A Statistical Learning Method for Logic Programs with Distribution Semantics

Taisuke SATO
Tokyo Institute of Technology
2-12-2 Ookayam Muguro-ku Tokyo Japan 305
email: sato@cs.titech.ac.jp

Abstract 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 possible least models of DB. We then define the semantics of DB with the associated distribution P_F as P_{DB}, and call it distribution semantics.

While the distribution semantics is a straightforward generalization of the traditional least model semantics, it can capture semantics of diverse information processing systems ranging from Bayesian networks to Hidden Markov models to Boltzmann machines in a single framework with mathematical rigor. Thus symbolic computation and statistical modeling are integrated at semantic level.

With this new semantics, we propose a statistical learning schema based on the EM algorithm known in statistics. It enables logic programs to learn from examples, and to adapt to the surrounding environment. We implement the schema for a subclass of logic programs called BS-programs.

1 Introduction

Symbolic computation combined with a probabilistic framework provides a powerful mechanism for a symbolic information system to handle uncertainty and makes the system more flexible and robust. Hidden Markov Models [14] in speech recognition and Bayesian networks [12] in knowledge engineering are classic examples. A similar approach has been proposed in natural language processing as well [4, 5].

Those are systems in application fields that have to deal with raw data from the real world, so the need for coping with uncertainty arises naturally. The same need arises in logical reasoning, for example, when we consider abduction and induction. In abduction, we generate a hypothesis that entails observations, but there are usually multiple hypotheses even for a single observation. Likewise in induction, we are required to discover a "law" by generalizing observations, but there can be many ways of generalization. Whichever case we may take, it is hardly possible to tell, by purely symbolic

reasoning, what the best candidate is. It seems that we draw on, more or less inevitably, probability as a means for measuring plausibility of the candidate. In logic programming, we can see a considerable body of research works that make use of probabilities [7, 8, 9, 13].

The objective of this paper is to provide basic components for a unified symbolic-statistical information processing system in the framework of logic programming. The first one is a semantic basis for probabilistic computation. The second one is a general learning schema for logic programs. The latter is derived by applying a well-known statistical inference method to the former.

Our semantics is called *distribution semantics*, which is defined, roughly, as a distribution over least models. As such it is a generalization of the traditional least model semantics and hence, it is expressive enough to describe Turing machines. In addition, since, for example, it can describe Markov chains [2] precisely, as one least fixed point corresponds to one sample process, any information processing model based on Markov chains is describable. Hidden Markov Model [14] in speech recognition is a typical example. Also connectionist learning models such as Boltzmann machines [1] are describable. Due to space limitations however, issues on operational semantics (how to execute programs with distribution semantics) will not be discussed.

Distribution semantics adds a new dimension to programming; the learning of (parameters of) a distribution. To exploit it, we apply the EM algorithm, which is an iterative method in statistics for computing maximum likelihood estimates with incomplete data[15], to logic programs with distribution semantics to obtain a general learning schema. We then specifically single out a subclass of logic programs (*BS-programs*) that are simple but powerful enough to cover the well-known existing probabilistic models such as Bayesian networks and Hidden Markov Models, and specialize the learning schema to this class. The obtained learning algorithm iteratively adjusts the parameters of an initial distribution so that the behavior of a program matches given examples. Distribution semantics thus bridges a gap between programming and learning.

In section 2, we formally introduce distribution semantics and its properties are described. However, for readability, most proofs are omitted.

In section 3, the EM algorithm is combined with the distribution semantics. In section 4, the class of BS-programs is introduced and a learning algorithm for this class is presented. Section 5 describes an experimental result with the learning algorithm. Section 6 is conclusion referring to related work.

2 Distribution semantics

2.1 Preliminaries

The relationship between logic and probability is quite an old subject and its investigation is inherently of interdisciplinary nature ([3, 6, 7, 9, 11, 13]). One of our purposes here is to show how to assign probabilities to *all* first order formulae containing $\forall$ and $\exists$ over an *infinite* Herbrand universe in such a way that the assignment satisfies Kolmogoroff's axioms for probability and causes no inconsistency. This seems required because, for example, Probabilistic Logic [9, 11], a prevalent formulation in AI for the assignment of probabilities to logical formulae, was not very keen on the problem of consistent assignment of probabilities to all logical formulae over an infinite domain. We follow the Gaifman's approach [3] (with a necessary twist for our purpose).

Let $DB = F \cup R$ be a definite clause program in a first order language with denumerably many variables, function symbols and predicate symbols, where F denotes a set of unit clauses (hereafter referred to as *facts*) and R a set of non-unit clauses (hereafter referred to as *rules*), respectively. We say that DB satisfies the *disjoint condition* if no atom in F unifies with the head of a rule in R. For simplicity, we make following assumptions throughout this paper.

- DB is ground[1].
- DB is denumerably infinite.
- DB satisfies the disjoint condition.

A ground atom A is treated as a random variable taking 1 (when A is true) or 0 (when A is false). Let $A_1, A_2, \ldots$ be an arbitrary enumeration of ground atoms in F and fix the enumeration. An interpretation ω for F, i.e., an assignment of truth values to atoms in F, is identified as an infinite vector $\omega = \langle x_1, x_2, \ldots \rangle$ with the understanding that x_i $(i = 1, 2, \ldots)$ denotes the truth value of the atom A_i.

Write the set of all possible interpretations for F as

$$\Omega_F \stackrel{\text{def}}{=} \prod_{i=1}^{\infty} \{0,1\}_i$$

Let P_F be a completely additive probability measure on the σ algebra $\mathcal{A}_F$[2] of sets in Ω_F. We call P_F a *basic distribution for* F.

[1] In case of a non-ground DB, we reduce it to the set of all possible ground instantiations of clauses in DB.

[2] Ω_F is a Cartesian products of $\{0,1\}$s with discrete topology. So it has the product topology and there exists the smallest σ algebra $\mathcal{A}_F$ including all open sets. By the way, the existence of P_F is not self-evident. We will show how to construct P_F later.

$\omega = \langle x_1, x_2 \rangle$	$F_{1\omega}$	$M_{DB_1}(\omega)$
$\langle 0, 0 \rangle$	$\{\}$	$\{\}$
$\langle 1, 0 \rangle$	$\{A_1\}$	$\{A_1, B_1\}$
$\langle 0, 1 \rangle$	$\{A_2\}$	$\{A_2, B_1, B_2\}$
$\langle 1, 1 \rangle$	$\{A_1, A_2\}$	$\{A_1, A_2, B_1, B_2\}$

Table 1: M_{DB_1}

In view of the fact that P_F defines for each n an n-place distribution function $P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n)$ and P_F is uniquely recoverable from those $P_F^{(n)}$s (as we will see next), we deliberately confuse, for notational convenience, P_F with the corresponding distribution functions. We henceforth write $P_F(A_1 = x_1, A_2 = x_2, \ldots)$ to mean P_F as a probability measure and the corresponding distribution function interchangeably. Also we won't mention the underlying σ algebra $\mathcal{A}_F$ when obvious.

Now each sample $\omega = \langle x_1, x_2, \ldots \rangle \in \Omega_F$ determines a set $F_\omega \subset F$ of true ground atoms. So we can speak of a logic program $F_\omega \cup R$ and its least model $M_{DB}(\omega)$. The crux of distribution semantics lies in the observation that $M_{DB}(\omega)$ decides all truth values of atoms in DB. $M_{DB}(\omega)$ is called *the least model derived from* ω. We show $M_{DB}(\omega)$ for a finite program DB_1.

$$\begin{aligned} DB_1 &= F_1 \cup R_1 \\ F_1 &= \{A_1, A_2\} \\ R_1 &= \{B_1 \leftarrow A_1, B_1 \leftarrow A_2, B_2 \leftarrow A_2\} \end{aligned}$$

We have $\Omega_{F_1} = \{0, 1\}_1 \times \{0, 1\}_2$ and $\omega = \langle x_1, x_2 \rangle \in \Omega_{F_1}$ means A_i takes $x_i (i = 1, 2)$ as its truth value (see Table 1).

2.2 The existence of P_F

Let $DB = F \cup R$ be a definite program, F facts, R rules, and Ω_F the sample space of all possible interpretations for F, respectively, as previously defined.

We first show how to construct a basic distribution P_F for F from the collection of finite distributions. Let $A_1, A_2, \ldots$ be the enumeration of atoms in F previously introduced. Suppose we have a series of finite distributions $P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n)$ $(n = 1, 2, \ldots, x_i \in \{0, 1\}, 1 \leq i \leq n)$ such that

$$\begin{gathered} 0 \leq P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n) \leq 1 \\ \textstyle\sum_{x_1, \ldots, x_n} P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n) = 1 \\ \textstyle\sum_{x_{n+1}} P_F^{(n+1)}(A_1 = x_1, \ldots, A_{n+1} = x_{n+1}) = P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n) \\ \ldots \textit{compatibility condition} \end{gathered}$$

It follows from the compatibility condition that there exists a completely additive probability measure P_F over Ω_F [10] (compactness of Ω_F is used)

satisfying for any n

$$P_F(A_1 = x_1, \ldots, A_n = x_n) = P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n).$$

Ω_F is isomorphic to the set of infinite strings consisting of 0s and 1s, and hence it has the cardinality of real numbers. The shape of P_F depends on how we estimate the likelihood of interpretations. If we assume every interpretation for F is likely to appear equally, P_F will be a uniform distribution. In that case, each $\omega \in \Omega_F$ receives probability 0. If, on the other hand, we stipulate no interpretation except ω_0 is possible for F, P_F will give probability 1 to ω_0 and 0 to others.

2.3 From P_F to P_{DB}

Let $A_1, A_2, \ldots$ be again an enumeration, but of all atoms appearing in DB this time[3]. Form Ω_{DB} as the Cartesian product of denumerably many $\{0,1\}$s. Similarly to Ω_F, Ω_{DB} represents the set of all possible interpretations for ground atoms appearing in DB and $\omega \in \Omega_{DB}$ determines the truth value of every ground atom. We here introduce a notation A^x for an atom A by

$$\begin{aligned} A^x &= A \quad \text{if} \quad x = 1 \\ A^x &= \neg A \quad \text{if} \quad x = 0 \end{aligned}$$

Recall that $M_{DB}(\omega)$ denotes the least model derived from an interpretation $\omega \in \Omega_F$ for F. We now extend P_F to a completely additive probability measure P_{DB} over Ω_{DB} as follows. Define a series of finite distributions $P_{DB}^{(n)}(A_1 = x_1, \ldots, A_n = x_n)$ for $n = 1, 2, \ldots$ by

$$\begin{aligned} [A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}]_F &\stackrel{\text{def}}{=} \{\omega \in \Omega_F \mid M_{DB}(\omega) \models A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}\} \\ P_{DB}^{(n)}(A_1 = x_1, \ldots, A_n = x_n) &\stackrel{\text{def}}{=} P_F([A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}]_F) \end{aligned}$$

$[\cdot]_F$ is P_F-measurable. By definition $P_{DB}^{(m)}$ satisfies the compatibility condition:

$$\sum_{x_{n+1}} P_{DB}^{(n+1)}(A_1 = x_1, \ldots, A_{n+1} = x_{n+1}) = P_{DB}^{(n)}(A_1 = x_1, \ldots, A_n = x_n).$$

It follows that there exists a completely additive measure P_{DB} over Ω_{DB}, and P_{DB} becomes an extension of P_F. *We define the denotation of a logic program $DB = F \cup R$ with the associated distribution P_F as P_{DB}.* Put differently, a program denotes a distribution in our semantics.

We are now in a position to assign probabilities to arbitrary formulae. Let G be an arbitrary sentence[4] whose predicates are among DB. Introduce $[G] \subset \Omega_{DB}$ by

$$[G] \stackrel{\text{def}}{=} \{\omega \in \Omega_{DB} \mid \omega \models G\}$$

[3] Note that this enumeration enumerates atoms in F as well.

[4] A sentence is a formula without free variables.

Then the probability of G is defined as $P_{DB}([G])$. Intuitively, $P_{DB}([G])$ represents the probability mass assigned to the set of interpretations (possible worlds) satisfying G.

Thanks to the complete additivity, we enjoy kind of continuity about quantification without special assumptions:

$$\begin{aligned} \lim_{n\to\infty} P_{DB}([G(t_1) \wedge \ldots \wedge G(t_n)]) &= P_{DB}([\forall x G(x)]) \\ \lim_{n\to\infty} P_{DB}([G(t_1) \vee \ldots \vee G(t_n)]) &= P_{DB}([\exists x G(x)]) \end{aligned}$$

where $t_1, t_2, \ldots$ is an enumeration of ground terms. We can also verify that $comp(R)$, the iff form of rule set, satisfies $P_{DB}(comp(R)) = 1$ regardless of the distribution P_F.

2.4 Properties of P_{DB}

Write a program DB as

$$\begin{aligned} DB &= F \cup R \\ F &= \{A_1, A_2, \ldots\} \\ R &= \{B_1 \leftarrow W_1, B_2 \leftarrow W_2, \ldots\} \\ head(R) &= \{B_1, B_2, \ldots\} \end{aligned}$$

A *support set* for an atom $B \in head(R)$ is a finite subset S of F such that $S \cup R \vdash B$. A *minimal support set* for B_i is a support set minimal w.r.t. set inclusion ordering. When there are only a finite number of minimal support sets for every $B \in head(R)$, we say that DB satisfies the *finite support condition*. The violation of this condition means there will be an atom $B \in head(R)$ for which we cannot be sure, within finite amount of time, if there exists a hypothesis set $S \subset F$ such that $S \cup R \vdash B$. Fortunately, usual programs seem to satisfy the finite support condition. Put

$$fix(DB) \stackrel{\text{def}}{=} \{M_{DB}(\omega) \mid \omega \in \Omega_F\}$$

$fix(DB) \subseteq \Omega_{DB}$ denotes the collection of least models derived from possible interpretation for F. For $\omega = \langle x_1, x_2, \ldots\rangle \in \Omega_{DB}$, we use $\omega|_F$ to stand for a sub-vector of ω whose components correspond to the truth values of atoms in F. By construction $\omega|_F$ belongs in Ω_F, and $M_{DB}(\omega|_F)$ belongs in Ω_{DB}. We however do not know beforehand if $M_{DB}(\omega|_F)$ coincides with the original ω.

Lemma 2.1 *Suppose DB satisfies the finite support condition. For $\omega = \langle x_1, x_2 \ldots\rangle$,*

$$\omega = M_{DB}(\omega|_F) \Leftrightarrow \forall n [A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}]_F \neq \emptyset$$

Proof: Since $\Rightarrow$ is rather obvious, we prove $\Leftarrow$. Suppose $\omega \neq M_{DB}(\omega|_F)$. Then $M_{DB}(\omega|_F) \models \neg A_k^{x_k}$ holds for some k.

There is a finite set $\{A_1, \ldots, A_n\}$ which contains all minimal support sets for A_k. We may assume without losing generality $n \geq k$. It follows from $[A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}]_F \neq \emptyset$ that there is $\omega' \in \Omega_F$ such that $M_{DB}(\omega') \models A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}$. In particular, we have $M_{DB}(\omega') \models A_k^{x_k}$.

On the other hand, from $M_{DB}(\omega') \models A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}$ and since no atom in F appears at the head of a rule, ω and ω' must agree on the truth value of any atom in $\{A_1, \ldots, A_n\} \cap F$. However this contradicts $M_{DB}(\omega|_F) \models \neg A_k^{x_k}$ and $M_{DB}(\omega') \models A_k^{x_k}$. Therefore we must have $\omega = M_{DB}(\omega|_F)$. Q.E.D.

Define a set $E_{y_1,\ldots,y_n} \subset \Omega_{DB}$ for a finite vector $\langle y_1, \ldots, y_n \rangle$ $(y_i \in \{0,1\}, 1 \leq i \leq n)$ by

$$E_{y_1,\ldots,y_n} \stackrel{\text{def}}{=} \{\langle y_1, \ldots, y_n, *, *, \ldots \rangle \in \Omega_{DB} \mid [A_1^{y_1} \wedge \ldots \wedge A_n^{y_n}]_F = \emptyset\}$$

where $*$ is a don't care symbol. Either $E_{y_1,\ldots,y_n} = \emptyset$ or $P_{DB}(E_{y_1,\ldots,y_n}) = 0$ holds.

Theorem 2.1 *If DB satisfies the finite support condition, $fix(DB)$ is P_{DB}-measurable. Also $P_{DB}(fix(DB)) = 1$.*

Proof: From Lemma 2.1, we see, for $\omega = \langle x_1, x_2, \ldots \rangle \in \Omega_{DB}$,

$$\begin{array}{rcl} \omega \neq M_{DB}(\omega|_F) & \Leftrightarrow & \exists n [A_1^{x_1} \wedge \ldots \wedge A_n^{x_n}]_F = \emptyset \\ & \Leftrightarrow & \omega \in \bigcup_{n=1}^{\infty} \bigcup_{y_1,\ldots,y_n} E_{y_1,\ldots,y_n} \end{array}$$

So $\{\omega \in \Omega_{DB} \mid \omega \neq M_{DB}(\omega|_F)\}$ is a null set (note $P_{DB}(E_{y_1,\ldots,y_m}) = 0$). Since we can prove

$$fix(DB) = \{\omega \in \Omega_{DB} \mid \omega = M_{DB}(\omega|_F)\}$$

we conclude $fix(DB)$ is P_{DB}-measurable and $P_{DB}(fix(DB)) = 1$. Q.E.D.

Theorem 2.1 says that under a certain condition (which we believe most programs satisfy), probability mass is distributed only over the least models of the form $M_{DB}(\omega)(\omega \in \Omega_F)$.

Theorem 2.2 *If P_F gives probability 1 to $\{\omega_0\} \subset \Omega_F$, P_{DB} gives probability 1 to $\{M_{DB}(\omega_0)\} \subset \Omega_{DB}$.*

Proof: easy and omitted.

Theorem 2.2 allows us to regard distribution semantics as a generalization of the least model semantics because we may think of a usual definite clause program $DB = F \cup R$ as one in which F always appears with probability 1.

Distribution semantics is highly expressive. Although we do not prove here, it can describe from Turing machines (recursive functions) to Bayesian

networks to Markov chains.

We show Proposition 2.1 which is convenient for the calculation of P_{DB} (proof is easy and omitted). $\{A_1, \ldots, A_n\} \subset F$ is said to *finitely determine* B if $\{A_1, \ldots, A_n\}$ includes all minimal support sets for B. When $\{A_1, \ldots, A_n\}$ finitely determines every atom in $\{B_1, \ldots, B_k\}$, it is said to finitely determine $\{B_1, \ldots, B_k\}$. The finite support condition is restated as any $B \in head(R)$ is finitely determined.

Lemma 2.2 *If* $\{A_1, \ldots, A_n\} \subset F$ *finitely determines* $\{B_1, \ldots, B_k\}$,

$$\forall x_1, \ldots, x_n \exists! \, y_1, \ldots, y_k$$
$$\forall \omega \in \Omega_F (\omega \models A_1^{x_1} \wedge \ldots \wedge A_n^{x_n} \rightarrow M_{DB}(\omega) \models B_1^{y_1} \wedge \ldots \wedge B_k^{y_k})$$

Consequently, if $\{A_1, \ldots, A_n\}$ finitely determines $\{B_1, \ldots, B_k\}$, the truth values of $\{B_1, \ldots, B_k\}$ are uniquely determined by those of $\{A_1, \ldots, A_n\}$. We introduce a function $\varphi_{DB}(x_1, \ldots, x_n)$ to designate this functional relationship.

$$\varphi_{DB}(x_1, \ldots, x_n) = \langle y_1, \ldots, y_k \rangle \quad \text{iff}$$
$$\forall \omega \in \Omega_F (\omega \models A_1^{x_1} \wedge \ldots \wedge A_n^{x_n} \rightarrow M_{DB}(\omega) \models B_1^{y_1} \wedge \ldots \wedge B_k^{y_k})$$

Proposition 2.1 *Suppose* $\{A_1, \ldots, A_n\} \subset F$ *finitely determines* $\{B_1, \ldots, B_k\}$. *Then*

$$P_{DB}(A_1 = x_1, \ldots, A_n = x_n, B_1 = y_1, \ldots, B_k = y_k)$$
$$= \begin{cases} P_F(A_1 = x_1, \ldots, A_n = x_n) & \text{if} \quad \varphi_{DB}(x_1, \ldots, x_n) = \langle y_1, \ldots, y_k \rangle \\ 0 \quad o.w. \end{cases}$$

$$P_{DB}(B_1 = y_1, \ldots, B_k = y_k)$$
$$= \sum_{\varphi_{DB}(x_1, \ldots, x_n) = \langle y_1, \ldots, y_k \rangle} P_F(A_1 = x_1, \ldots, A_n = x_n)$$

2.5 Program examples

To get a feel for distribution semantics, we show two program examples. First we take up the finite program DB_1 again and give a distribution P_{F_1} for $F_1 = \{A_1, A_2\}$ which is shown in Table 2 where $x_i (i = 1, 2)$ denotes the truth value of A_i. P_{DB_1} is calculated from P_{F_1} using Proposition 2.1. $\omega = \langle x_1, x_2, y_1, y_2 \rangle \in \Omega_{DB_1}$ indicates that $x_i (i = 1, 2)$ is the value of A_i and $y_j (j = 1, 2)$ is the value of B_j, respectively.

Next example DB_2 describes a Markov chain with infinite states[5]. The

[5] Prolog notation used for conjunction.

$\omega = \langle x_1, x_2 \rangle$	$P_{F_1}(x_1, x_2)$
$\langle 0, 0 \rangle$	0.2
$\langle 1, 0 \rangle$	0.3
$\langle 0, 1 \rangle$	0.4
$\langle 1, 1 \rangle$	0.1
others	0.0

$\omega = \langle x_1, x_2, y_1, y_2 \rangle$	$P_{DB_1}(x_1, x_2, y_1, y_2)$
$\langle 1, 0, 0, 0 \rangle$	0.2
$\langle 1, 0, 1, 0 \rangle$	0.3
$\langle 0, 1, 1, 1 \rangle$	0.4
$\langle 1, 1, 1, 1 \rangle$	0.1
others	0.0

Table 2: P_{F_1} & P_{DB_1} for DB_1

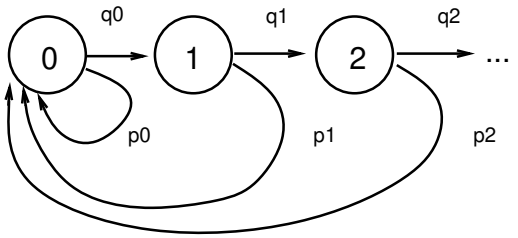

Figure 1: A renewal sequence

chain is a renewal sequence (see Figure 1).

$$
\begin{aligned}
DB_2 &= F_2 \cup R_2 \\
R_2 &= \begin{cases} s(0,0) \\ s(0,T+1) \leftarrow s(K,T), tr(K,T,1) \\ s(K+1,T+1) \leftarrow s(K,T), tr(K,T,0) \end{cases} \\
F_2 &= \begin{cases} tr(0,T,1) \quad tr(0,T,0) \\ tr(1,T,1) \quad tr(1,T,0) \\ \ldots \end{cases} \\
& \quad disjoint([tr(0,T,1) : p_0, tr(0,T,0) : q_0]) \\
& \quad disjoint([tr(1,T,1) : p_1, tr(1,T,0) : q_1]) \\
& \quad \ldots
\end{aligned}
$$

$s(K,T)$ describes the life of a machine M discretely. It says that M is in state K at time T. M's life starts from state 0 at time 0. M breaks down from time T to time $T+1$ with probability p_K and must be replaced with a new one. Otherwise, M survives (with probability $q_K = 1 - p_K$) and will be in state $K+1$ at time $T+1$.

Here $disjoint([tr(k,T,1) : p_k,\ tr(k,T,0) : q_k])$ means, for any instantiation of T, either one of $tr(k,T,1)$ or $tr(k,T,0)$ is true (the probability of $tr(k,T,1)$ being true is p_k) but they never become true at the same time. This disjoint notation is borrowed from [13]. We assume that $tr(k,\cdot,\cdot)$ and

$tr(k', \cdot, \cdot)$ are independent if $k \neq k'$. We also assume that $tr(k, t, x)$ and $tr(k, t', x)$ are independent and identically distributed if $t \neq t'$.

Then the distribution P_{F_2} for F_2 is defined in an obvious way. We assume that for any k, neither p_k nor q_k's takes values 0 or 1, which means M can go from any state to any state.

Apparently there is one-to-one correspondence between the least fixed model of ${F_2}' \cup R_2$ where ${F_2}' = \{tr(0, 0, x_0), tr(k_1, 1, x_1), tr(k_2, 2, x_2), \ldots\}$ is a sample drawn from P_{F_2} and a sample sequence that starts from state 0 at time t and traces states $0 \to k_1 \to k_2 \ldots$

Since DB_2 satisfies the finite support condition, the probability measure P_{DB_2} is defined over the set of all sample sequences. We know by calculation

$$\begin{array}{rcl} P_{DB_2}(s(k,t)) & = & \text{prob. of M being in state } k \text{ at time } t \\ P_{DB_2}(\exists t S(k,t)) & = & \text{prob. of M passing state } k \text{ sometime } =1 \\ \lim_{t\to\infty} P_{DB_2}(s(k,t)) & = & \text{prob. of M being in sate } k \text{ after } \infty \text{ time} \\ & = & \left\{ \begin{array}{l} \Pi_0 = \frac{1}{EX} \\ \Pi_k = \frac{q_0 q_1 \cdots q_k}{EX} (k > 0) \end{array} \right. \\ & & \text{where } EX \text{ is mean recurrence time.} \end{array}$$

3 EM learning

3.1 Learning a basic distribution

We have seen in the previous section that when a basic distribution P_F is given to facts F of a program $DB = F \cup R$ where $R = \{B_1 \leftarrow W_1, B_2 \leftarrow W_2, \ldots\}$, a distribution $P_{DB}(B_1 = y_1, B_2 = y_2, \ldots)$ is induced for $head(R) = \{B_1, B_2, \ldots\}$. We look at this distribution dependency upside down.

Suppose we have observed truth values of some atoms $B_1, \ldots, B_k$ repetitively and obtained an empirical distribution $P_{obs}(B_1 = y_1, \ldots, B_k = y_k)$. To infer a mechanism working behind this distribution, we write a logic program $DB = F \cup R$ such that $\{B_1, \ldots, B_k\} \subset head(R)$. We then set an initial basic distribution P_F to F and try to make $P_{DB}(B_1 = y_1, \ldots, B_k = y_k)$ as similar to $P_{obs}(B_1 = y_1, \ldots, B_k = y_k)$ as possible by adjusting P_F.

Or if we adopt MLE (Maximum Likelihood Estimation), we adjust, given the observations $\langle B_1 = y_1, \ldots, B_k = y_k \rangle$ where $y_i = 0, 1 (1 \leq i \leq k)$, the parameter θ of a parameterized distribution $P_F(\theta)$ [6] so that $P_{DB}(B_1 = y_1, \ldots, B_k = y_k \mid \theta)$ attains the optimum.

This is an act of learning, and when the learning succeeds, we would obtain a logical-statistical model of (part of) the real world described by DB with the distribution P_F. Since MLE is easier to implement, we focus on learning using MLE.

[6] Parameters means numbers that specify a distribution such as mean μ and variance σ^2 in a normal distribution $(2\pi\sigma^2)^{-1/2} exp[-\frac{1}{2}(\frac{x-\mu}{\sigma})^2]$

There is however a fundamental stumbling block. That is, we cannot simply apply MLE to P_{DB} because θ does not govern P_{DB} directly. In our framework, the truth values of $B_1, \ldots, B_k$ are only indirectly related to $P_F(\theta)$ through complicated logical interaction among rules. Nonetheless we can circumvent the obstacle by appealing to the EM algorithm.

3.2 A Learning schema

The EM algorithm is an iterative method used in statistics to compute maximum likelihood estimates with incomplete data [15]. We briefly explain it for the sake of self-containedness. Suppose $f(x, y \mid \theta)$ is a distribution function parameterized with θ. Also suppose we could not observe a "complete data" $\langle x, y \rangle$ but only observed y, part of the complete data, and x is missing for some reason. The EM algorithm is used to perform MLE in this kind of "missing data" situation. It estimates both missing data x and parameter θ by going back and forth between them through iteration [15].

Returning to our case, we notice that there is a close analogy. We have "incomplete observations" $\langle B_1 = y_1, \ldots, B_k = y_k \rangle$ which should be supplemented by "missing observations" $\langle A_1 = x_1, A_2 = x_2, \ldots \rangle$, and we have to estimate parameters $\theta_1, \theta_2, \ldots$ (there may be infinitely many statistical parameters) of $P_F(\theta_1, \theta_2, \ldots)$ lurking in the distribution $P_{DB}(A_1 = x_1, A_2 = x_2, \ldots, B_1 = y_1, \ldots, B_k = y_k)$. So the EM algorithm applies, if we can somehow keep the number of the A_i's and θ_j's finite. We therefore assume, in light of Proposition 2.1, that DB satisfies the finite support condition.

Then, there is a finite set $\langle A_1, \ldots, A_n \rangle$ whose value $\langle x_1, \ldots, x_n \rangle$ determines the truth value $\langle y_1, \ldots, y_k \rangle$ of $\langle B_1, \ldots, B_k \rangle$ (as already used here, we use vectors and sets interchangeably when no confusion arises). Hence, we have only to estimate those parameters that govern the distribution of $\langle A_1, \ldots, A_n \rangle$.

Put $\vec{A} = \langle A_1, \ldots, A_n \rangle$, $\vec{x} = \langle x_1, \ldots, x_n \rangle$, $\vec{B} = \langle B_1, \ldots, B_k \rangle$ and $\vec{y} = \langle y_1, \ldots, y_k \rangle$, and let $\vec{A} = \vec{x}$ stand for $\langle A_1 = x_1, \ldots, A_n = x_n \rangle$, and $\vec{B} = \vec{y}$ for $\langle B_1 = y_1, \ldots, B_k = y_k \rangle$, respectively.

Suppose $\vec{A} \subset F$ finitely determines $\vec{B} \subset head(R)$ in $DB = F \cup R$. Also suppose the distribution of $\vec{A}$ is parameterized by some $\vec{\theta} = \langle \theta_1, \ldots, \theta_h \rangle$ and write P_F as $P_F(\vec{A} = \vec{x} \mid \vec{\theta})$. Under this setting, we can derive an EM learning schema for the observation $\vec{B} = \vec{y}$ from DB by applying the EM algorithm to $P_{DB}(\vec{A} = \vec{x}, \vec{B} = \vec{y} \mid \vec{\theta})$. For a shorter description, we abbreviate $P_{DB}(\vec{A} = \vec{x}, \vec{B} = \vec{y} \mid \vec{\theta})$ to $P_{DB}(\vec{x}, \vec{y} \mid \vec{\theta})$ e.t.c.

Introduce a function $Q(\vec{\theta'}, \vec{\theta})$ by

$$Q(\vec{\theta'}, \vec{\theta}) \stackrel{\text{def}}{=} \sum_{\vec{x}: P_{DB}(\vec{x} \mid \vec{y}, \vec{\theta'}) > 0} P_{DB}(\vec{x} \mid \vec{y}, \vec{\theta'}) \ln P_F(\vec{x} \mid \vec{\theta})$$

In the EM learning schema illustrated in Figure 2, every time $\vec{\theta}$ is renewed, the likelihood $P_{DB}(\vec{y} \mid \vec{\theta})$ increases (≤ 1) [15]. Although the EM

Step 1. Start from $\vec{\theta}_0$ such that $P_{DB}(\vec{y} \mid \vec{\theta}_0) > 0$

Step 2. Suppose $\vec{\theta}'$ has been computed.
Find $\vec{\theta}$ such that $Q(\vec{\theta}', \vec{\theta}) > Q(\vec{\theta}', \vec{\theta}')$

Step 3. Repeat **Step 2** until $Q(\vec{\theta}', \vec{\theta}')$ saturates.

Figure 2: An EM learning schema for $\vec{B} = \vec{y}$

algorithm only guarantees to find a stationary point of $P_{DB}(\vec{y} \mid \vec{\theta})$ (does not necessarily find the global optimum), it is easy to implement and has been used extensibly in speech recognition based on Hidden Markov Models [14].

4 A learning algorithm for BS-programs

Since our EM learning schema is still relative to a distribution P_F (P_F determines P_{DB}), we need to instantiate it to arrive at a concrete learning algorithm. First we introduce *BS-programs* that have distributions of the simplest type.

4.1 BS-programs

We say $DB = F \cup R$ is a BS-program if F and the associated basic distribution P_F satisfy the following conditions.

- An atom in F takes the form $bs(i, n, 1)$ or $bs(i, n, 0)$. They are random variables. We call $bs(i, \cdot, \cdot)$ a *bs*-atom, i a group identifier.
- $disjoint([bs(i, n, 1) : \theta_i, bs(i, n, 0) : 1 - \theta_i])$
(this is already explained in Section 2, or see [13]) θ_i is called a *bs-parameter for* $bs(i, \cdot, \cdot)$.
- If $n \neq n'$, $bs(i, n, x)$ and $bs(i, n', x)$ $(x = 0, 1)$ are independent and identically distributed.
- If $i \neq i'$, $bs(i, \cdot, \cdot)$ and $bs(i', \cdot, \cdot)$ are independent.

A BS-program contains (infinitely many) *bs*-atoms. Each $bs(i, n, x)$ behaves as if x were a random variable taking 1 (resp. 0) with probability θ_i (resp. $1 - \theta_i$). Or more intuitively, $bs(i, n, x)$ is considered as a probabilistic switch that has binary states $\{0, 1\}$. Every time we ask it, it shows either on $(x = 1)$ or off $(x = 0)$ with probability θ for $x = 1$.

We have already seen a BS-program. DB_2 in Section 2 is a BS-program. It is practically important that we can write BS-programs in Prolog which are "operationally correct" in terms of distribution semantics. Figure 3 is

```
bernoulli(N,[R|Y]):-
    N>0,
    bs(coin,N,X),
    ( X=1, R=head ; X=0, R=tail ),
    N1 is N-1,
    bernoulli(N1,Y).

bernoulli(0,[]]):-true.
```

Figure 3: Bernoulli program

an example of BS-program written in Prolog that describes Bernoulli trials. `bs(coin,N,X)` is a probabilistic predicate that represents coin tossing. It says that the outcome of `N`-th tossing is `X`. Given `N`, `bernoulli(N,Y)` returns a list `Y` with length `N` consisting of `{head,tail}`.

Now we return to the problem of specifying P_F. Since it holds that $bs(i,n,0) \leftrightarrow \neg bs(i,n,1)$, it suffices to define a finite joint distribution $P_F(A_1 = x_1, \ldots, A_i = x_i)$ for $i = 1, 2, \ldots$ where A_i is a bs-atom of the form $bs(i,\cdot,1)$.

For an equation $\vec{A} = \vec{x}$ abbreviating $A_1 = x_1, \ldots, A_n = x_n$, $|\vec{A} \stackrel{i}{=} \vec{x}|_1$ denotes the number of equations in $\vec{A} = \vec{x}$ that take the form $bs(i,\cdot,1) = 1$, and $|\vec{A} \stackrel{i}{=} \vec{x}|_0$ denotes the number of equations in $\vec{A} = \vec{x}$ that take the form $bs(i,\cdot,1) = 0$, respectively. Also $G_{id}(\vec{A})$ is used to denote the set of group identifiers appearing in $\vec{A}$. In the case of the Bernoulli program, for an equation

$$\vec{A} = \langle bs(coin,1,1), bs(coin,2,1), bs(coin,3,1)\rangle = \langle 1,1,0\rangle$$

we have $G_{id}(\vec{A}) = \{coin\}$, $|\vec{A} \stackrel{\text{coin}}{=} \vec{x}|_1 = 2$ and $|\vec{A} \stackrel{\text{coin}}{=} \vec{x}|_0 = 1$.

Let $\vec{A}$ be a vector of bs-atoms of the form $bs(i,\cdot,1)$ and θ_i the probability of $bs(i,\cdot,1)$ being true. Put $G_{id}(\vec{A}) = \{i_1, \ldots, i_l\}$ and $\vec{\theta} = \langle \theta_{i_1}, \ldots, \theta_{i_l}\rangle$. P_F is then given by

$$P_F(\vec{A} = \vec{x} \mid \vec{\theta}) = \prod_{i \in G_{id}(\vec{A})} \theta_i^{|\vec{A} \stackrel{i}{=} \vec{x}|_1} \cdot (1-\theta_i)^{|\vec{A} \stackrel{i}{=} \vec{x}|_0}$$

4.2 A learning algorithm for BS-programs

Suppose a program $DB = F \cup R$ is a BS-program and a basic distribution P_F for F is given as above. We assume DB satisfies the finite support condition. Since DB satisfies the finite support condition, $\Sigma_{DB}(\vec{B})(\vec{B} \subset head(R))$ defined by

$\Sigma_{DB}(\vec{B}) \stackrel{\text{def}}{=}$

$\{A \in F \mid A$ belongs in a minimal support set for some $B_j (1 \leq j \leq k)\}$

becomes a finite set. Write $\Sigma_{DB}(\vec{B})$ as a vector and put $\Sigma_{DB}(\vec{B}) = \vec{A}$. Since $\vec{A}$ finitely determines $\vec{B}$, we may introduce $\varphi_{DB}(\vec{x}) = \vec{y}$ in Section 2 where $\vec{x}$ and $\vec{y}$ are the values of $\vec{A}$ and $\vec{B}$ respectively. Proposition 2.1 is then rewritten as

$$P_{DB}(\vec{y} \mid \vec{\theta}) = \sum_{\varphi_{DB}(\vec{x})=\vec{y}} P_F(\vec{x} \mid \vec{\theta})$$

Here $\vec{\theta}$ denotes the set of bs-parameters for $\vec{A}$.

Now let $\langle \vec{B}_1 = \vec{y}_1, \ldots, \vec{B}_M = \vec{y}_M \rangle$ be the result of M *independent* observations. Each $\vec{B}_m \subset head(R)$ $(1 \leq m \leq M)$ represents a set of atoms appearing in the heads of rules which we observed at m-th time. We can derive a learning algorithm to perform MLE with $\langle \vec{B}_1 = \vec{y}_1, \ldots, \vec{B}_M = \vec{y}_M \rangle$ by specializing the EM learning schema in Section 3 to BS-programs. Put

$$\begin{aligned} \vec{A}_m &\stackrel{\text{def}}{=} \Sigma_{DB}(\vec{B}_m)\ (1 \leq m \leq M) \\ \{i_1, \ldots, i_l\} &\stackrel{\text{def}}{=} G_{id}(\vec{A}_1) \cup \ldots \cup G_{id}(\vec{A}_M) \\ \vec{\theta} &\stackrel{\text{def}}{=} \{\theta_{i_1}, \ldots, \theta_{i_l}\} \end{aligned}$$

$\vec{\theta}$ is the set of bs-parameters concerning $\langle \vec{B}_1 = \vec{y}_1, \ldots, \vec{B}_M = \vec{y}_M \rangle$. The derived algorithm is described in Figure 4 where $P_F(\vec{A}_m = \vec{x}_m \mid \vec{\theta})$ and $P_{DB}(\vec{B}_m = \vec{y}_m \mid \vec{\theta})$ are abbreviated respectively to $P_F(\vec{x}_m \mid \vec{\theta})$ and to $P_{DB}(\vec{y}_m \mid \vec{\theta})$. It is used to estimate bs-parameters $\vec{\theta}$ by performing MLE with the results of M independent observations $\langle \vec{B}_1 = \vec{y}_1, \ldots, \vec{B}_M = \vec{y}_M \rangle$.

5 A learning experiment

To confirm that our EM learning algorithm for BS-programs actually works, we have built, using Prolog, a small experiment system and have conducted experiments with a program DB_3[7] expressing a Hidden Markov Model depicted in Figure 6.

The Hidden Markov Model in Figure 6 starts from state $S1$ and on each transition between the states, it outputs an alphabet a or b according to the specified probability. For example, it goes from $S1$ to $S2$ with probability 0.7 (= that of $bs(0,T,0)$) and outputs a or b with probability 0.5. The finial state is $S3$. In the corresponding program DB_3, predicate $S1(L,T)$ for example means the system has output list L until time T.

[7] Prolog notation is used for list, conjunction and disjunction.

Step 1.
Choose any $\vec{\theta}^{(0)}$ such that
$P_{DB}(\vec{y}_m \mid \vec{\theta}^{(0)}) > 0$ for $\forall m(1 \leq m \leq M)$

Step 2.
Until $\prod_{m=1}^{m=M} P_{DB}(\vec{y}_m \mid \vec{\theta})$ saturates
 Repeat
  Renew $\vec{\theta}^{(n)}$ to $\vec{\theta}^{(n+1)}$ by
   For $i \in \{i_1, \ldots, i_l\}$, renew $\theta_i^{(n)}$ to $\theta_i^{(n+1)}$
   where
   $\theta_i^{(n+1)} = \frac{ON_i(\vec{\theta}^{(n)})}{ON_i(\vec{\theta}^{(n)}) + OFF_i(\vec{\theta}^{(n)})}$

$ON_i(\vec{\theta}) \stackrel{\text{def}}{=} \sum_{m=1}^{m=M} \sum_{\varphi_{DB}(\vec{x}_m)=\vec{y}_m} \frac{P_F(\vec{x}_m|\vec{\theta})|\vec{A}_m \stackrel{i}{=} \vec{x}_m|_1}{P_{DB}(\vec{y}_m|\vec{\theta})}$

$OFF_i(\vec{\theta}) \stackrel{\text{def}}{=} \sum_{m=1}^{m=M} \sum_{\varphi_{DB}(\vec{x}_m)=\vec{y}_m} \frac{P_F(\vec{x}_m|\vec{\theta})|\vec{A}_m \stackrel{i}{=} \vec{x}_m|_0}{P_{DB}(\vec{y}_m|\vec{\theta})}$

Figure 4: A learning algorithm for BS-programs

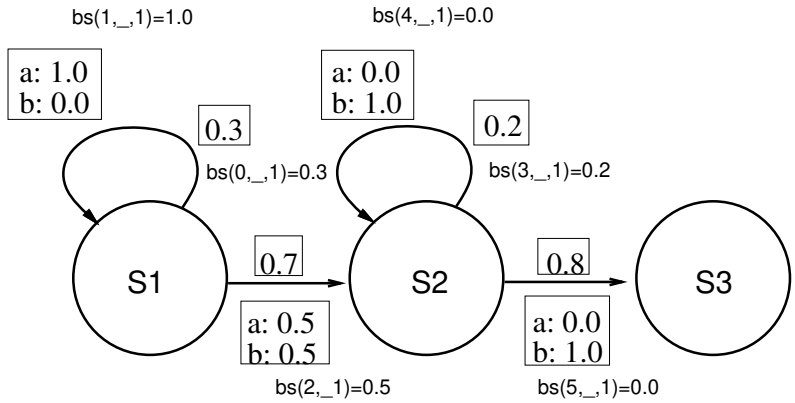

Figure 5: A transition diagram

bs-atom	original value	estimated value
$bs(0,T,1)$	0.3	0.348045
$bs(1,T,1)$	1.0	1.0
$bs(2,T,1)$	0.5	0.496143
$bs(3,T,1)$	0.2	0.15693
$bs(4,T,1)$	0.0	4.5499e-06
$bs(5,T,1)$	0.0	0.0

Table 3: The result of an experiment

$$DB_3 = \left\{ \begin{array}{l} s1([],0) \\ s1([W|X],T+1) \leftarrow \\ \quad s1(X,T), (bs(1,T,1), W=a; bs(1,T,0), W=b), bs(0,T,1) \\ s2([W|X],T+1) \leftarrow \\ \quad s1(X,T), (bs(2,T,1), W=a; bs(2,T,0), W=b), bs(0,T,0) \\ s2([W|X],T+1) \leftarrow \\ \quad s2(X,T), (bs(4,T,1), W=a; bs(4,T,0), W=b), bs(3,T,1) \\ s3([W|X],T+1) \leftarrow \\ \quad s2(X,T), (bs(5,T,1), W=a; bs(5,T,0), W=b), bs(3,T,0) \end{array} \right.$$

In the experiment, we first set the probabilities of bs atoms according to Table 3 (original value) and got 100 samples from the program. Then using this data set, the probabilities of bs atoms were estimated by the EM learning algorithm. We repeated this experiment several times and a typical result is shown in Table 3[8]. Estimated values seem rather close to the original values though we have not done any statistical testing.

6 Conclusion

We have proposed distribution semantics for probabilistic logic programs and have presented an associated learning schema based on the EM algorithm. They offer a way to the integration of so far unrelated areas such as symbol processing and statistical modeling, or programming and learning, at semantic level in a unified framework.

Distribution semantics does not deal with a single least model. It instead considers a distribution over the set of all possible least models for a program $DB = F \cup R$ which are generated from the rule set R and a sampling F' drawn form a distribution P_F given to the facts F. It includes the usual least model semantics as a special case.

[8] This case took 13 iterations to converge.

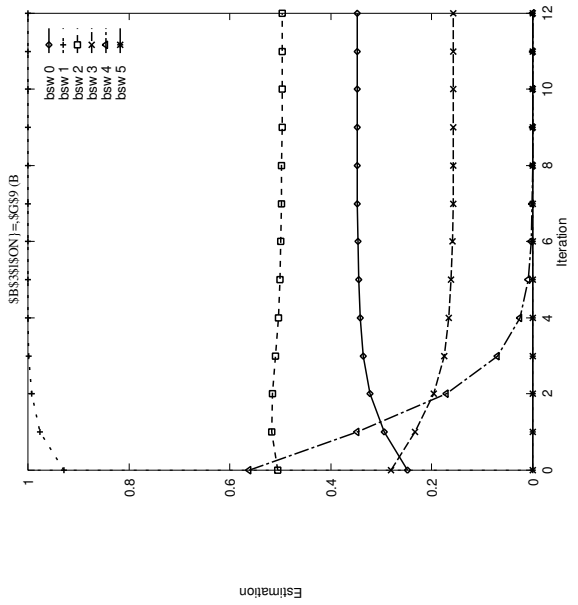

Figure 6: Convergens of parameters

Combining distribution semantics with the EM algorithm, we have derived a distribution learning schema and specialized it to the class of BS-programs, which still can express Markov chains as well as Bayesian networks. An experimental result was shown for a Hidden Markov Model learning program.

There remains much to be done. We need more experiments with BS-programs. If they turn out to be too simple to describe real data, more powerful distributions should be considered. Especially, Boltzmann distributions and Boltzmann machine learning are promising candidates.

We have assigned a distribution to facts but not to rules. This treatment might appear too restrictive, but not really so, because, if we have a meta-interpreter, rules are representable as unit clauses. Learning a distribution over rules through meta-programming should be pursued.

We state related work. While our approach equally concerns each of logic, probability and learning, we have not seen many papers of similar character. For example, there are a lot of research works on abduction but very few combine them with probability, let alone learning.

Poole, however, recently proposed a general framework for Probabilistic Horn abduction and has shown Bayesian networks are representable in his framework[13]. Although his formulation is elegant and powerful, it leaves something to be desired. The first is that his semantics excludes usual logic

programs and it can not be a generalization of the least model semantics[9].

The second is that probabilities are considered only for finite cases and there is no "joint distribution of denumearbly many random variables." As a result, neither can we have the complete additivity of a probability measure, nor we can express by his semantics stochastic processes such as Markov chains. Both problems do not exist in our semantics.

Also in the framework of Logic Programming, Ng and Subrahmanian proposed Probabilistic Logic Programming [9]. They first assign "probability ranges" to atoms in the program (the notion of a distribution seems secondary to their approach) and then check, using linear programming technique, if probabilities satisfying those ranges actually exist or not. Due to the usage of linear programming, their domain of discourse is confined to finite cases as in Poole's approach.

Natural language processing contains logical and probabilistic aspects. Hashida [4, 5] proposed a rather general framework for natural language processing by probabilistic constraint logic programming. Although formal semantics has not been provided, he assigned probabilities not to literals but to "between literals," and let them denote the degree of the possibility of invocation. He has shown constraints are efficiently solvable by making use of these probabilities. He also related his approach to the notion of utility.

We have tightly connected programming with learning in terms of distribution semantics. We hope that our semantics and a learning mechanism will shed light on the interaction between symbol processing and statistical data.

References

[1] Ackley,D.H., Hinton,G.E. and Sejnowski,T.J., A learning algorithm for Boltzmann machines, Cognitive Sci. 9, pp147-169, 1985.

[2] Feller,W., An Introduction to Probability Theory and Its Applications (2nd. ed), Wiley, 1971.

[3] Gaifman,H. and Snir,M., Probabilities over Rich Languages, Testing and Randomness, J. of Symbolic Logic 47, pp495-548, 1982.

[4] Hashida,K., Dynamics of Symbol Systems, New Generation Computing, 12, pp285-310, 1994.

[5] Hashida,K., et al. Probabilistic Constraint Programming (in J), SWoPP'94, 1994.

[6] Hintikka,J., Aspects of Inductive Logic Studies in Logic and the Foundation of Mathematics, North-Holland, 1966.

[9] This is mainly due to the acyclicity assumption made in [13]. It excludes any tautological clause such as $a \leftarrow a$ and any clause containing local variables such as Y in $a(X) \leftarrow b(X, Y)$ when the domain is infinite.

[7] Lakshmanan,L.V.S. and Sadri,F., Probabilistic Dedutive Databases, Proc. of ILPS'94 pp254-268, 1994.

[8] Muggleton,S., Inductive Logic Programming, New Generation Computing 8, pp295-318, 1991.

[9] Ng,R. and Subrahmanian,V.S., Probabilistic Logic Programming, Information and Computation 101, pp150-201, 1992.

[10] Nishio,M., Probability theory (in J), Jikkyo Syuppan, 1978.

[11] Nilsson,N.J., Probabilistic Logic, Artificial Intelligence 28, pp71-87, 1986.

[12] Pearl,J., Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, 1988.

[13] Poole,D., Probabilistic Horn abduction and Bayesian networks, Artificial Intelligence 64, pp81-129, 1993.

[14] Rabiner,L.R., A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, Proc. of the IEEE, Vol. 77, No. 2, pp257-286, 1989.

[15] Tanner,M., Tools for Statistical Inference (2nd ed.), Springer-Verlag, 1986.