn clauses and the independence assumption of Bayesian networks is explicitly stated. His work is developed along a different track than ours, however, by concentrating on using the theory for abduction. Our approach has several advantages over Poole's. We do not impose as many constraints on our representation language as he does. Probabilistic dependencies are directly represented in our language, while in Poole's language they are indirectly specified through the use of special predicates in the rules. Our probabilistic independence assumption is more intuitively appealing since it reflects the causality of the domain.

References

[1] K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, pages 335–363, Sept 1991.

[2] F. Bacchus. Using first-order probability logic for the construction of Bayesian networks. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 219–226, July 1993.

[3] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, pages 35–54, 1987. 68.

[4] J.S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 8(4):624–647, 1992.

[5] F.J. Diez. Parameter adjustment in bayes networks. the generalized noisy or-gate. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 99–105, Washington, D.C., July 1993.

[6] M. C. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, (11):91–116, 1988.

[7] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *JACM*, pages 620–650, July 1991.

[8] R.P. Goldman and E. Charniak. A language for construction of belief networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):196–208, March 1993.

[9] P. Haddawy. Generating Bayesian networks from probability logic knowledge bases. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 262–269, Seattle, July 1994.

[10] D. Heckerman and M.P. Wellman. Bayesian networks. *Communications of the ACM*, 38(3):27–30, March 1995.

[11] M. Kifer and V. S. Subramahnian. Theory of generalized annotated logic programs and its applications. *Journal of Logic Programming*, pages 335–367, 12 1992.

[12] J. W. Lloyd. *Foundation of Logic Programming. Second edition.* Springer-Verlag, 1987.

[13] Raymond Ng. Semantics and consistency of empirical databases. In *Proceedings of the 1993 International Conference on Logic Programming*, pages 812–826, 1993.

[14] Raymond Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probability in deductive databases. In *Proceedings of the 1991 International Conference on Logic Programming*, pages 565–580, 1991.

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

[16] L. Ngo, P. Haddawy, and J. Helwig. A theoretical framework for context-sensitive temporal probability model construction with application to plan projection. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 419–426, August 1995.

[17] Liem Ngo and Peter Haddawy. Plan projection as deduction and plan generation as abduction in a context-sensitive temporal probability logic. In *Submitted to AIPS*, 1995.

[18] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, 1988.

[19] D. Poole. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64(1):81–129, November 1993.

[20] S. Srinivas. A generalization of the noisy-or model. In *UAI-93*, pages 208–217, July 1993.

[21] van Emden M. H. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, pages 37–53, 4 1986.

[22] M.P. Wellman, J.S. Breese, and R.P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53, 1992.