Stony Brook University

Inference and Learning in Probabilistic Logic Programs with Continuous Random Variables

A Dissertation Presented

by

Muhammad Asiful Islam

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

August 2012

Stony Brook University

The Graduate School

Muhammad Asiful Islam

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

C.R. Ramakrishnan – Dissertation co-Advisor
Associate Professor, Department of Computer Science

I.V. Ramakrishnan – Dissertation co-Advisor
Professor, Department of Computer Science

David Warren – Chairperson of Defense
Professor, Department of Computer Science

Vítor Santos Costa
Associate Professor
University of Porto

This dissertation is accepted by the Graduate School.

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

Inference and Learning in Probabilistic Logic Programs with Continuous Random Variables

by

Muhammad Asiful Islam

Doctor of Philosophy

in

Computer Science

Stony Brook University

2012

Statistical Relational Learning (SRL), an emerging area of Machine Learning, aims at modeling problems which exhibit complex relational structure as well as uncertainty. It uses a subset of first-order logic to represent relational properties, and graphical models to represent uncertainty. Probabilistic Logic Programming (PLP) is an interesting subfield of SRL. A key characteristic of PLP frameworks is that they are conservative extensions to non-probabilistic logic programs which have been widely used for knowledge representation. PLP frameworks extend traditional logic programming semantics to a *distribution semantics*, where the semantics of a probabilistic logic program is given in terms of a distribution over possible models of the program. However, the inference techniques used in these works rely on enumerating sets of explanations for a query answer. Consequently, these languages permit very limited use of

random variables with continuous distributions.

In this thesis, we extend PRISM, a well-known PLP language, with Gaussian random variables and linear equality constraints over reals. We provide a well-defined distribution semantics for the extended language. We present a *symbolic* inference and parameter-learning algorithms for the extended language that represents sets of explanations without enumeration. This permits us to reason over complex probabilistic models such as Kalman filters and a large subclass of Hybrid Bayesian networks that were hitherto not possible in PLP frameworks. The inference algorithm can be extended to handle programs with Gamma-distributed random variables as well. An interesting aspect of our inference and learning algorithms is that they specialize to those of PRISM in the absence of continuous variables. By using PRISM as the basis, our inference and learning algorithms match the complexity of known specialized algorithms when applied to Hidden Markov Models, Finite Mixture Models and Kalman Filters.

Contents

List of Figures

Acknowledgements

I am very grateful to my advisors Prof. I.V. Ramakrishnan and Prof. C.R. Ramakrishnan. Prof. I.V. Ramakrishnan ignited my passion for research and made me fall in love with the exciting world of research. I am indebted to Prof. C.R. Ramakrishnan for his valuable and thoughtful inputs throughout this research. He taught me how to be meticulous in solving research problems.

I would like to extend my gratitude to Prof. David Warren and Prof. Vitor Santos Costa for their valuable comments and suggestions. I would also like to thank the reviewers for their valuable comments to improve our papers.

I am grateful to my parents, friends and colleagues for their support and encouragement.

Finally, I would like to thank the National Science Foundation (Grants CCF-1018459, CCF-0831298) and ONR (Grant N00014-07-1-0928) for supporting this research.

Chapter 1

Introduction

Statistical Relational Learning (SRL) [22] research aims at performing inference and learning in domains with complex relational and probabilistic structure. Traditional Machine Learning models (e.g., neural networks, decision trees, support vector machines, etc) use point based semantics, and ignore relational aspects of the data. SRL frameworks attempt to overcome this limitation of statistical models by capturing the relational structure of the domain as well as expressing the probabilistic model in a compact and natural way.

The SRL frameworks are based on combinations of graphical models and logical formulae (e.g., first-order). Thus the frameworks can be broadly classified as statistical-model-based or logic-based, depending on how their semantics is defined. The semantics of the statistical-model-based frameworks, e.g., Bayesian Logic Programs (BLPs) [31], Probabilistic Relational Models (PRMs) [20], and Markov Logic Networks (MLNs) [51], is given in terms of the ground graphical model. For example, MLN [51] gives a template for constructing Markov Random Field (MRF) [43], and uses first-order logical rules to specify a model compactly. Thus inference in MLNs follow the inference technique of the underlying MRFs.

In the second category are frameworks such as PRISM [54], Stochastic Logic Programs (SLP) [39], Independent Choice Logic (ICL) [45], and ProbLog [50]. These Probabilistic Logic Programming (PLP) frameworks are designed for combining statistical and logical knowledge representation and inference. PLP languages permit incorporation of statistical knowledge via the use of implicit or explicit random variables in a logic program. PLP languages such as SLP [39] and ProbLog [50] attach implicit random variables with certain clauses in a logic program; a clause's applicability is determined by the value of the associated random variable. Other languages such as PRISM [54] use explicit random variables whose valuations are specified analogous to relations in an extensional database.

The semantics of PLP languages is defined based on the semantics of the underlying non-probabilistic logic programs [5, 36]. A large class of PLP languages, including ICL [45], PRISM [54], ProbLog [50] and LPAD [62], have a declarative *distribution semantics*, which defines a probability distribution over possible models of the program. Operationally, the combined statistical/logical inference is performed based on the proof structures analogous to those created by purely logical inference. In particular, inference proceeds as in traditional LPs except when a random variable's valuation is used. Use of a random variable creates a branch in the proof structure, one branch for each valuation of the variable. Each proof for an answer is associated with a probability based on the random variables used in the proof and their distributions; an answer's probability is determined by the probability that at least one proof holds.

PLP languages have interesting algorithmic properties because of the logical proof structure. For example, PRISM's inference technique for Hidden Markov Models naturally reduces to Viterbi algorithm [19]. These languages has been applied in a number of applications, e.g., biological sequence analysis [9, 10, 50], analysis of chemical database [14], discovery of structural alerts for carcinogenicity [58], 3D pharmacophore discovery for drug design [18], protein structure prediction [17], knowledge synthesization in sensor networks [56], probabilistic planning [16], and so on. An overview of such applications appears in [16, 17].

Since the inference is based on enumerating the proofs/explanations for answers, the PLP languages have limited support for continuous random variables. There has been a very recent attempt at addressing this problem in [24]. While a more detailed comparison appears in Section 3.1 on Related Work, it is sufficient to say here that this work can not handle interesting problems such as Kalman Filters [53] and Hybrid Bayesian Networks where the parent/child conditional dependencies are captured using arbitrary discrete/continuous random variable combinations [40]. Thus a rigorous and extensive treatment of probabilistic LP along the lines of PRISM semantics, that includes both discrete and continuous random variables remains open and is the topic addressed in this thesis.

We provide an inference procedure to reason over PLPs with Gaussian or Gamma-distributed random variables (in addition to discrete-valued ones), and linear equality constraints over values of these continuous random variables. We describe the inference procedure based on extending PRISM with continuous random variables. This choice is based on the following reasons. First of all, the use of explicit random variables in PRISM simplifies the technical development. Secondly, standard statistical models such as Hidden

Markov Models (HMMs) [47], Bayesian Networks and Probabilistic Context-Free Grammars (PCFGs) can be naturally encoded in PRISM. Along the same lines, our extension permits natural encodings of Finite Mixture Models (FMMs) [38] and Kalman Filters. Thirdly, PRISM's inference, which is based on OLDT resolution [60], naturally reduces to the Viterbi algorithm [19] over HMMs, and the Inside-Outside algorithm [34] over PCFGs. This enables us to derive an inference procedure that naturally reduces to the ones used for evaluating Finite Mixture Models and Kalman Filters. The combination of well-defined model theory and efficient inference has enabled the use of PRISM for synthesizing knowledge in sensor networks [56].

It should be noted that, while the technical development in this thesis is limited to PRISM, the basic technique itself is applicable to other similar PLP languages such as ProbLog and LPAD (see Section 6.5).

We also address the important topic of learning the probability distributions of the random variables in extended PRISM programs in this thesis. Parameter learning in the PLP languages is typically done by variants of the EM algorithm [15]. The key aspect of our inference as well as learning algorithms is the construction of *symbolic derivations* that succinctly represent large (sometimes infinite) sets of traditional logical derivations. Our learning algorithm represents and computes Expected Sufficient Statistics (ESS) symbolically as well, for Gaussian as well as discrete random variables. It should be noted that our learning algorithm reduces to PRISM's graphical EM [55] in the absence of continuous random variables.

Our Contribution: We extend PRISM at the language level to seamlessly include discrete as well as continuous random variables. We develop inference and learning procedures to evaluate queries over such extended PRISM programs.

- For extending the PRISM language we introduce two ideas: distribution and constraints over continuous random variables. These two ideas enable the encoding of rich statistical models such as Kalman Filters and a large class of Hybrid Bayesian Networks which were hitherto not expressible in LP and its probabilistic extensions.

- To evaluate queries on extended PRISM programs we develop a *symbolic inference* technique to reason with constraints on the random variables. PRISM's inference technique becomes a special case of our technique when restricted to logic programs with discrete random variables.

- We develop an algorithm for parameter learning in PRISM extended with Gaussian random variables. It computes Expected Sufficient Statistics (ESS) symbolically for Gaussian as well as discrete random variables.

Note that the technique of using PRISM for in-network evaluation of queries in a sensor network [56] can now be applied directly when sensor data and noise are continuously distributed. Tracking and navigation problems in sensor networks are special cases of the Kalman Filter problem [11]. There are a number of other network inference problems, such as the indoor localization problem, that have been modeled as FMMs [23]. Moreover, our extension permits reasoning over models with finite mixture of Gaussians and discrete distributions. Our extension of PRISM brings us closer to the ideal of finding a declarative basis for programming in the presence of noisy data.

The rest of this thesis is organized as follows. We begin with a brief overview of statistical Machine Learning techniques in Chapter 2. We give a review of related work in Chapter 3 and describe the PRISM framework in detail in Chapter 4. Chapter 5 introduces the extended PRISM language. The symbolic inference technique for the extended language is presented in Chapter 6. Finally, we present the learning algorithm in Chapter 7 and conclude in Chapter 8.

Chapter 2

Background on Statistical ML

This chapter gives a brief overview of statistical Machine Learning techniques and the notations used in this thesis. Upper case letters (e.g., V) are used to denote variables. We use $\overline{X}$ to denote a vector of variables (e.g., $\overline{X} = \langle X_1, X_2 \rangle$), explicitly specifying the size only when it is not clear from the context. Note that the vector notation can be used to compactly represent linear combination of variables. For example $\overline{A} \cdot \overline{X}$ denotes a linear combination of variables.

Example 1. *If* $\overline{A} = \langle 2,\ 3,\ -1 \rangle$ *and* $\overline{X} = \langle X,\ Y,\ Z \rangle$, *then* $\overline{A} \cdot \overline{X} = 2X + 3Y - Z$.

2.1 Gaussian Distribution

The Gaussian or normal distribution is one of the most prominent or widely used models for the distribution of continuous variables. We use $\mathcal{N}_V(\mu, \sigma^2)$ to denote the Gaussian/normal probability density function (PDF) of random variable V.

$$\mathcal{N}_V(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(V-\mu)^2}{2\sigma^2}} \tag{2.1}$$

The distribution with $\mu = 0$ and $\sigma^2 = 1$ is called the *standard normal* distribution.

Note that integration of Equation 2.1 over its entire range is one.

$$\int_{-\infty}^{\infty} \mathcal{N}_V(\mu, \sigma^2) dv = 1$$

$\mathcal{N}_{(\overline{A} \cdot \overline{X})}(\mu, \sigma^2)$ denotes a Gaussian density function (with mean μ and variance σ^2) of a linear combination $\overline{A} \cdot \overline{X}$.

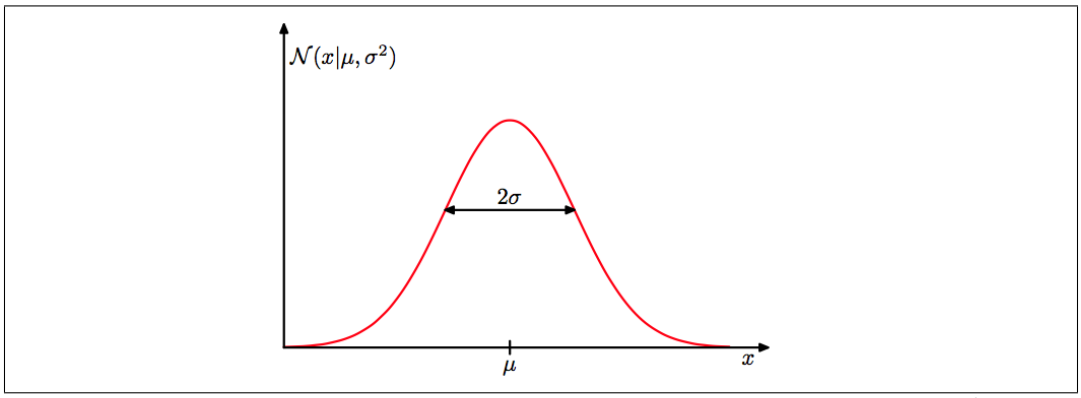


Figure 2.1: Gaussian distribution with mean μ and variance σ^2

Example 2. *Let* $\overline{A} = \langle 2,\ 3,\ -1\rangle$ *and* $\overline{X} = \langle X,\ Y,\ Z\rangle$. *Then the Gaussian density of the linear combination of variables* $\overline{A} \cdot \overline{X} = 2X + 3Y - Z$ *is*

$$\mathcal{N}_{(2X+3Y-Z)}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(2X+3Y-Z-\mu)^2}{2\sigma^2}}.$$

2.1.1 Properties of Gaussian Distributions

The following properties of Gaussian distributions are used in this thesis [1, 4].

Property 1. Standardizing normal random variables. *If* X *is normal distributed with mean* μ *and variance* σ^2, *then*

$$Z = \frac{X - \mu}{\sigma}$$

has mean zero and unit variance. Thus Z *has the standard normal distribution.*

Property 2. Product of two Gaussian PDFs is another Gaussian PDF.

$$\mathcal{N}_X(\mu_1, \sigma_1^2).\mathcal{N}_X(\mu_2, \sigma_2^2) = \mathcal{N}_X(\mu, \sigma^2)$$

where

$$\mu = \frac{\mu_1 * \sigma_2^2 + \mu_2 * \sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$
$$\sigma^2 = \frac{\sigma_1^2 * \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

Property 3. Normal distribution is closed under linear transformations. *If* X *is normal distributed with mean* μ *and variance* σ^2 *(i.e.,* $X \sim \mathcal{N}(\mu, \sigma^2)$*),*

then a linear transformation $aX + b$ is also normally distributed:

$$aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2).$$

If X_1, X_2 are two independent Gaussian random variables (with means μ_1, μ_2 and standard deviations σ_1^2, σ_2^2), then their linear combination $aX_1 + bX_2$ is also normally distributed:

$$aX_1 + bX_2 \sim \mathcal{N}(a\mu_1 + b\mu_2, a^2\sigma_1^2 + b^2\sigma_2^2).$$

Note that the converse of the above property is also true, i.e., if X_1 and X_2 are independent and their sum $X_1 + X_2$ is normally distributed, then both X_1 and X_2 must also be normally distributed.

Although Gaussian distribution has many other important properties, we summarize only the above mentioned properties as they are sufficient to understand the technical part of this thesis.

2.1.2 Estimation of Parameters

Suppose that we have a collection of observations $x_1, x_2, ..., x_n$ from a normal ($\mathcal{N}_X(\mu, \sigma^2)$) population and we would like to learn the approximate values of the parameters μ and σ^2. The standard approach to solve this problem is the maximum likelihood method which requires maximization of the following log-likelihood function:

$$LL(\mu, \sigma^2) = \sum_{i=1}^{n} \ln \mathcal{N}_X(x_i|\mu, \sigma^2) \tag{2.2}$$

Note that in practice, it is more convenient to maximize the log of the likelihood function as it simplifies the computation. Since the logarithm is a monotonically increasing function of its argument, maximization of the log of a function is equivalent to maximization of the function itself.

Now maximizing 2.2 (i.e., taking derivatives and equating to zero) with respect to μ and σ^2, we obtain the following maximum likelihood estimates

$$\mu_{MLE} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.3}$$

and

$$\sigma^2_{MLE} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_{MLE})^2. \tag{2.4}$$

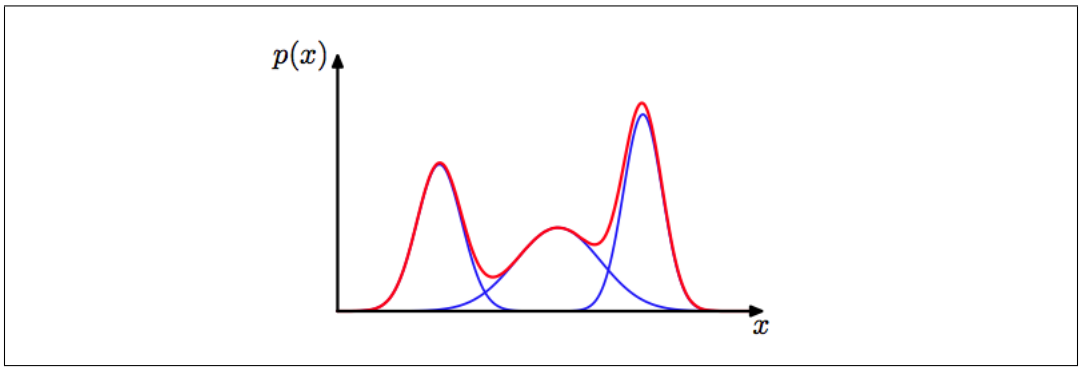


Figure 2.2: Mixture of Gaussians

2.1.3 Mixture of Gaussians

Although Gaussian distribution has many important analytical properties, it can not model multimodal distribution. Thus it suffers from significant limitations while modeling real data sets which do not follow a single Gaussian distribution. However a linear superposition of two or more Gaussians gives a better characterization of the data. The resultant distribution is called a *mixture distribution*. Figure 2.2 shows an example of Gaussian mixture distribution (component distributions in blue and their sum in red).

The density of a *mixture of Gaussian*, formed from K Gaussian distributions, has the following general form

$$p(X) = \sum_{k=1}^{K} p_k \mathcal{N}_X(\mu_k, \sigma_k^2). \tag{2.5}$$

where $\mathcal{N}_X(\mu_k, \sigma_k^2)$ denotes a Gaussian component of the mixture with mean μ_k and variance σ_k^2. p_k's are called mixing coefficient that sum up to 1.

$$\sum_{k=1}^{K} p_k = 1$$

In equation 2.5, p_k can be viewed as the prior probability of picking the k^{th} component, and the density $\mathcal{N}_X(\mu_k, \sigma_k^2)$ can be viewed as the probability of X conditioned on k.

Note that any continuous density can be approximated by using a sufficient number of Gaussians, and by adjusting their means and covariances as well as the mixing coefficient.

2.1.4 Multivariate Gaussian distribution

The multivariate Gaussian distribution has the following form for a d dimensional vector $\mathbf{X}$.

$$\mathcal{N}_{\mathbf{X}}(\mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}|}} \exp -\frac{1}{2}(\mathbf{X} - \mu)^T \mathbf{\Sigma}^{-1}(\mathbf{X} - \mu) \tag{2.6}$$

where μ is a d-dimensional mean vector, $\mathbf{\Sigma}$ is a $d * d$ covariance matrix and $|\mathbf{\Sigma}|$ denotes the determinant of $\mathbf{\Sigma}$.

Notice that Equation 2.1 is a special case (when $d = 1$) of Equation 2.6. Multivariate Gaussian also follows the properties (e.g., linear transformation, product and standardization) of univariate Gaussian. In this thesis, we'll discuss the technical sections using only univariate Gaussians.

2.2 Gamma Distribution

Gamma distribution is also an important continuous distribution in statistics. It is frequently used to model waiting time (e.g., life time of a light bulb until death is a Gamma random variable). Gamma distribution specializes to many other continuous distributions (e.g., Erlang distribution, Exponential distribution, etc). The probability density function of a Gamma distributed random variable X is

$$Gamma(k, \theta) = \frac{1}{\theta^k} \frac{1}{\Gamma(k)} X^{k-1} e^{-\frac{X}{\theta}} \tag{2.7}$$

where k and θ are shape and scale parameters respectively and $\Gamma(k) = (k-1)!$ when k is a positive integer.

Note that when $k = 1$, X has an exponential distribution with rate parameter $1/\theta$.

2.2.1 Properties of Gamma Distributions

The following properties of Gamma distributions are used in this thesis [4].

Property 4. Scaling. *If X is Gamma distributed with shape and scale parameters k and θ respectively, then aX is also Gamma distributed:*

$$aX \sim Gamma(k, a\theta).$$

Property 5. Summation. *If X_1, X_2 are two independent Gamma random variables (with shapes k_1, k_2 and scale parameter θ), then their summation*

$X_1 + X_2$ is also Gamma distributed:

$$X_1 + X_2 \sim Gamma(k_1 + k_2, \theta).$$

2.3 Hybrid Bayesian Networks

Bayesian Networks: Bayesian networks are graphs where each node represents a random variable and the arcs represent direct/causal relationship among the random variables (absence of arcs represent conditional independence assumption). Thus Bayesian network provides a compact way to represent the joint probability distribution of random variables. Let $X_1, X_2, ..., X_n$ be a set of random variables. We use $P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n)$ or simply $P(x_1, x_2, ..., x_n)$ to denote the joint probability distribution of the random variables. Let $Pa(X_i)$ denote the parents of node X_i. We can use the chain rule of probability to compute the joint distribution $P(x_1, x_2, ..., x_n)$ as follows

$$\begin{aligned} P(x_1, x_2, ..., x_n) &= P(x_n|x_{n-1}, ..., x_1)P(x_{n-1}|x_{n-2}, ..., x_1)...P(x_2|x_1)P(x_1) \\ &= \prod_{i=1}^{n} P(x_i|x_{i-1}, ..., x_1) \end{aligned}$$

Bayesian network simplifies this computation by introducing the conditional independence assumption which states that the conditional probability of a node is independent of the other nodes given its parents. Thus we can write the above equation as follows

$$P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i|Pa(x_i))$$

Hybrid Bayesian Networks: A Hybrid Bayesian Network (HBN) represents a probability distribution over a set of random variables where some are discrete and some are continuous. Hybrid Bayesian Networks can be divided into the following four categories based on the parent-child relationship.

1. *Discrete child and discrete parent.* (e.g., Hidden Markov Model).
2. *Continuous child and discrete parent.* (e.g., Finite Mixture Model).
3. *Continuous child and continuous parent.* (e.g, Kalman Filters).
4. *Discrete child and continuous parent.*

In the following subsections, we describe each of them in more detail.

2.3.1 Discrete child and discrete parent HBN

In discrete child-discrete parent Bayesian network, the conditional distribution of a discrete node given its parent is specified by means of a table called conditional probability table or CPT.

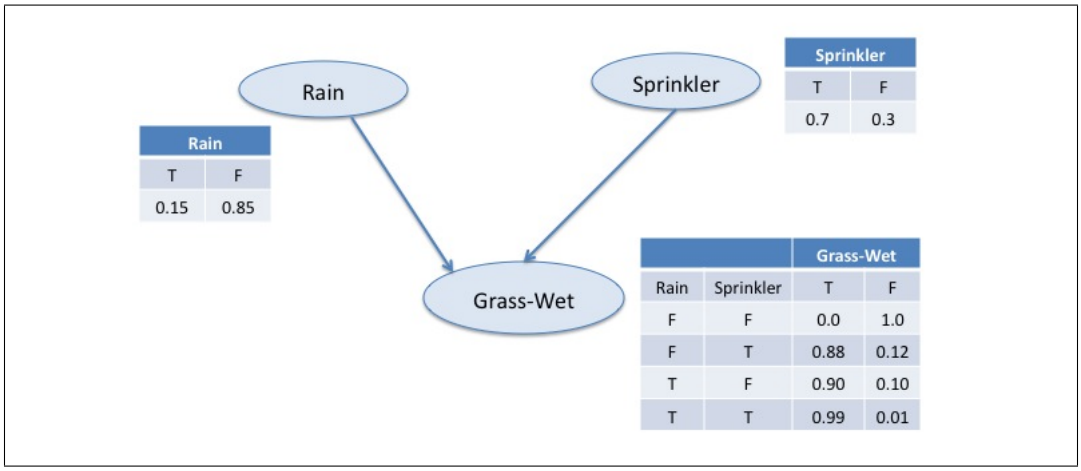


Figure 2.3: Discrete child and discrete parent HBN

Example 3. *Figure 2.3 presents a Bayesian network with three random variables namely* Rain, Sprinkler *and* Grass-Wet. *Each node represents a binary valued (i.e., True/False) random variable. Here the table associated with each node represents the CPT. Here the random variable GrassWet depends on other two random variable Rain and Sprinkler.*

Bayesian networks that model sequences of variables (e.g., speech signals, trajectory) are called dynamic Bayesian networks. Hidden Markov Model (HMM) is a classic example of dynamic Bayesian network where states and observation variables are discrete. In Figure 2.4, node S_i denotes a state and node V_j denotes an observation. In HMM, the CPTs define the transition and observation probabilities.

2.3.2 Continuous child and discrete parent HBN

The conditional distribution of a continuous node X_i given its parents $Pa(X_i)$ is specified using a Gaussian function, i.e., for each value of the discrete parent node, the child node has a Gaussian distribution. The conditional distribution of the child node is called a conditional probability density or CPD, and it is represented as follows

$$P(X_i|Pa(X_i)) = \mathcal{N}_{X_i}(\mu_{pa(x_i)}, \sigma^2_{pa(x_i)})$$

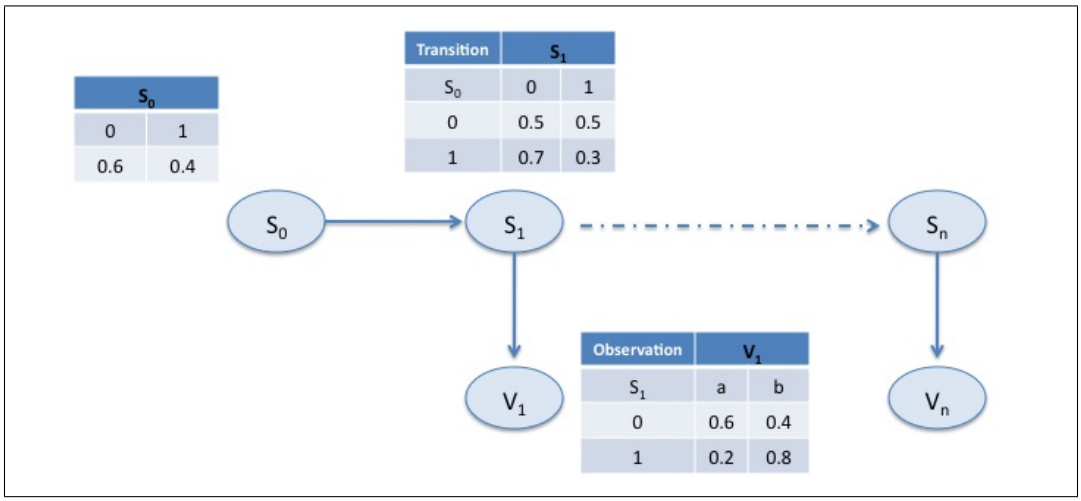


Figure 2.4: Hidden Markov Model

where $pa(x_i)$ denotes the valuation of the discrete parent nodes of X_i. Thus $P(X_i|Pa(X_i))$ represents a Gaussian distribution where mean and variances are conditioned on the values of discrete parents.

Example 4. *Finite Mixture Model is a classic example of Continuous child-discrete parent Bayesian network. Figure 2.5 presents a Bayesian network with two random variables* Machine *and* Widget*. Machine is a discrete random variable, taking values a and b. Each machine produces* Widget *whose weights are continuous valued.*

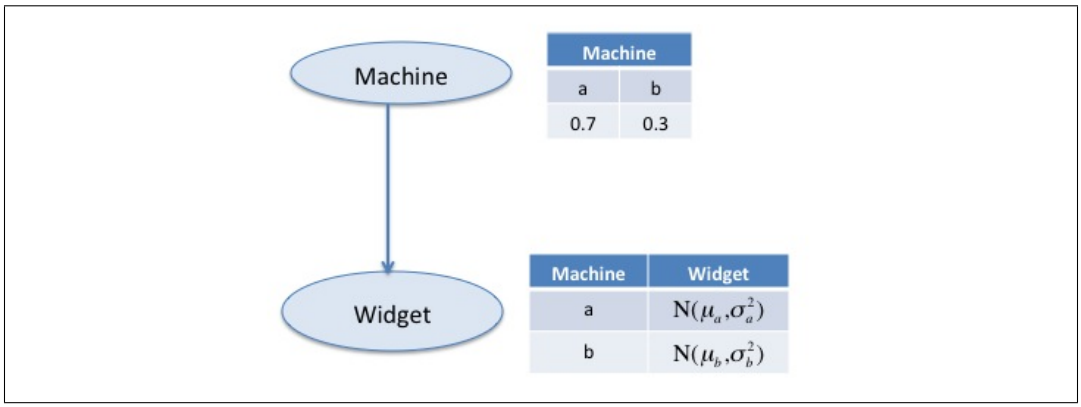


Figure 2.5: Continuous child and discrete parent HBN

2.3.3 Continuous child and continuous parent HBN

The conditional distribution of a continuous node X_i given its continuous parents $Pa(X_i)$ is represented as

$$P(X_i|Pa(X_i)) = \mathcal{N}_{X_i}(u_i, \sigma_i^2)$$

where $u_i = \mu_i + \sum_{k \in Pa(X_i)} b_{ki}(x_k - \mu_k)$, and the b_{ki} are the weights coming into node i from its parents k. Thus the random variable X_i can be represented using the following form

$$X_i = \mu_i + \sum_{k \in Pa(X_i)} b_{ki}(X_k - \mu_k)$$

In general, continuous child-continuous parent Bayesian networks are represented as

$$X_i = \sum_{k \in Pa(X_i)} b_{ki} X_k + Y_i$$

where $Y_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ is a Gaussian noise term.

Thus a continuous child node can be specified as a linear function of its continuous parent nodes. These type of Bayesian networks are called Conditional Linear Gaussians or CLG models.

Example 5. *Kalman Filters (KF) is a classic example of dynamic Bayesian network where state and observation variables are continuous. Let S_i denote a state variable and V_i denote an observation variable. Then the state transition is a CLG model where next state is a linear combination of current state and a Gaussian noise, i.e,*

$$S_{i+1} = S_i + E.$$

Similarly, observation also follows CLG model where observation at state S_i depend on the state S_i and a Gaussian noise, i.e.,

$$V_i = S_i + X.$$

2.3.4 Discrete child and continuous parent HBN

The conditional distribution of a discrete child given its continuous parent is represented by a *softmax* or *logistic* function. Let V be a discrete node with the possible values $v_1, v_2, ..., v_m$, and $X_1, X_2, ..., X_n$ be its parents. Then the

conditional distribution is represented as follows

$$P(V = v_i|x_1, x_2, ..., x_n) = \frac{\exp^{(b^i + \sum_{l=1}^{n} w_l^i x_l)}}{\sum_{j=1}^{m} \exp^{(b^j + \sum_{l=1}^{n} w_l^j x_l)}}$$

where w_l's are weights coming into node V from its parents X_l. For binary valued variable, the softmax function simplifies to standard *sigmoid* function,

$$P(V = v_1|x_1, x_2, ..., x_n) = \frac{1}{1 + \exp^{(b + \sum_{l=1}^{n} w_l x_l)}}$$

2.3.5 Inference in Bayesian networks

Inference in Bayesian networks involves computing the posterior distribution of some random variable given some observations or evidences. Let X denote a query variable and E denote an evidence variable. Then a query $P(X|E)$ can be answered using the following equation

$$P(X|E) = \alpha P(X, E) = \alpha \sum_{Y} P(X, E, Y)$$

where Y denotes hidden or latent variables. Thus a query can be answered by computing sums of products of the conditional probabilities of the Bayesian network.

The above mentioned inference technique sums over the joint probability distribution and is computationally expensive (exponential on the number of variables). The variable elimination algorithm substantially improves the performance by eliminating repeated calculation with the help of dynamic programming. The key idea of variable elimination is to push sums in as far as possible while summing out hidden variables. Other exact inference algorithms in Bayesian network include clique tree propagation, recursive conditioning and AND/OR search. Common approximate inference algorithms include sampling, Markov Chain Monte Carlo (MCMC), belief propagation, and variational methods [4, 53].

Inference in Dynamic Bayesian Networks. The general inference problem for dynamic Bayesian networks is to compute $P(X_t|Y_{(t_1,t_2)})$, where X_t represents the hidden/state variable at time t, and $Y_{(t_1,t_2)}$ represents all the observations between times t_1 and t_2.

Depending on the time steps and observations, we can divide the inference task into the following categories:

1. *Filtering:* is the task of computing the posterior distribution of current

state given all the evidence to date, i.e., $P(X_t|Y_{(1,t)})$.

2. *Prediction:* is the task of computing the posterior distribution of a future state given all the evidence to date, i.e., $P(X_{t+\delta}|Y_{(1,t)})$.

3. *Smoothing:* is the task of computing the posterior distribution of a past state given all the evidence to date, i.e., $P(X_k|Y_{(1,t)})$, for some k such that $0 \leq k < t$.

Figure 2.6 shows a graphical representation of these inference tasks.

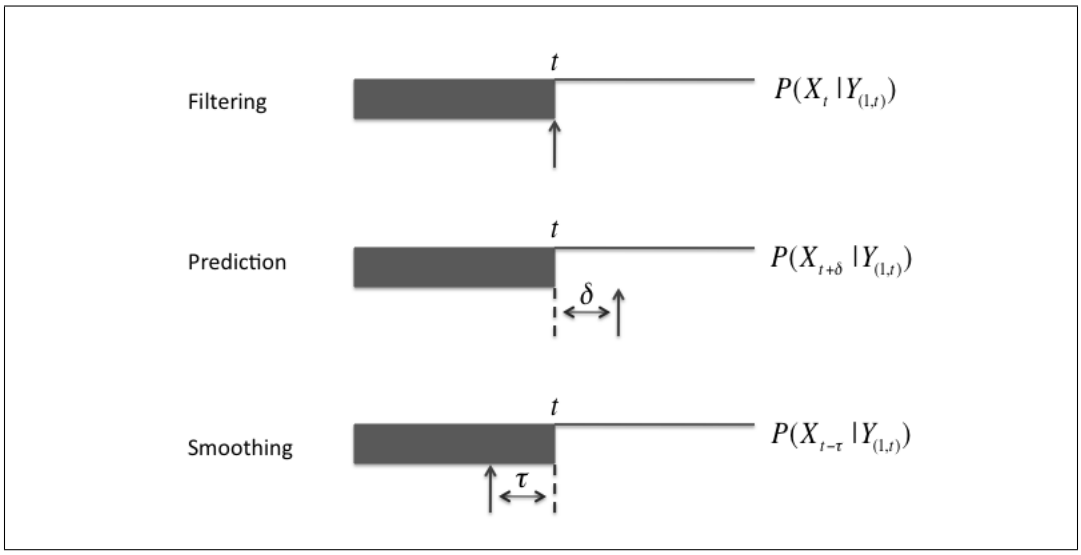


Figure 2.6: Inference in DBN

2.3.6 Parameter Learning in Bayesian networks

The parameter learning task is to estimate the distribution parameters (entries of CPT and/or Gaussian mean, variance) given a set of N training examples. We discuss how to find the Maximum Likelihood Estimates (MLEs) of the parameters in fully observable case here. For partially observable data, Expectation Maximization algorithm (discussed in Section 2.4) is used to learn the distribution parameters.

1. *Discrete case.* If X_i is a discrete random variable, then the parameter value $\theta_{ijk} = P(X_i = k|Pa(X_i) = j)$ is computed as follows

$$\theta_{ijk} = P(X_i = k|Pa(X_i) = j) = \frac{P(X_i = k, Pa(X_i) = j)}{Pa(X_i) = j)} = \frac{N_{ijk}}{N_{ij}}$$

where N_{ijk} denotes the number of times the event $(X_i = k, Pa(X_i) = j)$ occurs in the training set and $N_{ij} = \sum_k N_{ijk}$. So the sufficient statistics to estimate the distribution parameters are N_{ijk}.

2. *Continuous case.* If X_i is a continuous random variable, then the sufficient statistics to compute the mean and variance are $S_N = \sum_{l=1}^{N} x_l$ and $Q_N = \sum_{l=1}^{N} x_l^2$, since

$$\mu = \frac{1}{N} S_N = \frac{1}{N} \sum_{l=1}^{N} x_l$$

and

$$\begin{aligned} \sigma^2 &= \frac{1}{N} \sum_{l=1}^{N} (x_l - \mu)^2 \\ &= \frac{1}{N} Q_N - \mu^2 \end{aligned}$$

Structure Learning. The goal of the structure learning is to learn the directed graph of the Bayesian network that best explains the data. Structure learning algorithms uses optimization based search which requires a scoring function and a search strategy. In Probabilistic Logic Programming languages, we are interested in learning the distribution parameters. Thus structure learning is not a relevant problem for us.

2.3.7 Application of Bayesian networks

Bayesian networks are primarily used in Statistics and Machine Learning problems to model joint distribution of random variables. Recently, it has been extensively used for modeling knowledge in bioinformatics, computational biology, bio-surveillance, image processing, document classification, information retrieval, decision support systems and engineering [46].

2.3.8 Hybrid Models

Delta function. A Dirac-delta function, denoted by $\delta_c(X)$, represents a function which is zero everywhere except at point c and the integral is one over its entire range. The delta function is used in probability theory to represent discrete distribution. For example, the probability mass function $(P(V))$ of a discrete random variable V, taking values *head* and *tail* with probability 0.6

and 0.4 respectively, can be represented using the delta function as follows:

$$P(V) = 0.60\delta_{head}(V) + 0.40\delta_{tail}(V)$$

Hybrid Models. In probability theory, a hybrid probability distribution is a distribution which is partly discrete and partly continuous. Hybrid models can be thought as a mixture distribution where some component distributions are discrete (unlike Gaussian mixtures where all the component distributions are Gaussian). For example, consider a distribution which 0.7 of the time returns a value drawn from standard normal distribution and 0.3 of the time returns exactly the value 3.0. The density of this distribution can be written as

$$f(X) = 0.7\mathcal{N}_X(0.0, 1.0) + 0.3\delta_{3.0}(X)$$

where $\delta_{3.0}(X)$ denotes a Dirac-delta function.

Hybrid models is used to express complex densities in terms of simpler densities (discrete and continuous), and provide a good model for some data sets where different subsets of the data exhibit different characteristics [38].

2.4 Expectation-Maximization Algorithm

When data is incomplete, i.e., in the presence of hidden or missing information, we cannot use the direct computation for MLE as described in Section 2.1.2. Expectation-maximization (EM) is an iterative algorithm in statistics for finding maximum likelihood estimates of parameters in probabilistic models, where the model depends on unobserved latent variables. More specifically, the EM algorithm alternates between performing an expectation (E) step, which computes an expectation of the log likelihood with respect to the current estimate of the distribution for the latent variables, and a maximization (M) step, which computes the parameters that maximize the expected log likelihood found on the E step.

Given a likelihood function $P(X, Z|\theta)$, where θ is the model parameter, X is the observed data and Z represents the unobserved latent data or missing values; the EM algorithm seeks to find the MLE by iteratively applying the following two steps:

1. *Expectation step:* Calculate the expected value of the log likelihood function, with respect to the conditional distribution of Z given X under the current estimate of the parameters θ_n:

$$Q(\theta, \theta_n) = E_{Z|X,\theta_n}[lnP(X, Z|\theta)]$$

2. *Maximization step:* Find the parameter which maximizes the likelihood:

$$\theta_n = argmax_\theta Q(\theta, \theta_n)$$

Applications. The EM algorithm is frequently used for data clustering in machine learning, data mining and computer vision. Two prominent instances of the algorithm are the Baum-Welch algorithm [15] (also known as forward-backward) and the inside-outside algorithm for probabilistic context-free grammars [34]. PLP languages such as PRISM and ProbLog use EM-based learning algorithms [26, 29, 55] for estimating distribution parameters.

2.4.1 Alternative view of the EM algorithm

In this section, we present an alternative view of the EM algorithm in terms of expected sufficient statistics [40] of the random variables. The main idea of this approach is the computation of expected value when the actual value is unknown.

Sufficient Statistics. Recall that Equations 2.3 and 2.4 compute the MLE of the Gaussian distribution parameters μ and σ^2. To compute the MLE using those equations, we need to know three quantities $n, \sum x_i$ and $\sum x_i^2$. These quantities are called the sufficient statistics (SS) of Gaussian random variables [40].

For a discrete random variable, its sufficient statistics is the total count of each valuation. For example, if a discrete random variable V takes values a and b, then its sufficient statistics are N_a and N_b where N_a (N_b) is the total number of times V takes value a (b). These quantities are called sufficient statistics because one can compute the distribution of V (i.e., probability of $V = a$ and $V = b$) by knowing only N_a and N_b.

Expected Sufficient Statistics. When we do not know the exact value of some quantity, we can not compute the sufficient statistics. In this case, we compute its expected value or expected sufficient statistics (ESS) [28, 40]. For example, if we do not know all the values in Equations 2.3 and 2.4, then we compute the expected values of $n, \sum x_i$ and $\sum x_i^2$ to estimate μ, σ^2. Similarly for discrete random variables we compute the expected total count.

The main idea of the EM-algorithm is to fill in the missing values with their expected values (expectation w.r.t. the current set of parameters) and use these expected sufficient statistics (ESS) while computing the MLE. Thus the EM-algorithm can be simplified in the following two steps.

- *Expectation step:* Compute the expected sufficient statistics (ESS) of the random variables with respect to the current estimate of the parameters θ_n.

- *Maximization step:* Use the ESS to compute the MLE of the distribution parameters (θ).

We'll discuss more about ESS while developing our learning algorithm.

2.4.2 Derivation of the EM algorithm

In this section, we present the derivation of the EM algorithm (i.e., E and M-steps). The derivation is useful to understand the meaning of the expectation function ($Q(\theta, \theta_n)$), properties of the algorithm, and PRISM's parameter learning algorithm.

Let X denote a random variable representing observed data and θ denote the model parameters. Let Z denote the hidden variable and z be an instance of Z. The total probability $P(X|\theta)$ can be written as,

$$P(X|\theta) = \sum_z P(X|z,\theta)P(z|\theta).$$

We wish to find θ such that $P(X|\theta)$ is maximum. In order to estimate θ, we maximize the following *log likelihood function*

$$L(\theta) = lnP(X|\theta).$$

Let the current estimate for θ after n^{th} iteration is given by θ_n. Since the objective is to maximize $L(\theta)$, we would like to compute an updated estimate θ such that

$$L(\theta) > L(\theta_n).$$

which is equivalent to maximizing the difference of log likelihoods,

$$\begin{aligned} L(\theta) - L(\theta_n) &= lnP(X|\theta) - lnP(X|\theta_n) \\ &= ln\left(\sum_z P(X|z,\theta)P(z|\theta)\right) - lnP(X|\theta_n) \end{aligned} \tag{2.8}$$

Notice that the above expression involves the logarithm of sum. We can use Jensen's inequality for log function which states that,

$$ln\sum_{i=1}^{n} \alpha_i x_i \geq \sum_{i=1}^{n} \alpha_i ln(x_i)$$

for constants $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$. We can apply this to Equation 2.8 with $\alpha_i = P(z|X,\theta_n)$.

$$
\begin{aligned}
L(\theta) - L(\theta_n) &= ln\left(\sum_z P(X|z,\theta)P(z|\theta)\right) - lnP(X|\theta_n) \\
&= ln\left(\sum_z P(X|z,\theta)P(z|\theta).\frac{P(z|X,\theta_n)}{P(z|X,\theta_n)}\right) - lnP(X|\theta_n) \\
&= ln\left(\sum_z P(z|X,\theta_n)\frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)}\right) - lnP(X|\theta_n) \\
&\geq \sum_z P(z|X,\theta_n) ln\left(\frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)}\right) - lnP(X|\theta_n)
\end{aligned} \tag{2.9}
$$

Since $\sum_z P(z|X,\theta_n) = 1$,
$lnP(X|\theta_n)$ can be written as $\sum_z P(z|X,\theta_n) lnP(X|\theta_n)$.

$$
\begin{aligned}
L(\theta) - L(\theta_n) &= \sum_z P(z|X,\theta_n) ln\left(\frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)}\right) \\
&\quad - \sum_z P(z|X,\theta_n) lnP(X|\theta_n) \\
&= \sum_z P(z|X,\theta_n) ln\left(\frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)P(X|\theta_n)}\right) \\
&= \delta(\theta|\theta_n)
\end{aligned} \tag{2.10}
$$

$$
L(\theta) \geq L(\theta_n) + \delta(\theta|\theta_n) \tag{2.11}
$$

For simplicity let's define,

$$
l(\theta|\theta_n) = L(\theta_n) + \delta(\theta|\theta_n)
$$

Thus Equation 2.9 can be expressed as

$$
L(\theta) \geq l(\theta|\theta_n)
$$

where $l(\theta|\theta_n)$ is bounded above by the likelihood function $L(\theta)$.

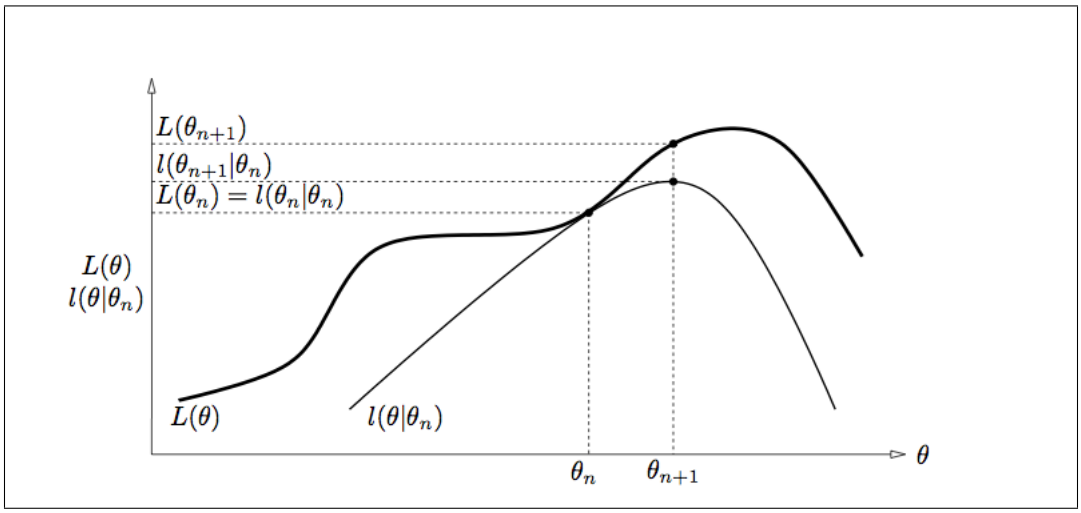


Figure 2.7: Illustration of one iteration of the EM algorithm

Notice that,

$$\begin{aligned} l(\theta_n|\theta_n) &= L(\theta_n) + \delta(\theta_n|\theta_n) \\ &= L(\theta_n) + \sum_z P(z|X,\theta_n) ln \left(\frac{P(X|z,\theta_n)P(z|\theta_n)}{P(z|X,\theta_n)P(X|\theta_n)} \right) \\ &= L(\theta_n) + \sum_z P(z|X,\theta_n) ln \left(\frac{P(X,z|\theta_n)}{P(X,z|\theta_n)} \right) \\ &= L(\theta_n) + \sum_z P(z|X,\theta_n) ln 1 \\ &= L(\theta_n) \end{aligned}$$

Thus $l(\theta|\theta_n)$ and $L(\theta)$ are equal when $\theta = \theta_n$.

Since any θ which increases $l(\theta|\theta_n)$ also increases $L(\theta)$, the EM-algorithm chooses θ_{n+1} as the value of θ for which $l(\theta|\theta_n)$ is a maximum. Figure 2.7 illustrates this process.

Thus

$$\begin{aligned} \theta_{n+1} &= argmax_\theta \{ l(\theta|\theta_n) \} \\ &= argmax_\theta \left\{ L(\theta_n) + \sum_z P(z|X,\theta_n) ln \left(\frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)P(X|\theta_n)} \right) \right\} \end{aligned}$$

Now we can drop the terms which are constant w.r.t. θ

$$\begin{aligned}\theta_{n+1} &= argmax_\theta \left\{ \sum_z P(z|X,\theta_n) lnP(X|z,\theta) P(z|\theta) \right\} \\ &= argmax_\theta \left\{ \sum_z P(z|X,\theta_n) lnP(X,z|\theta) \right\} \\ &= argmax_\theta \{ E_{Z|X,\theta_n}[lnP(X,z|\theta)] \}\end{aligned}$$

From the above equation, the expectation and maximization steps are apparent:

1. *E-step:* Compute the conditional expectation $E_{Z|X,\theta_n}[lnP(X,z|\theta)]$.

2. *M-step:* Maximize the expectation w.r.t. θ.

2.4.3 Convergence of the EM algorithm

Notice that θ_{n+1} is the estimate for θ which maximizes the log likelihood or the difference $(\delta(\theta|\theta_n))$ of log likelihoods. For $\theta = \theta_n$, we have $\delta(\theta_n|\theta_n) = 0$. Since θ_{n+1} maximizes $\delta(\theta|\theta_n)$, we have $\delta(\theta|\theta_n) \geq \delta(\theta_n|\theta_n) = 0$. Thus the likelihood $L(\theta)$ is non-decreasing during each iteration.

Although an EM iteration increases likelihood of the the observed data, there is no guarantee that the sequence of iterations converges to a maximum likelihood estimator. Depending on initial value of model parameter, the EM algorithm may converge to a local maximum. There are some heuristic methods for escaping a local maximum, e.g., random restart (starting with several different random initial values), or simulated annealing [4, 38].

2.4.4 The Generalized EM algorithm

In the above formulation of the EM algorithm, θ_{n+1} was selected such that it maximizes the likelihood. It may be possible that the maximization step is intractable, i.e., there may be no closed form solution. In this case it's possible to relax the requirement of maximization to simply increasing the likelihood so that $l(\theta_{n+1}|\theta_n) \geq l(\theta_n|\theta_n)$. This approach of simply increasing the likelihood instead of maximizing $l(\theta_{n+1}|\theta_n)$ is known as the Generalized EM algorithm [4].

Chapter 3

Related Work

Over the past decade, a number of Statistical Relational Learning (SRL) frameworks have been developed, which support modeling, inference and/or learning using a combination of logical and statistical methods. These frameworks can be broadly classified as statistical-model-based or logic-based, depending on how their semantics is defined. In the first category are frameworks such as Bayesian Logic Programs (BLPs) [31], Probabilistic Relational Models (PRMs) [20], and Markov Logic Networks (MLNs) [51], where logical relations are used to specify a model compactly. Logic-based SRL frameworks include PRISM [54], Stochastic Logic Programs (SLP) [39], Independent Choice Logic (ICL) [45], and ProbLog [50]. Inference in statistical model based SRL frameworks follows the inference technique of the underlying statistical model. Inference in SRL frameworks such as PRISM [54], Stochastic Logic Programs (SLP) [39], Independent Choice Logic (ICL) [45], and ProbLog [50] is primarily driven by query evaluation over logic programs. We review the SRL frameworks in the following subsections.

3.1 SRL frameworks

3.1.1 Bayesian Logic Programs

A Bayesian Logic Program [31] consists of a set of Bayesian clauses (constructed from Bayesian network structure), and a set of conditional probabilities (constructed from conditional probability tables of Bayesian network). More specifically, a Bayesian clause c has the following form

$$A|A_1, ..., A_n.$$

where $n \geq 0$ and $A, A_1, ..., A_n$ are Bayesian atoms (universally quantified). The differences between a Bayesian clause and a logical clause are: (i) the atoms

$p(t_1, ..., t_m)$ and predicates have an associated set of states or domain $D(p)$, (ii) clauses use "|" instead of ":-" to denote conditional densities. To represent probabilistic model, each Bayesian clause c is associated with a probability mass function $cpt(c)$ encoding $p(head(c)|body(c))$ (where $head(c) = A$ and $body(c) = A_1, ..., A_n$). In general, it represents the conditional probabilities of all ground instances $c\theta$ of c.

Inference and learning in BLPs follow the inference and learning techniques of the underlying statistical model, i.e., Bayesian network. BLP was originally defined over discrete-valued random variable. Continuous BLP [32] extend the base model by using Hybrid Bayesian Networks [40]. To encode continuous variables, continuous BLP associates a probability density function $cpd(c)$ with each clause c. In general, $cpd(c)$ represents the conditional probability density or $p(head(c)|body(c))$. Thus the extension readily encodes Hybrid Bayesian Networks and uses the inference technique of the underlying HBNs.

Example 6. *Let c denote the following Bayesian clause which defines the height of an individual X in terms of the heights of his/her father Y and mother Z.*

$$height(X)|father(Y, X), mother(Z, X), height(Y), height(Z).$$

The domains of the Bayesian predicates $father$, $mother$ and $height$ are $D(father) = D(mother) = \{true, false\}$ and $D(height) = \mathbb{R}$.

The conditional probability density $cpd(c)$ associated to the Bayesian clause c is given in the following table, where v_3, v_4 denote the heights of X's parents.

$father(Y, X) = v_1$	$mother(Z, X) = v_2$	$cpd(c)(h\|v_1, v_2, v_3, v_4)$
$true$	$true$	$\mathcal{N}(h, \frac{1}{2}(v_3 + v_4), 10)$
$true$	$false$	$\mathcal{N}(h, v_3, 10)$
$false$	$true$	$\mathcal{N}(h, v_3, 10)$
$false$	$false$	$\mathcal{N}(h, 60, 10)$

$\mathcal{N}(V, \mu, \sigma^2)$ denotes a Gaussian density function of variable V with mean μ and variance σ^2.

Let $\theta = \{X : eric, Y : brian, Z : ann\}$ be a substitution. Then the ground instance $c\theta$ specifies the following conditional probability density.

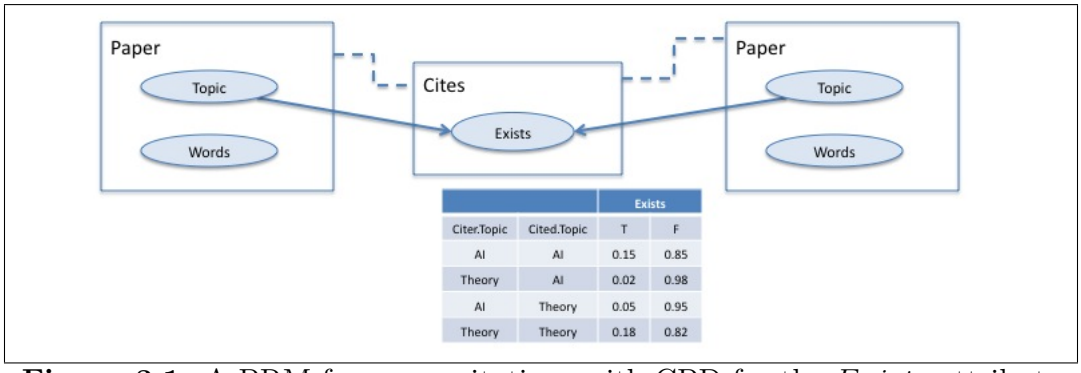
$$P(height(eric)|father(brian, eric), mother(ann, eric), height(brian), height(ann)).$$

□

3.1.2 Probabilistic Relational Models

PRMs [20] encode discrete Bayesian Networks with Relational Models or Schemas. The probabilistic model consists of a dependency structure which is defined by associating with each attribute A a set of parents $Pa(A)$ (similar to Bayesian Networks). Then given a set of parents $Pa(A)$ for attribute A, PRM associates A with a conditional probability distribution (CPD) that specifies $P(A|Pa(A))$.

Example 7. *Figure 3.1 shows a PRM where Paper and Cites represents the classes in the relational schema. Ellipses represent attributes and arrows represent dependency among attributes. For example, attribute 'Exists' depend on the 'Topic' of Citer and Cited papers. The CPD associated with attribute 'Exists' denote the probability* $P(Exists|Citer.Topic, Cited.Topic)$. □



		Exists	
Citer.Topic	Cited.Topic	T	F
AI	AI	0.15	0.85
Theory	AI	0.02	0.98
AI	Theory	0.05	0.95
Theory	Theory	0.18	0.82

Figure 3.1: A PRM for paper citations with CPD for the *Exists* attribute

Similar to BLP, Hybrid PRM [42] also uses the idea of conditional probability density functions (in Hybrid Bayesian Networks [40]) to extend the regular PRM with continuous variables.

Inference and learning in PRMs also follow the inference and learning techniques of the underlying Bayesian network. Exact inference is possible when the ground network is small. However when the ground network is complex, PRM uses approximate inference algorithms, e.g., belief propagation [43].

3.1.3 Markov Logic Networks

An MLN [51] is a set of formulas in first order logic associated with weights. These weights signify how strict the constraints are (e.g., a weight of infinite means first order logic). The semantics of a model in MLN is given in terms of an underlying statistical model obtained by expanding the relations. For instance, a Markov Random Field (MRF) [43] is constructed from an MLN with nodes drawn from the set of all ground instances of predicates in the

first order formulas. The set of (ground) predicates in a ground instance of a formula forms a factor, and the factor potential is obtained from the formula's weight and its truth value. Inference in an MLN is thus reduced to inference over the underlying MRF. Markov Chain Monte Carlo (MCMC) [65] and Gibbs sampling [21] are the most widely used techniques for approximate inference in MRFs.

Example 8. *Consider the following two first-order logic formulas where each formula is associated with a weight.*

$$\forall_x Smokes(x) \Rightarrow Cancer(x) \qquad 1.5$$
$$\forall_x \forall_y Friends(x, y) \Rightarrow (Smokes(x) \Leftrightarrow Smokes(y)) \qquad 1.1$$

The first formula states that smoking causes cancer and the second formula states that if two person are friends, then either both of them smoke or neither does. Figure 3.2 shows the ground Markov Random Field or Markov network

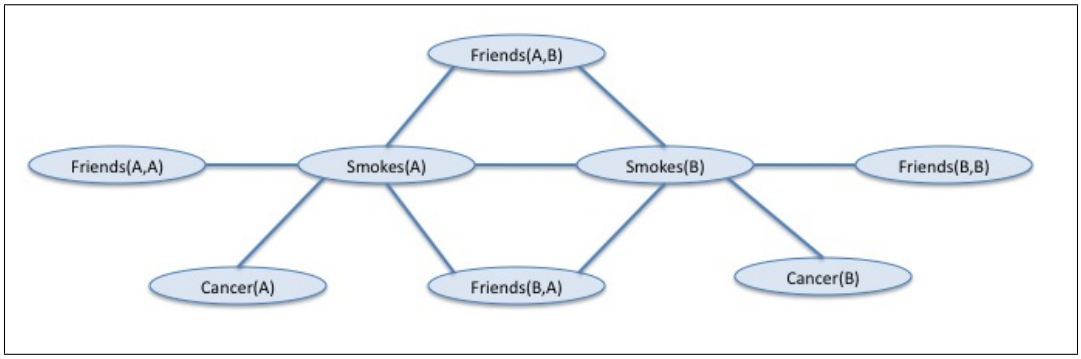


Figure 3.2: Markov Random Field

obtained by applying the above formulas to constants A and B. □

Hybrid MLN [63] allows description of continuous properties and attributes (e.g., the formula $length(x) = 5$ with weight w) deriving MRFs with continuous-valued nodes (e.g., *length(a)* for a grounding of x, with mean 5 and standard deviation $1/\sqrt{2w}$). For doing inference with continuous random variables, [63] describes an approximate inference algorithm based on sampling, search and local optimization.

Learning. Discriminative learning techniques are used for parameter learning in MLNs [37, 57]. Singla et al. [57] described an algorithm for discriminative learning of MLN parameters by combining the voted perceptron with a weighted satisfiability solver. Their experiments showed the advantages of the algorithm compared to generative MLN learning. The discriminative learning

of MLN weights is essentially a gradient descent algorithm. As the weight learning problem can be ill-conditioned, the gradient descent algorithm may be slow to converge. In [37], the authors explored a number of alternatives and report that the best performing is the preconditioned scaled conjugate gradient descent algorithm.

3.1.4 Relational Gaussian Models

Relational Gaussian Models (RGMs) efficiently represent and model dynamic systems in a relational (first-order) fashion [7, 8]. RGMs are composed of three types of parfactor (defined below) models: (i) Relational Transition Models (RTMs), (ii) Relational Observation Models (ROMs), and (iii) Relational Pairwise Models (RPMs). Each parfactor consists of a set of logical variables L, constraints on L, a list of relational atoms X, and a potential function (e.g., Gaussian) on X. Note that relational atoms represent a set a random variables corresponding to the ground substitutions of the logical variables. RTMs model the dependence between relational atoms of the current and next time steps. Similarly, ROMs model the relationship between the observation and state variables. Finally, RPMs capture the dependences between pairs of relational atoms within the same time step.

Note that most probabilistic inference algorithms work on propositional representation level. Lifted inference algorithms [13] carry much of the computations without propositionalizing the first-order model. [8] describes an exact lifted inference algorithm for Kalman Filters which is represented using RGMs. Lifted relational Kalman filter (LRKF) works just like the traditional Kalman filter, i.e., uses two recursive computations: prediction and correction steps. However, LRKFs do not ground the relational atoms when different observations are made.

The idea of relational KF is similar to logical HMMs (LOHMM) [33], which combines ideas from dynamic models (e.g., HMM, KF) and SRLs. However, LOHMM handles only discrete distributions.

Note that RGMs are statistical model based SRL framework, which naturally support Gaussian distributions and can model Kalman filters. In contrast, we propose a general framework which can encode not only Gaussian distribution, but also discrete and Gamma distributions. Thus it permits us to model a large sub-class of Hybrid Bayesian networks and Hybrid models.

3.1.5 Stochastic Logic Programs

Stochastic Logic Programs (SLP) [39] can be thought as the generalization of stochastic context-free grammars. An SLP consist of a set of labelled clauses $p : C$ where p denotes probability and C denote a range-restricted clause. A clause C is range-restricted if and only if every variable occurring in the head

of C also appears in the body of C.

Example 9. *A simple SLP program which encodes a fair coin where the probability of the coin coming up either head or tail is 0.5 is as follows:*

$$0.5 : coin(h)$$
$$0.5 : coin(t)$$

□

Thus clauses of a logic program are annotated with probabilities, which are then used to associate probabilities with the atoms in the Herbrand base (computed according to the SLD-refutation strategy in logic programs). Parameter learning in the PLP languages [48, 49] is typically done by variants of the EM algorithm [15]. SLPs use failure-adjusted maximization (FAM) [12] algorithm to learn parameters. FAM is an instance of the EM algorithm which provides closed-form solution for computing parameter updates in an iterative maximization approach.

3.1.6 Independent Choice Logic

ICL [44] consists of definite clauses and disjoint declarations of the form $disjoint([h_1 : p_1, ..., h_n : p_n])$ that specifies a probability distribution over the hypotheses (i.e., $\{h_1, .., h_n\}$). Any probabilistic knowledge representable in a discrete Bayesian network can be represented in this framework. While the language model itself is restricted (e.g., ICL permits only acyclic clauses), it had declarative distribution semantics. This semantic foundation was later used in other frameworks such as PRISM and ProbLog.

Example 10. *Consider a Bayesian network consisting of two nodes: 'Fire' and 'Smoke', where 'Fire' is the parent node of 'Smoke'. 'Fire' can be represented using the following hypothesis:*

$$\{fire, nofire\}$$

and the probability distribution over these hypothesis is $P(fire) = 0.05$ *and* $P(nofire) = 0.95$. *Now the dependence of 'Smoke' on 'Fire' can be expressed using the following sets of hypothesis and distributions.*

$$\{smokeFromFire : 0.98, nosmokeFromFire : 0.02\}$$
$$\{smokeFromNoFire : 0.01, nosmokeFromNoFire : 0.99\}$$

Now the following two rules can be used to specify when there is smoke

$$smoke \leftarrow fire \wedge smokeFromFire.$$
$$smoke \leftarrow \neg fire \wedge smokeFromNoFire.$$

□

Inference in ICL includes variable elimination, explanation generation and stochastic simulation techniques. ICL uses the belief networks and Bayesian learning approaches to learn the distribution of the hypotheses.

3.1.7 PRISM

In this section, we give a brief overview of PRISM (detailed discussion appears in Chapter 4). PRISM uses explicit random variables (defined using *msw* relation) and a simple inference but restricted procedure. A PRISM program $DB = F \cup R$ consists of a set of definite clauses where F is a set of facts and R is a set of rules. The distribution semantics of PRISM programs is specified by first defining a probability distribution P_F over the facts, and then extending P_F into a distribution over least Herbrand models of the PRISM program DB.

Example 11. *In the following program, direction(D) determines the direction to go by tossing a fair coin. msw(coin, C) defines a random process 'coin' and variable C contains the outcome (head/tail) of the random process.*

```
direction(D) :-
       msw(coin, C),
       (C==head  -> D = left; D = right).
```

□

```
% Sample space of random variables.
values(coin, [head, tail]).

% Probability distribution.
:- set_sw(coin, [0.5, 0.5]).
```

PRISM demands that the set of proofs for an answer are pairwise mutually exclusive, and that the set of random variables used in a single proof are pairwise independent. The inference procedures of LPAD and ProbLog lift these restrictions.

Learning. Sato and Kameya proposed a statistical learning scheme based on the EM algorithm which enables PRISM to learn from examples. The learning algorithm called graphical EM algorithm [55] runs a data structure called support graph which describes the logical relationship between observations and their explanations. The algorithm learns parameters by computing inside

and outside probabilities, and it generalizes to existing EM algorithms (e.g., the Baum-Welch algorithm for HMMs).

BO-EM [29] is a BDD-based parameter learning algorithm for PRISM. One advantage of BDD based learning scheme is that it can relax the exclusiveness restriction. Compared to other BDD-based EM learning algorithms, BO-EM uses shared-BDDs (SBDDs) to efficiently compute probabilities and expectations.

3.1.8 CP-Logic and LPAD

CP-logic. CP-Logic [61] is a logical language to represent probabilistic causal laws. Let ϕ denote a property which causes an event, and the effect of the event makes at most one of the properties ω_i true with probability p_i. Then a CP-event is a statement of the following form

$$(\omega_1 : p_1) \vee \cdots \vee (\omega_n : p_n) \leftarrow \phi.$$

A CP-logic is defined as a CP-theory which is a multiset of CP-events.

Example 12. *The following statement is an example of CP-event which states that bacterial infection can cause either pneumonia (with probability 0.6) or angina (with probability 0.4).*

$$(Pneumonia : 0.6) \vee (Angina : 0.4) \leftarrow Infection.$$

□

The semantics of CP-logic is equivalent to probability distribution over well-founded models of certain logic programs. In fact, [61] shows that it is equivalent to a probabilistic extension of logic programs, called Logic Programs with Annotated Disjunctions (LPAD).

LPAD. A LPAD consists of a set of rules of the following form

$$(h_1 : p_1) \vee \cdots \vee (h_n : p_n) \leftarrow b_1, \ldots, b_m.$$

where h_i and b_j are atoms and literals respectively, and p_i are probabilities such that $\sum_{i=1}^{n} p_i = 1$.

Thus specifications in LPAD [62] resemble those in CP-Logic: probabilistic predicates are specified with disjunctive clauses, i.e., clauses with multiple disjunctive consequents, with a distribution defined over the consequents. LPAD has a distribution semantics, and a proof-based operational semantics similar to that of PRISM.

Example 13. *A Hidden Markov Model with states s_0, s_1 and observations a, b can be modeled by the following LPAD.*

$$\begin{aligned}
(state(s0, s(T)) : 0.6) \vee (state(s1, s(T)) : 0.4) &\leftarrow state(s0, T).\\
(state(s0, s(T)) : 0.3) \vee (state(s1, s(T)) : 0.7) &\leftarrow state(s1, T).\\
(obs(a, T) : 0.6) \vee (obs(b, T) : 0.4) &\leftarrow state(s0, T).\\
(obs(a, T) : 0.2) \vee (obs(b, T) : 0.8) &\leftarrow state(s1, T).\\
state(s0, 0).&
\end{aligned}$$

Here the 1st clause states that if the HMM is in state s_0, then it can either go to state s_1 (with probability 0.4) or stay in state s_0 (with probability 0.6). □

LPAD and ICL are equally expressive and each acyclic LPAD can be transformed into an ICL program. Since LPAD does not have an implemented inference algorithm of its own, queries to acyclic LPAD programs can be solved using the inference techniques of ICL.

3.1.9 ProbLog

ProbLog specifications follow SLP's style, annotating facts in a logic program with probabilities. In contrast to SLP, ProbLog has a distribution semantics and a proof-based operational semantics. More specifically, a ProbLog theory $T = F \cup BK$ consists of a set of labeled facts $F = \{p_1 :: f_1, \ldots, p_n :: f_n\}$ and a set of definite clauses BK. Each fact f_i in F is annotated with a probability p_i.

In contrast to BLP, PRM and MLN, SRL frameworks that are primarily based on logical inference offer limited support for continuous variables. In fact, among such frameworks, only ProbLog has been recently extended with continuous variables. Hybrid ProbLog [24] extends ProbLog by adding a set of continuous probabilistic facts (e.g., $(X_i, \phi_i) :: f_i$, where X_i is a variable appearing in atom f_i, and ϕ_i denotes its Gaussian density function). It adds three predicates namely *below, above, ininterval* to the background Prolog knowledge to process values of continuous facts.

A ProbLog program may use a continuous random variable, but further processing can be based only on testing whether or not the variable's value lies in a given interval. As a consequence, statistical models such as Finite Mixture Models can be encoded in Hybrid ProbLog, but others such as certain classes of Hybrid Bayesian Networks (with continuous child with continuous parents) and Kalman Filters cannot be encoded. The extension to PRISM described in this thesis makes the framework general enough to encode such statistical models.

Example 14. *The following ProbLog program encodes a Gaussian mixture model.*

$$\begin{aligned} 0.6 &:: head. \\ (X, gaussian(2,1)) &:: p_1(X). \\ (X, gaussian(10,5)) &:: p_2(X). \\ tail &: -problog\_not(head). \\ gmix(X) &: -head, p_1(X). \\ gmix(X) &: -tail, p_2(X). \end{aligned}$$

□

ProbLog's inference mechanism employs SLD-resolution to compute the proofs of a query. The proofs are represented using a monotone DNF formula. Then the probability of this formula is computed based on the Binary Decision Diagram (BDD) [6] of the formula. Note that BDD is an efficient graphical representation of a boolean formula.

More recently, [27] introduced a sampling based approach for (approximate) probabilistic inference in a ProbLog-like language. It combines sampling-based inference techniques with forward reasoning. The inference procedure can be used with arbitrary query and evidence variables and can sample from continuous distributions. In contrast, we propose an exact inference mechanism for logic programs with continuous random variables that matches the complexity of specialized inference algorithms for important classes of statistical models (e.g., Kalman filters).

Learning. [25] introduced a least squares optimization approach to learn the parameters of ProbLog. The algorithm, called LeProbLog, computes the probabilities attached to facts by minimizing the error on the training examples as well as on unseen examples. Recently, in [26] the authors introduced an EM-based learning algorithm called CoPrEM for estimating parameters from interpretations (i.e., possible worlds). The algorithm computes binary decision diagrams for each interpretation and uses a dynamic programming approach to estimate parameters.

These techniques enumerate derivations (even when represented as BDDs), and do not readily generalize when continuous random variables are introduced. In this thesis we present a parameter learning algorithm for probabilistic logic programs involving discrete and continuous random variables. One interesting aspect of our algorithm is that in the absence of continuous random variables it specializes to PRISM's learning algorithm.

Discussion. It is not sufficient just to define continuous distributions in SRL frameworks, and use them to denote continuous properties of real world objects. These properties may interact with each other and create an entirely new property. For example, consider an HMLN program where $height(x)$ denotes height of individuals. Now height of individuals in a family are highly correlated, and one may want to create a new random variable which denotes the difference between two heights. But it's not possible in HMLN or ProbLog to define another random variable which is a linear function (e.g., $height(a) - height(b)$) of other random variables, and have Gaussian properties. In addition, inference in temporal models (e.g., HMM, Kalman filters) [53] involve computation of filter (the posterior distribution of current state given all the evidence up to the present) and smoothed (posterior distribution of a past state given all the evidence to date) distributions of random variables. So, the distributions of continuous variables need to be updated as we gather more evidence. HMLN or ProbLog only define a prior distribution of continuous variables, and do not provide any mechanism for updating these distributions or creating an entirely new random variable which is a linear combination of other random variables (e.g., $Y = \overline{A}.\overline{X} + b$). In contrast, we present a framework which permits us to perform the above mentioned tasks.

Chapter 4

PRISM

PRISM programs have Prolog-like syntax (see Figure 4.1). In a PRISM program the `msw` relation ("multi-valued switch") has a special meaning: `msw(X,I,V)` says that `V` is a random variable. More precisely, `V` is the outcome of the `I`-th instance from a family `X` of random processes[1]. The set of variables $\{\mathtt{V}_i \mid \mathtt{msw}(p, i, \mathtt{V}_i)\}$ are i.i.d. for a given random process p, and their distribution is given by p. The `msw` relation provides the mechanism for using random variables, thereby allowing us to weave together statistical and logical aspects of a model into a single program. The distribution parameters of the random variables are specified separately.

The program in Figure 4.1 encodes a Hidden Markov Model (HMM) in PRISM. In the figure, the clause defining `hmm` says that `T` is the `N`-th state if we traverse the HMM starting at an initial state `S` (itself the outcome of

```
hmm(N, T) :-
    msw(init, S),
    hmm_part(0, N, S, T).

hmm_part(I, N, S, T) :-
    I < N, NextI is I+1,
    msw(trans(S), I, NextS),
    obs(NextI, A),
    msw(emit(NextS), NextI, A),
    hmm_part(NextI, N, NextS, T).
hmm_part(I, N, S, T) :- I=N, S=T.
```

Figure 4.1: PRISM program for an HMM

[1]Following PRISM, we often omit the instance number in an `msw` when a program uses only one instance from a family of random processes.

the random process `init`). In `hmm_part(I, N, S, T)`, we may view `S` as the current state with `I` as its index, and `T` as the final state (with `N` as its index). The first clause of `hmm_part` defines the conditions under which we can go from state `S` at position `I` to state `NextS` at position `NextI`, where `NextI` is `I+1`.

1. `msw(trans(S), I, NextS)` means that `NextS` is a random variable whose distribution depends on the value of `S`;

2. `obs(NextI,A)` means that symbol `A` is at the `I+1`-th position in the observation sequence; and

3. `msw(emit(NextS), NextI, A)` means that the observed symbol `A` is a random variable whose distribution depends on `NextS`.

The family of random processes `trans(·)` and `emit(·)` are such that `trans(S)` and `emit(S)` give the distributions of transitions and emissions, respectively, from state `S`.

Query evaluation in PRISM closely follows that for traditional logic programming, with one modification. When the goal selected at a step is of the form `msw(X,I,Y)`, then `Y` is bound to a possible outcome of a random process `X`. The derivation step is associated with the probability of this outcome. If all random processes encountered in a derivation are independent, then the probability of the derivation is the product of probabilities of each step in the derivation. If a set of derivations are pairwise mutually exclusive, the probability of the set is the sum of probabilities of each derivation in the set[2]. Finally, the probability of an answer to a query is computed as the probability of the set of derivations corresponding to that answer. The total probability of all answers to a subgoal is called the *inside* probability of the subgoal.

As an illustration, consider the query `hmm(n,T)` where `n` is a fixed integer, evaluated over program in Figure 4.1. One step of resolution derives goal of the form `msw(init,S), hmm_part(0,n,S,T)`. Now note that there are a number of possible next steps: one for each value in the range of `init`. For instance, if the range of `init` is $\{$`s0`, `s1`$\}$, there are two possible next steps: `hmm_part(0,n,s0,T)` and `hmm_part(0,n,s1,T)`. *Thus in PRISM, derivations are constructed by enumerating the possible outcomes of each random variable.*

Note that evaluation of `hmm(n,T)` over the program in Figure 4.1 obeys the independence and exclusiveness assumptions of PRISM. For instance, process `trans(S)` at the `I`-th step is independent of process `emit(NextS)` at the `I`+1-th step. The only branches in the proofs are due to different outcomes of same random process (e.g., `init` described in the previous paragraph).

[2]The evaluation procedure is defined only when the independence and exclusiveness assumptions hold.

Answers to `hmm(n,T)` and their probabilities give the *filter* distribution of the `n`-th state. In addition to answer probabilities, PRISM can compute the probability that a subgoal G' is encountered in some derivation starting from query G; this is called the *outside* probability of G' w.r.t. G. In the program in Figure 4.1, the outside probability of `hmm_part(i,n,S,T)` for different values of `S` w.r.t. initial query `hmm(n,T)` gives the filter distribution of the `i`-th state. This distribution, multiplied with the inside probability for query `hmm_part(i,n,S,T)` gives the *smoothed* distribution of the `i`-th state.

4.1 Distribution Semantics

The meaning of a PRISM program is given in terms of a *distribution semantics* [54, 55] defined as follows. A PRISM program can be treated as a logic program defined over a set of facts that define the `msw` relation. An instance of the `msw` relation defines one choice of values of all random variables. A PRISM program, given an instance of the `msw` relation, is a non-probabilistic logic program, and its semantics is its least Herbrand model. Thus a PRISM program is associated with a set of least models, one for each `msw` relation instance. A probability distribution is defined over this set of models, based on the probability distribution of the `msw` relation instances. The distribution over models thus obtained is the semantics of a PRISM program. Note that the distribution semantics is defined without regard to any specific computation procedure. For PRISM programs, [55] defines an efficient procedure for computing this semantics based on OLDT resolution, a proof technique with memoization for definite logic programs.

Theoretically, a probability distribution P_F is given to a set of facts F in a PRISM program $DB = F \cup R$ where R is a set of rules. The sample space of F is the set of all Herbrand interpretations i.e., the truth assignments to the ground atoms in F. A sampling of P_F gives a set of true atoms which in turn determines the least Herbrand model of DB. Then P_F is extended it to a probability distribution P_{DB} over the set of possible least models of DB. Thus the semantics of DB with the associated distribution P_{DB} is called the distribution semantics. The distribution semantics is a straightforward generalization of the traditional least model semantics, and can capture semantics of many probabilistic models e.g., Bayesian Networks and HMMs.

4.2 Parameter Learning in PRISM

Given a list of observable atoms $f_1, f_2, ..., f_N$, PRISM finds the explanations of the observations/facts, and then uses those explanations to learn the parameters of the distributions. Let F be the random variable representing the observations, and E be the random variable representing the explanations.

The probability of a goal is the summation of the probabilities of its explanations.

$$P(F = f) = \sum_{k=1}^{|E_f|} P(E = e_k)$$

where E_f denotes the set of explanations of f.

The probability of an explanation is the product of the probabilities of all the random variables in that explanation.

$$P(E = e) = \prod_{r} P(X_r = v)$$

In this parameter learning setting, the facts or goals are viewed as observed data, and explanations are viewed as hidden data. EM algorithm gives a nice framework for learning the parameters of the distributions. At each iteration, it first calculates the value of Q function introduced below using current parameter value Θ^{old} (**E-step**):

$$\begin{aligned} Q(\Theta, \Theta^{old}) &= \sum_{i=1}^{N} \sum_{k=1}^{|E_{f_i}|} P(E = e_k | F = f_i, \Theta^{old}) lnP(E = e_k, F = f_i | \Theta) \\ &= \sum_{i=1}^{N} \sum_{k=1}^{|E_{f_i}|} \frac{P(E = e_k, F = f_i | \Theta^{old})}{P(F = f_i | \Theta^{old})} lnP(E = e_k, F = f_i | \Theta) \end{aligned}$$

Now

$$P(E = e_k, F = f_i | \Theta) = \begin{cases} 0 \text{ if } e_k \notin E_{f_i} \\ P(E = e_k | \Theta) \text{ if } e_k \in E_{f_i} \end{cases}$$

So,

$$\begin{aligned} Q(\Theta, \Theta^{old}) &= \sum_{i=1}^{N} \sum_{k=1}^{|E_{f_i}|} \frac{P(E = e_k | \Theta^{old})}{P(F = f_i | \Theta^{old})} lnP(E = e_k | \Theta) \\ &= \sum_{i=1}^{N} \frac{1}{P(F = f_i | \Theta^{old})} \sum_{k=1}^{|E_{f_i}|} P(E = e_k | \Theta^{old}) ln(\alpha P(X_r = v | \Theta)) \end{aligned}$$

Here, $P(E = e_k | \Theta) = \alpha P(X_r = v | \Theta)$ where α contains the probabilities of all the random variables in e_k except X_r.

Next it maximizes $Q(\Theta, \Theta^{old})$ as a function of Θ, and updates the parameter

values (**M-step**):

$$\Theta^{new} = argmax_{\Theta} Q(\Theta, \Theta^{old})$$

4.2.1 Specialization of E and M Steps for Discrete Distribution

Let X_r denote the discrete random variable and $\theta_{r,v}$ denote the probability of X_r taking a specific discrete value v. We use $C_{r,v,k}$ to denote the count of random variable X_r taking value v in explanation e_k. Thus the probability of an explanation can be written as

$$\begin{aligned} P(E = e_k|\Theta) &= \alpha P(X_r = v|\Theta) \\ &= \alpha \theta_{r,v}^{C_{r,v,k}} \end{aligned}$$

The E and M-steps of the learning algorithm are as follows:

E-step:

$$\begin{aligned} Q(\Theta, \Theta^{old}) &= \sum_{i=1}^{N} \frac{1}{P(F = f_i|\theta^{old})} \sum_{k=1}^{|E_{f_i}|} P(E = e_k|\Theta^{old}) ln(\alpha P(X_r = v|\Theta)) \\ &= \sum_{i=1}^{N} \frac{1}{P(F = f_i|\Theta^{old})} \sum_{k=1}^{|E_{f_i}|} P(E = e_k|\Theta^{old}) ln(\alpha \theta_{r,v}^{C_{r,v,k}}) \\ &= \sum_{i=1}^{N} \frac{1}{P(F = f_i|\Theta^{old})} \sum_{k=1}^{|E_{f_i}|} P(E = e_k|\Theta^{old}) ln(\alpha) \\ &+ \sum_{i=1}^{N} \frac{1}{P(F = f_i|\Theta^{old})} \sum_{k=1}^{|E_{f_i}|} P(E = e_k|\Theta^{old}) C_{r,v,k} ln(\theta_{r,v}) \\ &= \beta + EC[r, v] ln(\theta_{r,v}) \end{aligned}$$

where β is a constant which does not involve the random variable X_r, and $EC[r, v]$ is the expected count of value v of the random variable X_r,

$$EC[r, v] = \sum_{i=1}^{N} \frac{1}{P(F = f_i|\theta^{old})} \sum_{k=1}^{|E_{f_i}|} P(E = e_k|\theta^{old}) C_{r,v,k} \tag{4.1}$$

M-step:
At the M-step, we compute $\theta_{r,v}$ which maximizes $Q(\theta, \theta^{old})$,

$$\theta_{r,v} = \frac{EC[r,v]}{\sum_{v' \in values} EC[r,v']} \tag{4.2}$$

The algorithm starts with an initial set of parameters Θ. After each iteration it computes the log-likelihood of the observed data ($\sum_{i=1}^{N} \ln P(f_i)$) and repeats the above steps until the convergence of the log-likelihood.

4.2.2 Graphical EM algorithm

Although the EM algorithm presented above is simple and correctly calculates the MLE of Θ, the computation of $EC[r,v]$ and $P(F = f_i|\theta)$ may suffer from combinatorial explosion of explanations, i.e., the size of explanations set grows exponentially in the complexity of the model. Note that explanations share common goals, i.e., they share common partial paths in the derivation tree for a given top goal. Thus it is possible to eliminate redundant computation by saving results of common goals. Motived by this observation, Sato and Kameya proposed an efficient framework for EM learning of PRISM programs called the graphical EM (g-EM) algorithm [55], which combines tabled search (memoization) technique for logic programs and a dynamic programming-based EM algorithm [60, 64, 66].

The efficiency of the g-EM algorithm is achieved through the introduction of a data-structure called *support graph* which describes the logical relationship between observations and their explanations. The support graph for a goal f_i is a graphical representation of the hierarchical system of tabled explanations (Figure 4.2). It consists of disconnected subgraphs, each of them is labeled with a corresponding tabled atom (e.g., α_j in Figure 4.2). Thus data sharing is achieved through the use of tabling and support graph.

Complexity of the g-EM algorithm: Let ξ_{num} denote the maximum number of tabled explanations in a support graph for a goal f_i and $\xi_{maxsize}$ denote the maximum size of a tabled explanations for the goal f_i.

The time complexity of the graphical EM algorithm per iteration is linear in the total size of support graphs, $O(\xi_{num}\xi_{maxsize}N)$, which also coincides with the space complexity as g-EM algorithm runs on support graphs.

In general, the total time complexity of the learning algorithm also includes the time to construct the support graph. Thus the actual total learning time is

$$\text{OLDT time} \ + \ (\text{the number of iterations}) \ * O(\xi_{num}\xi_{maxsize}N)$$

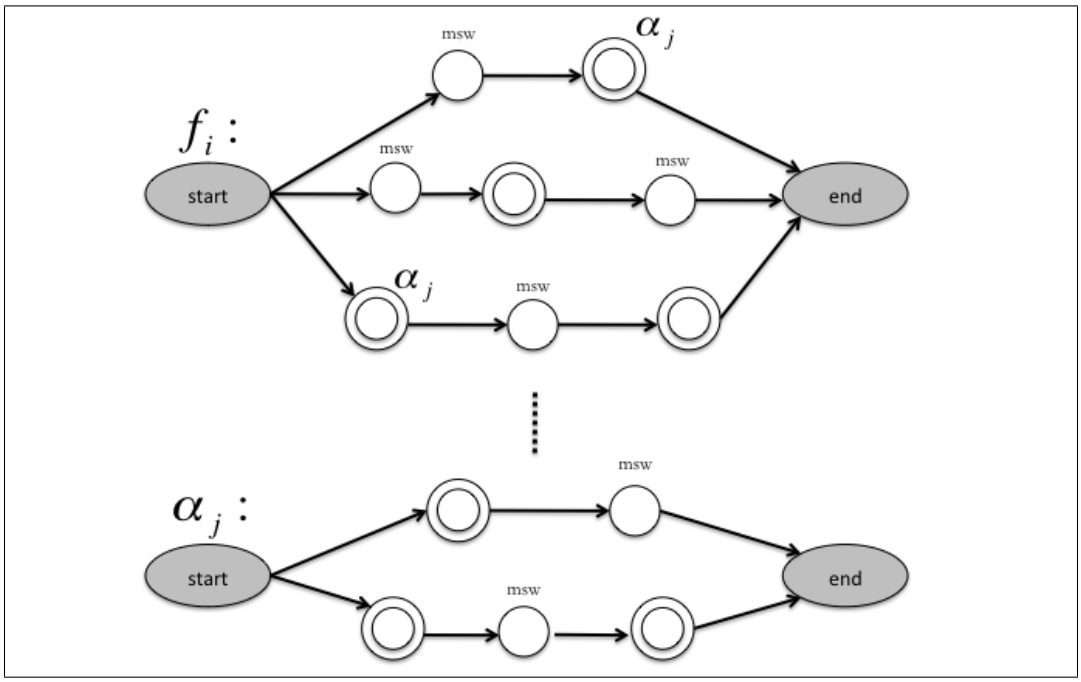


Figure 4.2: Support graph. A double circled node denotes a tabled atom.

where OLDT time denotes time to construct all support graphs.

The complexity of g-EM algorithm specializes to the complexity of several popular EM algorithms, e.g., the Baum-Welch algorithm for HMMs and the Inside-Outside algorithm for PCFGs.

Chapter 5

Extended PRISM

Support for continuous variables is added by modifying PRISM's language in two ways. We use the `msw` relation to sample from discrete as well as continuous distributions. In PRISM, a special relation called `values` is used to specify the ranges of values of random variables; the probability mass functions are specified using `set_sw` directives. In our extension, we extend the `set_sw` directives to specify probability density functions as well. For instance, `set_sw(r, norm(Mu,Var))` specifies that outcomes of random processes `r` have Gaussian distribution with mean `Mu` and variance `Var`[1]. Parameterized families of random processes may be specified, as long as the parameters are discrete-valued. For instance, `set_sw(w(M), norm(Mu,Var))` specifies a family of random processes, with one for each value of `M`. As in PRISM, `set_sw` directives may be specified programmatically; for instance, in the specification of `w(M)`, the distribution parameters may be computed as functions of `M`.

Additionally, we extend PRISM programs with linear equality constraints over reals. Without loss of generality, we assume that constraints are written as linear equalities of the form $Y = a_1 * X_1 + \ldots + a_n * X_n + b$ where a_i and b are all floating-point constants. The use of constraints enables us to encode Hybrid Bayesian Networks and Kalman Filters as extended PRISM programs. In the following, we use *Constr* to denote a set (conjunction) of linear equality constraints. We also denote by $\overline{X}$ a vector of variables and/or values, explicitly specifying the size only when it is not clear from the context. This permits us to write linear equality constraints compactly (e.g., $Y = \overline{a} \cdot \overline{X} + b$).

Encoding of Kalman Filter specifications in the extended PRISM uses linear constraints and closely follows the structure of the HMM specification, and is shown in Section 6.4.

[1]The technical development in this thesis considers only univariate Gaussian variables; see Section 6.5 on a discussion on how multivariate Gaussian as well as other continuous distributions are handled.

Example 15 (Finite Mixture Model)**.** *In the following PRISM program, which encodes a finite mixture model [38],* `msw(m, M)` *chooses one distribution from a finite set of continuous distributions,* `msw(w(M), X)` *samples* `X` *from the chosen distribution.*

```
fmix(X) :- msw(m, M),
           msw(w(M), X).

% Ranges of RVs
values(m, [a,b]).
values(w(M), real).

% PDFs and PMFs
:- set_sw(m, [0.3, 0.7]),
   set_sw(w(a), norm(1.0, 0.2)),
   set_sw(w(b), norm(1.05, 0.1)).
```

□

5.1 Encoding Hybrid Bayesian Networks in Extended PRISM

In this section, we present encoding of different types of hybrid Bayesian networks (discussed in Section 2.3) in our language.

Consider a hybrid Bayesian network with two nodes (X and Y), where X, Y denote parent and child nodes respectively. In the following examples, we encode different types of hybrid Bayesian networks depending on the type (discrete/continuous) of X and Y.

Example 16 (Discrete child-discrete parent hybrid Bayesian network)**.** *In the following program, $msw(x, X)$ defines the distribution of X and $msw(y(X), Y)$ defines the conditional probability of Y given its parent X. Here, both X and Y takes discrete values $(0, 1)$.*

```
hbn(X, Y) :- msw(x, X),
             msw(y(X), Y).

% Ranges of RVs
values(x, [0, 1]).
values(y(0), [0, 1]).
values(y(1), [0, 1]).

% PMFs:
:- set_sw(x, [0.4, 0.6]),
   set_sw(y(0), [0.5, 0.5]),
   set_sw(y(1), [0.8, 0.2]).
```

□

Note that multiple parent nodes can be encoded by defining the distribution of the child node depending on the valuations of its parent nodes. In the following example, we encode a continuous child-discrete parent hybrid Bayesian network.

Example 17 (Continuous child-discrete parent hybrid Bayesian network)**.** *The main difference between this example and the previous example is that here Y is a continuous random variable, and it takes values from two Gaussian distributions depending on the value of its discrete parent X.*

```
hbn(X, Y) :- msw(x, X),
             msw(y(X), Y).

% Ranges of RVs
values(x, [0, 1]).
values(y(X), real).

% PDFs and PMFs:
:- set_sw(x, [0.4, 0.6]),
   set_sw(y(0), norm(1.0, 0.5)),
   set_sw(y(1), norm(2.0, 0.5)).
```

□

Example 18 (Continuous child-continuous parent hybrid Bayesian network)**.** *In this example, we encode a continuous child-continuous parent hybrid Bayesian network. Here both X and Y are Gaussian random variables. The parent-child relationship is encoded using a linear equality constraint where Y is a linear combination of its parent X and a Gaussian noise E.*

```
hbn(X, Y) :- msw(x, X),
             msw(e, E),
             Y = X + E.

% Ranges of RVs
values(x, real).
values(e, real).

% PDFs:
:- set_sw(x, norm(1.0, 0.5)),
   set_sw(e, norm(0.0, 0.1)).
```

□

5.2 Distribution Semantics

In this section, we present the distribution semantics of the extended PRISM programs. Recall that the core of Sato's distribution semantics lies in defining a probability distribution P_F over the facts, and then extending P_F into a

distribution over least Herbrand models of the logic program. Sato shows that, given a fixed enumeration $f_1, f_2, ...$ of facts in F, P_F can be constructed from a series of finite distribution $P_F^n(f_1 = x_1, ..., f_n = x_n)$ (where $x_i = 0/1$) provided that it satisfies the compatibility condition, which is

$$P_F^n(f_1 = x_1, ..., f_n = x_n) = \sum_{x_{n+1}} P_F^{n+1}(f_1 = x_1, ..., f_{n+1} = x_{n+1}).$$

[27] introduced a sampling based approach for (approximate) probabilistic inference in a ProbLog-like language, where they define a distribution semantics for programs with continuous variables. The authors claim that the set of probabilistic facts is enumerable, and then construct (closely following PRISM's mechanism) the distribution semantics. But the set of probabilistic facts is infinite with continuous random variable (Gaussian/Gamma). So PRISM's approach (e.g., enumerable set of facts, compatibility condition) does not generalize naturally for continuous random variables as the Herbrand universe is infinite. Thus the formulation of the distribution semantics of the extended PRISM programs is an important problem. In the next subsections, we first describe some preliminaries on probability space and then define the distribution semantics of the extended PRISM programs.

5.2.1 Preliminaries

Probability space: A probability space consists of three components $(\Omega, \mathcal{F}, P)$ where Ω denotes the sample space, $\mathcal{F}$ denotes the event space which is a σ-algebra over Ω, and P is a probability measure on $\mathcal{F}$.

A σ-algebra over a set Ω is a nonempty collection $\mathcal{F}$ of subsets of Ω that is closed under complementation and countable unions of its members.

Example 19. *Let the sample space is $\Omega = \{a_1, a_2\}$. Then the power set of Ω is a σ-algebra, $\mathcal{F} = \{\{\}, \{a_1\}, \{a_2\}, \{a_1, a_2\}\}$. Notice that $\mathcal{F}$ is closed under complementation and countable unions of its members.* □

P is a probability measure with total measure one ($P(\Omega) = 1$). A function P from $\mathcal{F}$ to the real numbers is called a measure if it satisfies the following properties.

- Probabilities are non-negative: $P(E) \geq 0$ for all events $E \in \mathcal{F}$.
- Countable additivity: For all countable collections $\{E_i\}$ of pairwise disjoint sets in $\mathcal{F}$.
  $P(\cup E_i) = \sum_i P(E_i)$
- Probability of empty set is zero: $P(\emptyset) = 0$

Example 20. *Let's consider the sample space and σ-algbera of the previous example, and let P is a probability measure over $\mathcal{F}$ which is defined as follows: $P(\{\}) = 0.0$, $P(\{a_1\}) = 0.7$, $P(\{a_2\}) = 0.3$ and $P(\{a_1, a_2\}) = 1.0$. Notice that P satisfies the above mentioned properties of probability measure.* □

Probability space of continuous random variables: The sample space Ω of a continuous random variable is the set of real numbers $\mathbb{R}$. The event space $\mathcal{F}$ is a *Borel* algebra which is the smallest σ-algebra on $\mathbb{R}$ containing all the intervals, and P is a *Lebesgue* measure on $\mathcal{F}$. A *Lebesgue* measure assigns measure (e.g., length, area, volume) to subsets of n-dimensional Euclidean space. We will use $\mathcal{B}(\mathbb{R})$ to denote the Borel set of $\mathbb{R}$.

Example 21. *An element of the Borel algebra is called a Borel set. Any interval on the real line $\mathbb{R}$ is a Borel set, e.g., the line segment [0,0.7] is a Borel set and one of its Lebesgue measure is the length (0.7) of the line segment.* □

5.2.2 Distribution Semantics of Extended PRISM Programs

Let $PL = B \cup R$ be a probabilistic logic program where B is a set of probabilistic facts and R is a set of rules and constraints. We construct a probability space for PL in the following two steps. First we introduce a probability space for B. Next we extend it to a probability space for PL using the least model semantics.

Basic Distribution P_B: We construct a probability space $(\Omega_B, \mathcal{F}_B, P_B)$ for B, where Ω_B is the sample space, $\mathcal{F}_B$ is a σ-algebra over Ω_B, and P_B is a probability measure on $\mathcal{F}_B$.

Definition 1 (Sample space of base distribution)**.** *Let $r_1, r_2, ..., r_n$ be a set of random processes in B, and $V_r = \{msw(r, v) | v \in values(r)\}$ where $values(r)$ denotes the set of values the random process r takes. The sample space Ω_B of B is defined as the enumeration of all the random variable valuations, i.e.,*

$$\Omega_B = \{\langle a_1, a_2, ..., a_n \rangle | a_i \in V_{r_i}\}.$$

Note that $values(r)$ is a finite set for discrete random variables, and for continuous random variables it is the set of real numbers $\mathbb{R}$. In the above equation, msw atom a_i represents the i^{th} random process and its value. Since the random variables are ordered in a sequence, the elements in the sample space are represented using vectors. For simplicity, we omit the instance numbers (i) in $msw(r, i, v)$ in this section.

Definition 2 (Event space of base distribution)**.** *The event space $\mathcal{F}_B$ is defined as a σ-algebra over Ω_B. An event in $\mathcal{F}_B$ is represented as*

$$\{\langle a_1, a_2, ..., a_n\rangle | a_j = V_{r_j} \text{ when } r_j \text{ is discrete};\\ \text{and } a_j = msw(r_j, i_j) \text{ when } r_j \text{ is continuous.}\}$$

Here i_j represents an element of Borel-algebra.

Finally, P_B is a probability measure on $\mathcal{F}_B$. The choice of P_B is free as long as it satisfies the properties of probability measure and maintains the independence assumption, i.e., random processes (r_i) are independent of each other. Note that the probability distributions of the random processes are specified using the *set_sw* predicate.

Following examples illustrate the basic probability space $(\Omega_B, \mathcal{F}_B, P_B)$.

Example 22. *Consider the following extended PRISM program which has a single discrete random variable.*

```
q(V) :- msw(a, V).

values(a, [0,1]).

:- set_sw(a, [0.3, 0.7]).
```

The sample space Ω_B is the enumeration of all the random variable valuations, i.e.,

$$\Omega_B = \{\langle msw(a, 0)\rangle, \langle msw(a, 1)\rangle\}.$$

The event space $\mathcal{F}_B$ is a σ-algebra over Ω_B, and P_B is a probability measure on $\mathcal{F}_B$. Probabilities of some elements of the event space:
$P_B(\{\langle msw(a, 0)\rangle\}) = 0.3$, $P_B(\{\langle msw(a, 1)\rangle\}) = 0.7$. □

Example 23. *Consider the following extended PRISM program which uses continuous random variables.*

```
r(X, Y) :- msw(b, X),
           Y = X + 1.

values(b, real).

:- set_sw(b, norm(0,1)).
```

The sample space Ω_B of B is

$$\Omega_B = \{\langle msw(b, v)\rangle | v \in \mathbb{R}\}.$$

The event space $\mathcal{F}_B$ is a Borel-algebra over Ω_B. Each event in $\mathcal{F}_B$ contains elements of the following form

$$\{\langle msw(b, i_v)\rangle | i_v \in \mathcal{B}(\mathbb{R})\}.$$

Here i_v represents an element of $\mathcal{B}(\mathbb{R})$.

Let $e_1 = \{\langle msw(b, [0, 0.3])\rangle\}$ be an element of $\mathcal{F}_B$. The probability measure of e_1 is

$$P_B(e_1) = \int_0^{0.3} \mathcal{N}(0, 1)dx$$

where $\mathcal{N}(0, 1)$ denotes the distribution of random process b (defined using the set_sw *predicate).*

Let $e_2 = \{\langle msw(b, [0, 0.3])\rangle, \langle msw(b, [0.7, 0.9])\rangle\}$ be another element of $\mathcal{F}_B$. The probability measure of e_2 is

$$P_B(e_2) = \int_0^{0.3} \mathcal{N}(0, 1)dx + \int_{0.7}^{0.9} \mathcal{N}(0, 1)dx$$

Note that the event space may contain elements of the form $\{\langle msw(b, [0, 0.3]$, $[0.7, 0.9])\rangle\}$, and the probability of this event is same as that of e_2. □

Next we construct a probability space for PL.

Extending P_B to P_{PL}: We construct a probability space $(\Omega_{PL}, \mathcal{F}_{PL}, P_{PL})$ for PL where Ω_{PL} is the sample space, $\mathcal{F}_{PL}$ is a σ-algebra over Ω_{PL}, and P_{PL} is a probability measure on $\mathcal{F}_{PL}$.

As PL defines a set of rules R, in addition to the random variable valuations, Ω_{PL} also contains the atoms entailed by the rules of the program. For example, $q(0), q(1)$ in Example 22, and $r(x, y)|x, y \in \mathbb{R}$ in Example 23.

Since the predicates define relations, sample space of PL contains msw atoms as well as all possible relations. For example, the sample space Ω_{PL} of the program in Example 22 is

$$\begin{aligned}\Omega_{PL} = \{&\langle msw(a, 0), \{\}\rangle, \langle msw(a, 0), \{q(0)\}\rangle, \langle msw(a, 0), \{q(1)\}\rangle,\\ &\langle msw(a, 0), \{q(0), q(1)\}\rangle, \langle msw(a, 1), \{\}\rangle, \langle msw(a, 1), \{q(0)\}\rangle,\\ &\langle msw(a, 1), \{q(1)\}\rangle, \langle msw(a, 1), \{q(0), q(1)\}\rangle\}.\end{aligned}$$

Similarly, for the program in Example 23, $r(x, y)$ defines relations over reals (i.e., elements of Borel algebra). Thus the sample space Ω_{PL} is

$$\Omega_{PL} = \{\langle msw(b, v), r(x, y)\rangle | v \in \mathbb{R}; x, y \in \mathcal{B}(\mathbb{R})\}.$$

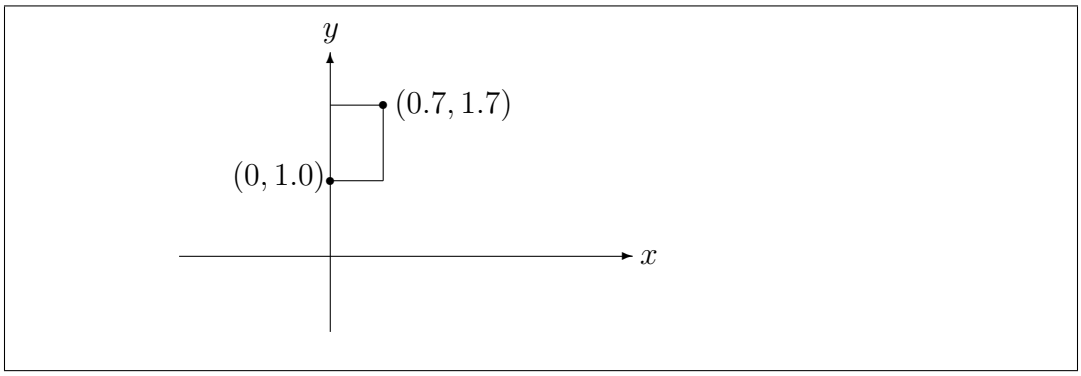


Figure 5.1: Rectangle with corner points $(0, 1.0)$ and $(0.7, 1.7)$

As illustrated in the above discussion, the sample space of the extended program can be defined as follows.

Definition 3 (Sample space of extended distribution)**.** *Let S_b denote the set of all random variable valuations, i.e.,*

$$S_b = \{\langle a_1, a_2, ..., a_m\rangle | a_i \in V_{r_i}\}$$

and S_p denote the following set for an n-ary predicate $p(X_1, X_2, ..., X_n)$

$$S_p = \{\langle p(x_1, x_2, ..., x_n)\rangle | x_j \in values(X_j) \text{ when } X_j \text{ is discrete};\\ \text{and } x_j \in \mathcal{B}(\mathbb{R}) \text{ when } X_j \text{ is continuous}\}.$$

Then the sample space Ω_{PL} of PL is defined as

$$\Omega_{PL} = \{\langle s_b, s_{p_1}, s_{p_2}, ...\rangle | s_b \in S_b \text{ and } s_{p_i} \in S_{p_i}\}$$

where $p_1, p_2, ...$ are predicates in PL.

Now we construct the event space of PL as follows. For programs with only discrete random variables, each event is simply a subset of Ω_{PL}. Consider an event containing continuous values. An event containing the relation $r([0, 0.7], [1.0, 1.7])$ specifies a rectangular region (bottom-left and top-right corner points $(0, 1.0)$ and $(0.7, 1.7)$ respectively) in a 2-dimensional space (Figure 5.1). Thus each event in Example 23 has the following form

$$\{\langle msw(b, i_v), \cup_{j\geq 0, j\in\mathbb{N}}\{r(x, y) | x \in [u_{4j}, u_{4j+1}], y \in [u_{4j+2}, u_{4j+3}]\}\rangle\\ | i_v \in \mathcal{B}(\mathbb{R}) \text{ and } \forall_{l\geq 0} u_l \in z_l \text{ and } z_0, z_1, ... \in \mathcal{B}(\mathbb{R}^\omega)\}.$$

In general, let $p(\overline{x})$ contain a set of intervals of the form $[p_{1b}, p_{1e}], [p_{2b}, p_{2e}], ...$, where p_{ib} and p_{ie} denote the start and end point of the i^{th} interval respectively.

Next we order the interval points as follows: $p_{1b}, p_{2b}, ..., p_{nb}, p_{1e}, p_{2e}...$, and so on. This set of points represents an element of $\mathbb{R}^\omega$. Now each element of $\mathcal{F}_{PL}$ is constructed from $\mathcal{B}(\mathbb{R}^\omega)$ in conjunction with the σ-algebra of the discrete valued predicates and msw.

We formally define the event space of the extended program as follows.

Definition 4 (Event space of extended distribution)**.** *Let E_b denote the following set*

$$E_b = \{\langle a_1, a_2, ..., a_k\rangle | a_j = V_{r_j} \text{ when } r_j \text{ is discrete;} \\ \text{and } a_j = msw(r_j, i_j) \text{ when } r_j \text{ is continuous.}\}$$

and E_p denote the following set for an n-ary predicate $p(X_1, X_2, ..., X_n)$

$$E_p = \cup_{j\geq 0}\{p(x_1, x_2, ..., x_n) | x_i \in values(X_i) \text{ when } X_i \text{ is discrete;} \\ x_i \in [u_j, u_{j+1}] | \forall_{l\geq 0} u_l \in z_l \text{ and } z_0, z_1, ... \in \mathcal{B}(\mathbb{R}^\omega) \text{ when } X_i \text{ is continuous}\}.$$

Then an event $e \in \mathcal{F}_{PL}$ is defined as

$$e = \{\langle e_b, e_{p_1}, e_{p_2}, ..., \rangle | e_b \in E_b \text{ and } e_{p_i} \in E_{p_i}\}$$

where $p_1, p_2, ...$ are predicates in PL.

Finally, we define a probability measure P_{PL} for events $e_i \in \mathcal{F}_{PL}$. Note that each event e_i is a set of vectors. Now if e_i contains the models entailed by the probabilistic atoms of e_i, then probability of the event is computed based on the probabilistic atoms.

For example, the probability of an event $\{\langle msw(a, 0), \{q(0)\}\rangle\}$ of the program in Example 22, is simply the probability of $\{\langle msw(a, 0)\rangle\}$ as the least model entailed by the base fact $msw(a, 0)$ is a subset of $\{msw(a, 0), q(0)\}$. Thus

$$P_{PL}(\{\langle msw(a, 0), q(0)\rangle\}) = P_B(\{\langle msw(a, 0)\rangle\}) = 0.3$$

Similarly, the probability of $\{\langle msw(a, 0), \{q(0), q(1)\}\rangle\}$ is 0.3, as the least model entailed by the base fact $msw(a, 0)$ is a subset of this event.

On the other hand, the probability of $\{\langle msw(a, 0), \{q(1)\}\rangle\}$ is 0 as $msw(a, 0)$ does not entail $q(1)$.

Similarly, the probability of an event $\{\langle msw(b, [0, 0.3]), r([0, 0.3], [1.0, 1.3])\rangle\}$ of the program in Example 23, is the probability of $\{\langle msw(b, [0, 0.3])\rangle\}$ as

$r([0, 0.3], [1.0, 1.3])$ is entailed by the base fact $msw(b, [0, 0.3])$.

$$P_{PL}(\{\langle msw(b, [0, 0.3]), r([0, 0.3], [1.0, 1.3])\rangle\}) = P_B(\{\langle msw(b, [0, 0.3])\rangle\}) = \int_0^{0.3} \mathcal{N}(0, 1)dx.$$

Similarly, the probability of $\{\langle msw(b, [0, 0.3]), r([0, 0.4], [1.0, 1.3])\rangle\}$ is same as the above probability.

Let κ be an event in $\mathcal{F}_{PL}$, and $p(\kappa)$ denote the set of non-probabilistic or predicate atoms of κ. Now we can construct an event $\nu \in \mathcal{F}_B$ such that it contains only the probabilistic atoms of κ. Let $M_{PL}(\nu)$ denote the least Herbrand model starting with ν. We use the notion of least model or fixpoint semantics of constraint logic programs [30]. The fixpoint semantics is based on the "immediate consequence operator" or T_p [2] (defined below)

$$T_p(I) = \{A | \exists A \leftarrow B_1, ..., B_k \in PL(0 \leq k) \text{ such that } \{B_1, ..., B_k\} \subseteq I\}$$

where I is a set of ground atoms.

Now we define the probability measure P_{PL} of an event κ as follows.

Definition 5 (Probability measure of extended distribution)**.** *Let $\kappa \in \mathcal{F}_{PL}$ and $\nu \in \mathcal{F}_B$ contains only the probabilistic atoms of κ. Then $P_{PL}(\kappa)$ is defined as*

$$P_{PL}(\kappa) = \begin{cases} P_B(\nu) & \text{if } M_{PL}(\nu) \subseteq p(\kappa) \\ 0 & \text{otherwise.} \end{cases}$$

Thus there exists a probability measure P_{PL} over $\mathcal{F}_{PL}$ which is an extension of P_B.

Example 24. *For the Finite Mixture Model program in Example 34, the sample space Ω_B of the base distribution is*

$$\Omega_B = \{\langle msw(m, v), msw(w(v), x)\rangle | v \in \{a, b\}, x \in \mathbb{R}\}$$

and the sample space Ω_{PL} of PL is

$$\Omega_{PL} = \{\langle msw(m, v), msw(w(v), x), fmix(z)\rangle | v \in \{a, b\}, x \in \mathbb{R}, z \in \mathcal{B}(\mathbb{R})\}.$$

Next the event space $\mathcal{F}_B$ is a σ-algebra over Ω_B. Each event in $\mathcal{F}_B$ contains elements of the following form

$$\{\langle msw(m, v), msw(w(v), i_x)\rangle | v \in \{a, b\}, i_x \in \mathcal{B}(\mathbb{R})\}$$

and each event in $\mathcal{F}_{PL}$ *contains elements of the following form*

$$\{\langle msw(m,v), msw(w(v), i_x), \cup_{j\geq 0}\{fmix(z)|z \in [u_{2j}, u_{2j+1}]\}\rangle \\ |v \in \{a,b\}, i_x \in \mathcal{B}(\mathbb{R}) \ and \ \forall_{l\geq 0} u_l \in z_l; z_0, z_1, ... \in \mathcal{B}(\mathbb{R}^\omega)\}.$$

The probability of $\{\langle msw(m,a), msw(w(a),[1.0,1.1]), fmix([1.0,1.1])\rangle\}$ *is the probability of* $\{\langle msw(m,a), msw(w(a),[1.0,1.1])\rangle\}$

$$\begin{aligned} &P_{PL}(\{\langle msw(m,a), msw(w(a),[1.0,1.1]), fmix([1.0,1.1])\rangle\}) \\ &\quad = P_B(\{\langle msw(m,a), msw(w(a),[1.0,1.1])\rangle\}) \\ &\quad = 0.3\int_{1.0}^{1.1} \mathcal{N}(1.0,0.2)dw \end{aligned}$$

□

Chapter 6

Inference

We present an inference algorithm to reason over extended PRISM programs with Gaussian random variables (in addition to discrete-valued ones), and linear equality constraints over values of these continuous random variables. Instead of relying on enumerating sets of explanations for a query answer, we present a *symbolic derivation* procedure that uses constraints and represents sets of explanations without enumeration. In this chapter, we present our inference algorithm in detail along with correctness and complexity analysis. We also provide an illustrative example on Kalman filter.

6.1 Inference Algorithm

Recall that PRISM's inference explicitly enumerates outcomes of random variables in derivations. For example, Figure 6.1 shows the derivation for goal $hbn(X, Y)$ in Example 16. The derivation takes different paths for different outcomes of the random variables X and Y.

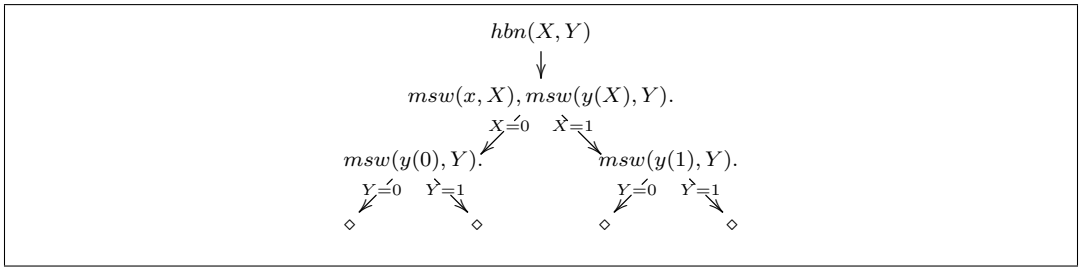


Figure 6.1: Derivation for goal `hbn(X, Y)`

The key to inference in the presence of continuous random variables is avoiding enumeration by representing the derivations and their attributes symbolically. For example, Figure 6.2 shows the symbolic derivation for goal $hbn(X, Y)$ in Example 16. Notice that X and Y are represented symbolically

and are not bounded to any outcome.

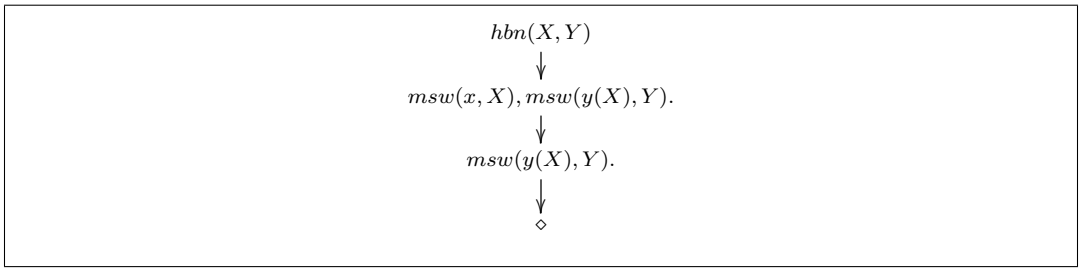


Figure 6.2: Symbolic derivation for goal `hbn(X, Y)`

A single step in the construction of a symbolic derivation is defined below.

Definition 6 (Symbolic Derivation)**.** *A goal G* directly derives *goal G', denoted $G \to G'$, if:*

PCR: *$G = q_1(\overline{X_1}), G_1$, and there exists a clause in the program, $q_1(\overline{Y}) :- r_1(\overline{Y_1}), r_2(\overline{Y_2}), \ldots, r_m(\overline{Y_m})$, such that $\theta = \mathrm{mgu}(q_1(\overline{X_1}), q_1(\overline{Y}))$; then, $G' = (r_1(\overline{Y_1}), r_2(\overline{Y_2}), \ldots, r_m(\overline{Y_m}), G_1)\theta$;*

MSW: *$G = \mathtt{msw}(\mathrm{rv}(\overline{X}), Y), G_1$: then $G' = G_1$;*

CONS: *$G = Constr, G_1$ and $Constr$ is satisfiable: then $G' = G_1$.*

A symbolic derivation *of G is a sequence of goals $G_0, G_1, \ldots$ such that $G = G_0$ and, for all $i \geq 0$, $G_i \to G_{i+1}$.*

We only consider successful derivations, i.e., the last step of a derivation resolves to an empty clause.

Note that the traditional notion of derivation in a logic program coincides with that of symbolic derivation when the selected subgoal (literal) is not an `msw` or a constraint. When the selected subgoal is an `msw`, PRISM's inference will construct the next step by enumerating the values of the random variable. In contrast, symbolic derivation skips `msw`'s and constraints and continues with the remaining subgoals in a goal. The effect of these constructs is computed by associating (a) variable type information and (b) a success function (defined below) with each goal in the derivation. Note that the derivation still takes different paths for different values of logic variables and the only difference is the `msw`s and constraints. The symbolic derivation for the goal `widget(X)` over the program in Example 25 is shown in Figure 6.3.

Example 25. *Consider a factory with two machines `a` and `b`. Each machine produces a widget structure and then the structure is painted with a color. In*

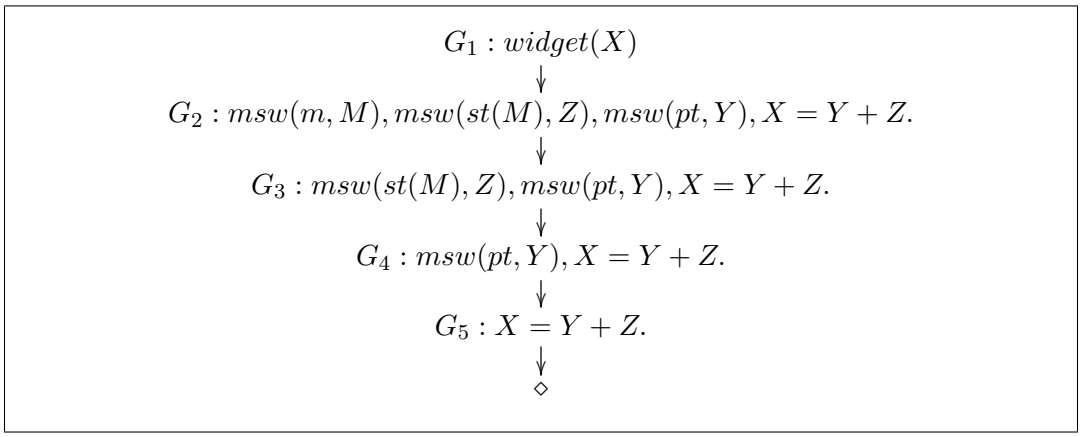


Figure 6.3: Symbolic derivation for goal `widget(X)`

the following program, ***`msw(m, M)`*** *chooses either machine* **`a`** *or* **`b`**, ***`msw(st(M), Z)`*** *gives the cost* **`Z`** *of a product structure,* ***`msw(pt, Y)`*** *gives the cost* **`Y`** *of painting, and finally* ***`X = Y + Z`*** *returns the price of a painted widget* ***`X`***.

```
widget(X) :- msw(m, M),
             msw(st(M), Z),
             msw(pt, Y),
             X = Y + Z.

% Ranges of RVs
values(m, [a,b]).
values(st(M), real).
values(pt, real).

% PDFs and PMFs:
:- set_sw(m, [0.3, 0.7]),
   set_sw(st(a), norm(2.0, 1.0)),
   set_sw(st(b), norm(3.0, 1.0)),
   set_sw(pt, norm(0.5, 0.1)).
```

□

Example 26. *This example illustrates how symbolic derivation differs from traditional logic programming derivation. Consider the following program, where predicate* **`q`** *is defined in terms of* ***`msw(rv, X)`*** *and predicate* **`p`**. *Predicate* **`p`** *has two definitions, in terms of predicates* **`r`** *and* **`s`**. *Figure 6.4 shows the symbolic derivation for goal* $q(Y)$.

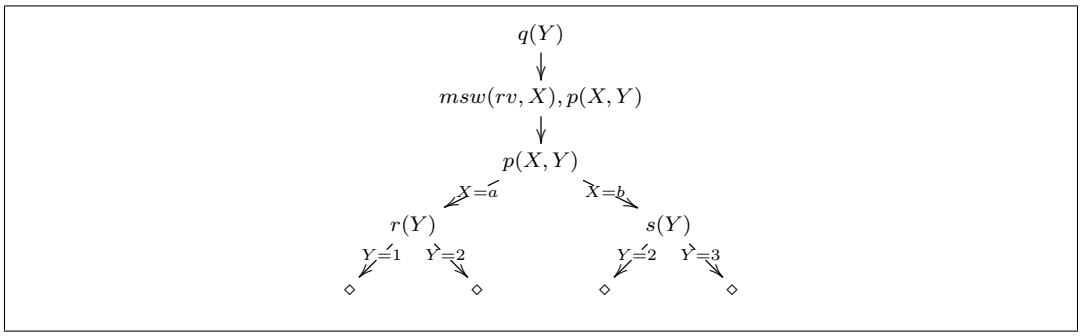


Figure 6.4: Symbolic derivation for goal `q(Y)`

```
q(Y) :- msw(rv, X),
        p(X, Y).

p(a, Y) :- r(Y).
p(b, Y) :- s(Y).

r(1).
r(2).
s(2).
s(3).

% Ranges of RVs
values(rv, [a,b]).

% PDFs and PMFs:
:- set_sw(rv, [0.3, 0.7]).
```

Notice that the symbolic derivation still makes branches in the derivation tree for various logic definitions and outcomes. But the main difference with traditional logic derivation is that it skips `msw` *and* `Constr` *definitions, and continues with the remaining subgoals in a goal.* □

Success Functions: Goals in a symbolic derivation may contain variables whose values are determined by `msw`'s appearing subsequently in the derivation. With each goal G_i in a symbolic derivation, we associate a set of variables, $V(G_i)$, that is a subset of variables in G_i. The set $V(G_i)$ is such that the variables in $V(G_i)$ subsequently appear as parameters or outcomes of `msw`'s in some subsequent goal G_j, $j \geq i$. We can further partition V into two disjoint sets, V_c and V_d, representing continuous and discrete variables, respectively. The sets V_c and V_d are called the derivation variables of G_i, defined below.

Definition 7 (Derivation Variables)**.** *Let $G \rightarrow G'$ such that G' is derived from G using:*

PCR: *Let θ be the mgu in this step. Then $V_c(G)$ and $V_d(G)$ are the largest sets of variables in G such that $V_c(G)\theta \subseteq V_c(G')$ and $V_d(G)\theta \subseteq V_d(G')$.*

MSW: *Let $G = \texttt{msw}(\mathrm{rv}(\overline{X}), Y), G_1$. Then $V_c(G)$ and $V_d(G)$ are the largest sets of variables in G such that $V_c(G) \subseteq V_c(G') \cup \{Y\}$, and $V_d(G) \subseteq V_d(G') \cup \overline{X}$ if Y is continuous, otherwise $V_c(G) \subseteq V_c(G')$, and $V_d(G) \subseteq V_d(G') \cup \overline{X} \cup \{Y\}$.*

CONS: *Let $G = Constr, G_1$. Then $V_c(G)$ and $V_d(G)$ are the largest sets of variables in G such that $V_c(G) \subseteq V_c(G') \cup vars(Constr)$, and $V_d(G) \subseteq V_d(G')$.*

Given a goal G_i in a symbolic derivation, we can associate with it a *success function*, which is a function from the set of all valuations of $V(G_i)$ to $[0, 1]$. Intuitively, the success function represents the probability that the symbolic derivation represents a successful derivation for each valuation of $V(G_i)$. Note that the success function computation uses a set of distribution parameters Θ. For simplicity, we often omit it in the equations and use it when it's not clear from the context.

Representation of success functions: Given a set of variables $\mathbf{V}$, let $\mathbf{C}$ denote the set of all linear equality constraints over reals using $\mathbf{V}$. Let $\mathbf{L}$ be the set of all linear functions over $\mathbf{V}$ with real coefficients. Let $\mathcal{N}_X(\mu, \sigma^2)$ be the PDF of a univariate Gaussian distribution with mean μ and variance σ^2, and $\delta_x(X)$ be the Dirac delta function which is zero everywhere except at x and integration of the delta function over its entire range is 1. Expressions of the form $k * \prod_l \delta_v(V_l) \prod_i \mathcal{N}_{f_i}$, where k is a non-negative real number and $f_i \in \mathbf{L}$, are called *product PDF (PPDF) functions over* $\mathbf{V}$. We use ϕ (possibly subscripted) to denote such functions. A pair $\langle \phi, C \rangle$ where $C \subseteq \mathbf{C}$ is called a *constrained PPDF function*. A sum of a finite number of constrained PPDF functions is called a *success function*, represented as $\sum_i \langle \phi_i, C_i \rangle$.

We use $C_i(\psi)$ to denote the constraints (i.e., C_i) in the i^{th} constrained PPDF function of success function ψ; and $D_i(\psi)$ to denote the i^{th} PPDF function of ψ.

Success functions of base predicates: The success function of a constraint C is $\langle 1, C \rangle$. The success function of *true* is $\langle 1, true \rangle$. The PPDF component of $\texttt{msw}(\mathrm{rv}(\overline{X}), Y)$'s success function is the probability density function of rv's distribution if rv is continuous, and its probability mass function if rv is discrete; its constraint component is *true*.

Example 27. *The success function ψ_1 of `msw(m,M)` for the program in Example 25 is such that $\psi_1 = 0.3\delta_a(M) + 0.7\delta_b(M)$. Note that we can represent*

the success function using tables, where each table row denotes discrete random variable valuations. For example, the above success function can be represented as

M	ψ_1
a	0.3
b	0.7

Thus instead of using delta functions, we often omit it in examples and represent success functions using tables.

The success function ψ_2 of `msw(st(M), Z)` *for the program in Example 25 is such that*

M	ψ_2
a	$\mathcal{N}_Z(2.0, 1.0)$
b	$\mathcal{N}_Z(3.0, 1.0)$

Similarly, the success function ψ_3 of `msw(pt, Y)` *for the program in Example 25 is $\psi_3 = \mathcal{N}_Y(0.5, 0.1)$.*

Finally, the success function ψ_4 of `X = Y + Z` *for the program in Example 25 is $\psi_4 = \langle 1, X = Y + Z \rangle$.*

□

Success functions of user-defined predicates: If $G \rightarrow G'$ is a step in a derivation, then the success function of G is computed bottom-up based on the success function of G'. This computation is done using *join* and *marginalize* operations on success functions.

Definition 8 (Join)**.** *Let $\psi_1 = \sum_i \langle D_i, C_i \rangle$ and $\psi_2 = \sum_j \langle D_j, C_j \rangle$ be two success functions, then join of ψ_1 and ψ_2 represented as $\psi_1 * \psi_2$ is the success function $\sum_{i,j} \langle D_i D_j, C_i \wedge C_j \rangle$.*

Example 28. *Let $\psi_{msw(m,M)}(M)$ and $\psi_G(X, Y, Z, M)$ be defined as follows:*

M	$\psi_{msw(m,M)}(M)$
a	0.3
b	0.7

M	$\psi_G(X, Y, Z, M)$
a	$\langle \mathcal{N}_Z(2.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle$
b	$\langle \mathcal{N}_Z(3.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle$

Then join of $\psi_{msw(m,M)}(M)$ and $\psi_G(X, Y, Z, M)$ yields:

M	$\psi_{msw(m,M)}(M) * \psi_G(X, Y, Z, M)$
a	$\langle 0.3\mathcal{N}_Z(2.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z\rangle$
b	$\langle 0.7\mathcal{N}_Z(3.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z\rangle$

□

Note that we perform a simplification of success functions after the join operation. We eliminate any PPDF term in ψ which is inconsistent w.r.t. delta functions. For example, $\delta_a(M)\delta_b(M) = 0$ as M can not be both a and b at the same time.

Given a success function ψ for a goal G, the success function for $\exists X.\ G$ is computed by the marginalization operation. Marginalization w.r.t. a discrete variable is straightforward and omitted. Below we define marginalization w.r.t. continuous variables in two steps: first rewriting the success function in a projected form and then doing the required integration.

The goal of projection is to eliminate any linear constraint on V, where V is the continuous variable to marginalize over. The projection operation involves finding a linear constraint (i.e., $V = \overline{a} \cdot \overline{X} + b$) on V and replacing all occurrences of V in the success function by $\overline{a} \cdot \overline{X} + b$.

Definition 9 (Projection)**.** *Projection of a success function ψ w.r.t. a continuous variable V, denoted by $\psi \downarrow_V$, is a success function ψ' such that*
$\forall i.\ D_i(\psi') = D_i(\psi)[\overline{a} \cdot \overline{X} + b/V]$; and
$C_i(\psi') = (C_i(\psi) - C_{ip})[\overline{a} \cdot \overline{X} + b/V]$,
where C_{ip} is a linear constraint ($V = \overline{a} \cdot \overline{X} + b$) on V in $C_i(\psi)$ and $t[s/x]$ denotes replacement of all occurrences of x in t by s.

Note that the replacement of V by $\overline{a} \cdot \overline{X} + b$ in PDFs and linear constraints does not alter the general form of a success function. Thus projection returns a success function. Notice that if ψ does not contain any linear constraint on V, then the projected form remains the same.

Example 29. *Let ψ_1 represent the following success function*

$$\psi_1 = \langle 0.3\mathcal{N}_Z(2.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z\rangle.$$

Then projection of ψ_1 w.r.t. Y yields

$$\psi_1 \downarrow_Y = 0.3\mathcal{N}_Z(2.0, 1.0).\mathcal{N}_{X-Z}(0.5, 0.1). \tag{6.1}$$

Notice that Y is replaced by $X - Z$. □

Proposition 6. *Integration of a PPDF function with respect to a variable V is a PPDF function, i.e.,*

$$\alpha \int_{-\infty}^{\infty} \prod_{k=1}^{m} \mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k, \sigma_k^2) dV = \alpha' \prod_{l=1}^{m'} \mathcal{N}_{(\overline{a'_l}\cdot\overline{X'_l})+b'_l}(\mu'_l, \sigma'^2_l)$$

where $V \in \overline{X_k}$ and $V \notin \overline{X'_l}$.

For example, the integration of $\mathcal{N}_{a_1V-X_1}(\mu_1, \sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2, \sigma_2^2)$ w.r.t. variable V is

$$\begin{aligned} &\int_{-\infty}^{\infty} \mathcal{N}_{a_1V-X_1}(\mu_1, \sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2, \sigma_2^2) dV \\ &= \mathcal{N}_{a_2X_1-a_1X_2}(a_1\mu_2 - a_2\mu_1, a_2^2\sigma_1^2 + a_1^2\sigma_2^2). \end{aligned} \tag{6.2}$$

Here X_1, X_2 are linear combinations of variables (except V). A proof of the proposition is given in 6.7.

Definition 10 (Integration)**.** *Let ψ be a success function that does not contain any linear constraints on V. Then integration of ψ with respect to V, denoted by $\oint_V \psi$ is a success function ψ' such that $\forall i. D_i(\psi') = \int D_i(\psi) dV$.*

It is easy to see (using Proposition 6) that the integral of success functions are also success functions. Note that if ψ does not contain any PDF on V, then the integrated form remains the same.

Example 30. *Let ψ_2 represent the following success function*

$$\psi_2 = 0.3\mathcal{N}_Z(2.0, 1.0).\mathcal{N}_{X-Z}(0.5, 0.1).$$

Then integration of ψ_2 w.r.t. Z yields

$$\begin{aligned} \oint_Z \psi_2 &= \int 0.3\mathcal{N}_Z(2.0, 1.0).\mathcal{N}_{X-Z}(0.5, 0.1) dZ \\ &= 0.3\mathcal{N}_X(2.5, 1.1). \\ &\textit{(using Equation 6.2)} \end{aligned} \tag{6.3}$$

□

Definition 11 (Marginalize)**.** *Marginalization of a success function ψ with respect to a variable V, denoted by $\mathbb{M}(\psi, V)$, is a success function ψ' such that*

$$\psi' = \oint_V \psi \downarrow_V$$

We overload $\mathbb{M}$ to denote marginalization over a set of variables, defined such that $\mathbb{M}(\psi, \{V\} \cup \overline{X}) = \mathbb{M}(\mathbb{M}(\psi, V), \overline{X})$ and $\mathbb{M}(\psi, \{\}) = \psi$.

Proposition 7. *The set of all success functions is closed under join and marginalize operations.*

The success function for a derivation is defined as follows.

Definition 12 (Success function of a goal)**.** *The success function of a goal G, denoted by ψ_G, is computed based on the derivation $G \to G'$:*

$$\psi_G = \begin{cases} \sum_{G'} \mathbb{M}(\psi_{G'}, V(G') - V(G)) & \text{for all program clause resolution } G \to G' \\ \psi_{msw(rv(\overline{X}),Y)} * \psi_{G'} & \text{if } G = \texttt{msw}(\mathrm{rv}(\overline{X}), Y), G_1 \\ \psi_{Constr} * \psi_{G'} & \text{if } G = Constr, G_1 \end{cases}$$

Note that the above definition carries PRISM's assumption that an instance of a random variable occurs at most once in any derivation. In particular, the PCR step marginalizes success functions w.r.t. a set of variables; the valuations of the set of variables must be mutually exclusive for correctness of this step. The MSW step joins success functions; the goals joined must use independent random variables for the join operation to correctly compute success functions in this step.

Example 31. *Figure 6.3 shows the symbolic derivation for the goal* `widget(X)` *over the mixture model program in Example 25. The success function of goal G_5 is $\psi_{G_5}(X, Y, Z) = \langle 1, X = Y + Z \rangle$.*
[Join]

$$\begin{aligned} \psi_{G_4}(X, Y, Z) &= \psi_{msw(pt,Y)}(Y) * \psi_{G_5}(X, Y, Z) \\ &= \langle \mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle. \end{aligned}$$

The success function of goal G_3 is $\psi_{msw(st(M),Z)}(Z) * \psi_{G_4}(X, Y, Z)$, which yields:

M	$\psi_{G_3}(X, Y, Z, M)$
a	$\langle \mathcal{N}_Z(2.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle$
b	$\langle \mathcal{N}_Z(3.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle$

Then join of $\psi_{msw(m,M)}(M)$ and $\psi_{G_3}(X, Y, Z, M)$ yields (see Example 28):

M	$\psi_{G_2}(X, Y, Z, M)$
a	$\langle 0.3\mathcal{N}_Z(2.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle$
b	$\langle 0.7\mathcal{N}_Z(3.0, 1.0).\mathcal{N}_Y(0.5, 0.1), X = Y + Z \rangle$

[Marginalize]
Finally, $\psi_{G_1}(X) = \mathbb{M}(\psi_{G_2}(X,Y,Z,M), \{M,Y,Z\})$.

First we marginalize $\psi_{G_2}(X,Y,Z,M)$ *w.r.t.* M*:*

$$\begin{aligned}\mathbb{M}(\psi_{G_2}, M) &= \oint_M \psi_{G_2} \downarrow_M \\ &= \langle 0.3\mathcal{N}_Z(2.0,1.0).\mathcal{N}_Y(0.5,0.1), X = Y + Z\rangle \\ &+ \langle 0.7\mathcal{N}_Z(3.0,1.0).\mathcal{N}_Y(0.5,0.1), X = Y + Z\rangle.\end{aligned}$$

Next we marginalize $\psi_{G_2}(X,Y,Z)$ *w.r.t.* Y*:*

$$\begin{aligned}\mathbb{M}(\psi_{G_2}, Y) &= \oint_Y \psi_{G_2} \downarrow_Y \\ &= 0.3\mathcal{N}_Z(2.0,1.0).\mathcal{N}_{X-Z}(0.5,0.1) + 0.7\mathcal{N}_Z(3.0,1.0).\mathcal{N}_{X-Z}(0.5,0.1).\end{aligned}$$

Finally, we marginalize $\psi_{G_2}(X,Z)$ *over variable* Z *to get* $\psi_{G_1}(X)$*:*

$$\begin{aligned}\psi_{G_1}(X) &= \mathbb{M}(\psi_{G_2}, Z) \\ &= \oint_Z \psi_{G_2} \downarrow_Z \\ &= 0.3\mathcal{N}_X(2.5,1.1) + 0.7\mathcal{N}_X(3.5,1.1). \\ &\textit{(using Equation 6.3)}\end{aligned}$$

□

Example 32. *In this example, we compute success function of goal* $q(Y)$ *in Example 26. Figure 6.4 shows the symbolic derivation for goal* $q(Y)$*. Success function of* $r(Y)$ *is* $\delta_1(Y)+\delta_2(Y)$*, and success function of* $s(Y)$ *is* $\delta_2(Y)+\delta_3(Y)$*. Similarly, success function of* $p(X,Y)$ *is* $\delta_a(X)(\delta_1(Y)+\delta_2(Y))+\delta_b(X)(\delta_2(Y)+\delta_3(Y))$*. Now*

$$\psi_{q(Y)} = \mathbb{M}(\psi_{msw(rv,X)} * \psi_{p(X,Y)}, X)$$

Success function of $msw(rv,X)$ *is* $0.3\delta_a(X) + 0.7\delta_b(X)$*. Join of* $\psi_{msw(rv,X)}$ *and* $\psi_{p(X,Y)}$ *yields* $0.3\delta_a(X)(\delta_1(Y)+\delta_2(Y))+0.7\delta_b(X)(\delta_2(Y)+\delta_3(Y))$*. Finally,* $\psi_{q(Y)} = 0.3(\delta_1(Y) + \delta_2(Y)) + 0.7(\delta_2(Y) + \delta_3(Y))$.

When $Y = 1$*, only* $p(a,1)$ *is true. Thus* $\psi_{q(1)} = 0.3$*. On the other hand,* $\psi_{q(2)} = 1.0$ *as both* $p(a,2)$ *and* $p(b,2)$ *are true when Y=2. Similarly,* $\psi_{q(3)} = 0.7$*.* □

Note that the definition of success function applies to a symbolic derivation. We can define success function for a goal in terms of the success functions of all

symbolic derivations for that goal; if the symbolic derivations are all mutually exclusive, this is simply the sum of success functions of all derivations. Thus the success function of a goal represents the likelihood of a successful derivation for each instance of a goal. Hence the probability measure computed by the success function is what PRISM calls *inside* probability.

Since the derivation can take different paths based on the valuation of logic variables (non-random variables), the same goal can be encountered in different paths of a derivation (i.e., common goal in the derivation tree for a given top goal). Thus it is possible to eliminate redundant computation by saving results of common goals. We save the success function of a goal in a table (indexed by that goal). We call it *tabled success function*. Whenever we encounter a goal, we first check whether there exists a success function in the table for that goal. If the success function exists then we return it; otherwise we compute a success function using Definition 12.

6.2 Correctness of the Inference Algorithm

The technically complex aspect of correctness is the closure of the set of success functions w.r.t. join and marginalize operations. Proposition 6 and 7 state these closure properties and the proofs of these propositions are presented in Section 6.7.

Definition 12 represents the inference algorithm for computing the success function of a goal. The distribution of a goal is formally defined in terms of the distribution semantics (i.e., P_{PL} in Section 5.2) of extended PRISM programs and is computed using the inference algorithm. We show that the success function computed by our inference algorithm represents the distribution of a goal. For example, the success function of goal G_1 in Example 31 represents the distribution of variable X. More specifically, success functions represent the distribution of an answer to a goal, e.g., distribution of $G\theta$ for a goal G with variable substitution θ.

Theorem 8. *The success function of a goal computed by the inference algorithm represents the distribution of the answer to that goal.*

Proof. Correctness w.r.t. distribution semantics follows from the definition of join and marginalize operations, and PRISMs independence and exclusiveness assumptions. We prove this by induction on derivation length n. For $n = 1$, the definition of success function for base predicates gives a correct distribution.

Now let's assume that for a derivation of length n, our inference algorithm computes valid distribution. Let's assume that G' has a derivation of length n and $G \to G'$. Thus G has a derivation of length $n + 1$. We show that the success function of G represents a valid distribution.

We compute ψ_G using Definition 12 and it carries PRISM's assumption that an instance of a random variable occurs at most once in any derivation. More specifically, the PCR step marginalizes $\psi_{G'}$ w.r.t. a set of variables $V(G') - V(G)$. Since according to PRISM's exclusiveness assumption the valuations of the set of variables are mutually exclusive, the marginalization operation returns a valid distribution. Analogously, the MSW/CONS step joins success functions, and the goals joined use independent random variables (following PRISM's assumption) for the join operation to correctly compute ψ_G in this step. Thus ψ_G represents a valid distribution. □

6.3 Complexity Analysis

Complexity: Let S_i denote the number of constrained PPDF terms in ψ_i; P_i denote the maximum number of product terms in any PPDF function in ψ_i; and Q_i denote the maximum size of a constraint set (C_i) in ψ_i. The time complexity of the two basic operations used in constructing a symbolic derivation is as follows.

Proposition 9 (Time Complexity)**.** *The worst-case time complexity of* $Join(\psi_i, \psi_j)$ *is* $O(S_i * S_j * (P_i * P_j + Q_i * Q_j))$.

The worst-case time complexity of $\mathbb{M}(\psi_g, V)$ *is* $O(S_g * P_g)$ *when* V *is discrete and* $O(S_g * (P_g + Q_g))$ *when* V *is continuous.*

Note that when computing the success function of a goal in a derivation, the join operation is limited to joining the success function of a single `msw` or a single constraint set to the success function of a goal, and hence the parameters S_i, P_i, and Q_i are typically small. The complexity of the *size* of success functions is as follows.

Proposition 10 (Success Function Size)**.** *For a goal* G *and its symbolic derivation, the following hold:*

1. *The maximum number of product terms in any PPDF function in* ψ_G *is linear in* $|V_c(G)|$*, the number of continuous variables in* G*.*
2. *The maximum size of a constraint set in a constrained PPDF function in* ψ_G *is linear in* $|V_c(G)|$*.*
3. *The maximum number of constrained PPDF functions in* ψ_G *is potentially exponential in the number of discrete random variables in the symbolic derivation.*

Proof. We prove (1) by contradiction. Let there are n continuous variables in goal G, then any PPDF function in G will contain at most n Gaussians. If it contains more than n Gaussians, then there exists a continuous variable V which has more than one Gaussian functions. Thus multiplying these Gaussians will yield a single Gaussian according to Gaussian functions properties (Property 2). Similarly, it can be proved that the maximum size of a constraint set is linear in $|V_c(G)|$.

The success functions can be alternatively represented using tabular form where each row denotes discrete random variables valuation and corresponding constrained PPDF function. Since the total number of rows is exponential in the number of discrete random variables, the maximum number of constrained PPDF functions in ψ_G is also exponential in the number of discrete random variables. □

The number of product terms and the size of constraint sets are hence independent of the length of the symbolic derivation. Note that for a program with only discrete random variables, there may be exponentially fewer symbolic derivations than concrete derivations. The compactness is only in terms of *number* of derivations and not the total size of the representations. In fact, for programs with only discrete random variables, there is a one-to-one correspondence between the entries in the tabular representation of success functions and PRISM's answer tables. For such programs, it is easy to show that the time complexity of the inference algorithm presented in this paper is same as that of PRISM.

Example 33. *In this example, we show that for programs with only discrete random variables, there is a one-to-one correspondence between the entries in the tabular representation of success functions and PRISM's derivations paths.*

Figure 6.1 shows PRISM's derivation for the goal `hbn(X,Y)` *over the Bayesian network program in Example 16. The derivation has four paths, e.g., for (i) X=0, Y=0; (ii) X= 0, Y=1; (iii) X=1, Y=0; and (iv) X =1, Y=1.*

The success function of the goal $hbn(X,Y)$ *can be represented using the following table. It is computed by joining the success functions of* $msw(x,X)$ *and* $msw(y(X),Y)$.

X	Y	$\psi_{hbn(X,Y)}(X,Y)$
0	0	0.20
0	1	0.20
1	0	0.48
1	1	0.12

Notice that there is a one-to-one correspondence between the entries in the table and derivation paths in Figure 6.1. In addition, PRISM also gives the four answers with the above mentioned probabilities. □

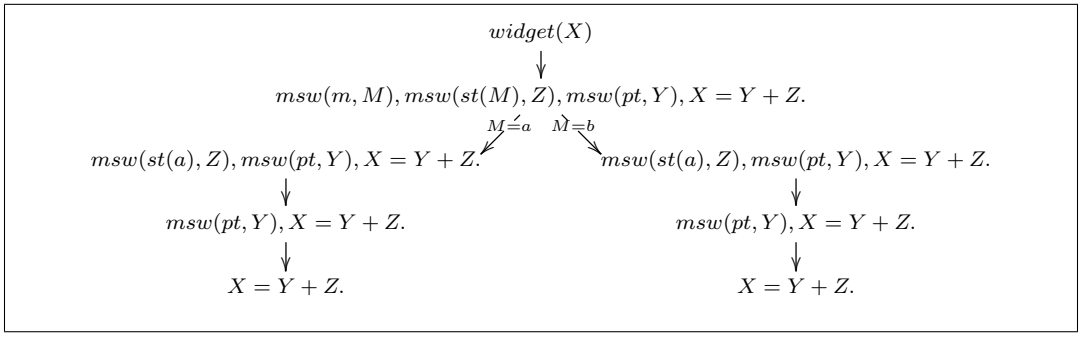


Figure 6.5: Derivation for goal `widget(X)`

Like PRISM, it's possible to construct the next step of a derivation by enumerating the values of discrete variables, i.e., creating different paths in a derivation based on discrete variable values and constructing symbolic derivation only for continuous variables. For example, the symbolic derivation for *widget(X)* in Figure 6.3 can be represented as in Figure 6.5 where the derivation takes two paths based on the valuation of the random process m. Each path (along with the discrete variable valuation) represents a table entry in the tabular representation of success function (Example 31). For programs with only discrete random variables, each path corresponds to an explanation in PRISM, i.e., there is a one-to-one correspondence between an explanation (in PRISM) and entries in the table. Thus the complexity of the inference algorithm is same as that of PRISM's for programs with only discrete random variables.

Instead of enumerating the values of discrete random variables, we follow the symbolic computation for discrete as well as continuous random variables. The reason is as follows. First, it gives a uniform approach to handle all types of random variables. Second and more importantly, it makes the language more powerful and gives flexibility to model complex problems (e.g., Hybrid models in Section 2.3.8).

6.4 Illustrative Example: Kalman Filter

In this section, we model Kalman filters [53] using probabilistic logic programs. The model describes a random walk of a single continuous state variable S_t with noisy observation V_t. The initial state distribution is assumed to be Gaussian with mean μ_0, and variance σ_0^2. The transition and sensor models are Gaussian noises with zero means and constant variances σ_s^2, σ_v^2 respectively.

```
kf(N, T) :-
    msw(init, S),
    kf_part(0, N, S, T).

kf_part(I, N, S, T) :-
    I < N, NextI is I+1,
    trans(S, NextS),
    emit(NextS, V),
    obs(NextI, V),
    kf_part(NextI, N, NextS, T).
kf_part(I, N, S, T) :- I=N, T=S.

trans(S, NextS) :-
    msw(trans_err, E),
    NextS = S + E.

emit(NextS, V) :-
    msw(obs_err, X),
    V = NextS + X.
```

Figure 6.6: Logic program for Kalman Filter.

Figure 6.6 shows a logic program for Kalman filter, and Figure 6.7 shows the derivation for a query $kf(1, T)$. Note the similarity between this and `hmm` program (Figure 4.1): only trans/emit definitions are different. We label the i^{th} derivation step by G_i which is used in the next subsection to refer to appropriate derivation step. Here, our goal is to compute filtered distribution of state T.

Success Function Computation: Using our definition of success functions, the success function of the leaf goal in Figure 6.7 (G_{15}) is
$\psi_{G_{15}} = \langle 1, T = NextS \rangle$.

$\psi_{G_{13}}$ and $\psi_{G_{14}}$ are same as $\psi_{G_{15}}$.

$\psi_{G_{12}}$ is same as $\psi_{G_{13}}$ except that $obs(1, V)$ binds V to an observation v_1. Thus, $\psi_{G_{11}}$ is $Join(\psi_{v_1=NextS+X}, \psi_{G_{12}})$ which yields

$$\psi_{G_{11}} = \langle 1, T = NextS \wedge v_1 = NextS + X \rangle.$$

Now $\psi_{G_{10}}$ is $Join(\psi_{msw(obs\_err)}, \psi_{G_{11}})$ which gives

$$\psi_{G_{10}} = \langle \mathcal{N}_X(0, \sigma_v^2), T = NextS \wedge v_1 = NextS + X \rangle.$$

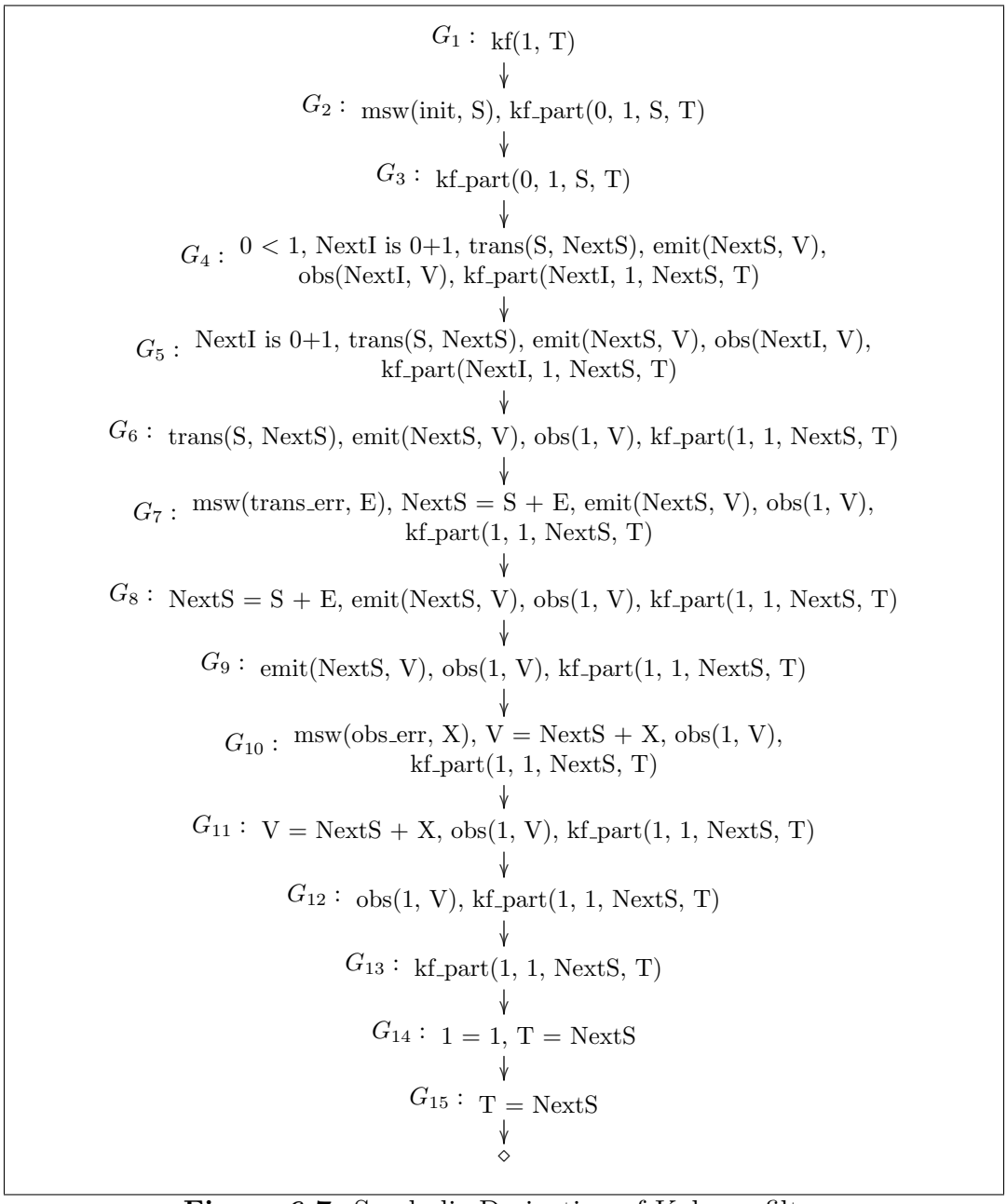


Figure 6.7: Symbolic Derivation of Kalman filter

Marginalizing $\psi_{G_{10}}$ over X yields ψ_{G_9}. Thus,

$$\begin{aligned}\psi_{G_9} &= \mathbb{M}(\psi_{G_{10}}, X) \\ &= \langle \mathcal{N}_{v_1 - NextS}(0, \sigma_v^2), T = NextS \rangle.\end{aligned}$$

Similarly,

$$\begin{aligned}\psi_{G_8} &= Join(\psi_{NextS=S+E}, \psi_{G_9}) \\ &= \langle \mathcal{N}_{v_1 - NextS}(0, \sigma_v^2), T = NextS \wedge NextS = S + E \rangle.\end{aligned}$$

$$\begin{aligned}\psi_{G_7} &= Join(\psi_{msw(trans\_err)}, \psi_{G_8}) \\ &= \langle \mathcal{N}_{v_1 - NextS}(0, \sigma_v^2).\mathcal{N}_E(0, \sigma_s^2), \\ &\quad T = NextS \wedge NextS = S + E \rangle.\end{aligned}$$

$$\begin{aligned}\psi_{G_6} &= \mathbb{M}(\psi_{G_7}, E) \\ &= \langle \mathcal{N}_{v_1 - NextS}(0, \sigma_v^2).\mathcal{N}_{NextS - S}(0, \sigma_s^2), T = NextS \rangle.\end{aligned}$$

ψ_{G_4} and ψ_{G_5} are same as ψ_{G_6}. Next,

$$\begin{aligned}\psi_{G_3} &= \mathbb{M}(\psi_{G_4}, NextS) \\ &= \mathcal{N}_{v_1 - T}(0, \sigma_v^2).\mathcal{N}_{T-S}(0, \sigma_s^2).\end{aligned}$$

$$\begin{aligned}\psi_{G_2} &= Join(\psi_{msw(init)}, \psi_{G_3}) \\ &= \mathcal{N}_{v_1 - T}(0, \sigma_v^2).\mathcal{N}_{T-S}(0, \sigma_s^2).\mathcal{N}_S(\mu_0, \sigma_0^2).\end{aligned}$$

Finally,

$$\begin{aligned}\psi_{G_1} &= \mathbb{M}(\psi_{G_2}, S) \\ &= \mathcal{N}_{v_1 - T}(0, \sigma_v^2).\mathcal{N}_T(\mu_0, \sigma_0^2 + \sigma_s^2). \\ &\text{(using Equation 6.2)} \\ &= \mathcal{N}_T(v_1, \sigma_v^2).\mathcal{N}_T(\mu_0, \sigma_0^2 + \sigma_s^2). \\ &\text{(constant shifting, Property 3)} \\ &= \mathcal{N}_T\left(\frac{(\sigma_0^2 + \sigma_s^2) * v_1 + \sigma_v^2 * \mu_0}{\sigma_0^2 + \sigma_s^2 + \sigma_v^2}, \frac{(\sigma_0^2 + \sigma_s^2) * \sigma_v^2}{\sigma_0^2 + \sigma_s^2 + \sigma_v^2}\right). \\ &\text{(product of two Gaussian PDFs is another PDF, Property 2)}\end{aligned}$$

which is the filtered distribution of state T after seeing one observation, which is equal to the filtered distribution presented in [53].

6.5 Extensions

6.5.1 Gamma Distribution

We can also readily extend our inference techniques to support Gamma distributions. More generally, the PDF functions can be generalized to contain Gaussian or Gamma density functions, such that variables are not shared between Gaussian and Gamma density functions.

The probability density function of a Gamma distributed random variable x is

$$f(x; k, \theta) = \frac{1}{\theta^k} \frac{1}{\Gamma(k)} x^{k-1} e^{-\frac{x}{\theta}}$$

where k and θ are shape and scale parameters respectively. The definition of *join* operation for Gamma distribution remains the same. For *marginalize* operation the only change is to extend the integration function to handle PDFs of Gamma distribution.

6.5.2 Multivariate Gaussian

For simplicity, in this thesis we focused only on univariate Gaussians. However, the techniques can be easily extended to support multivariate Gaussian distributions, by extending the integration function (*Definition 10*), and `set_sw` directives. We presented the closed form solution of the integration for univariate Gaussian in Proposition 6. For multivariate Gaussian, the closed form solution of the integration can be computed similarly.

6.5.3 Call Functions and Smoothed Distributions

We can define a function that represents the likelihood that a goal G' will be encountered in a symbolic derivation starting at goal G. This "call" function will represent the *outside* probability of PRISM. Alternatively, we can use the Magic Sets transformation originally proposed for deductive databases [3] to compute call functions of a program in terms of success functions of a transformed program. As mentioned in the background section, the ability to compute inside and outside probabilities can be used to infer smoothed distributions for temporal models.

6.5.4 Hybrid Models

The symbolic inference procedure enables us to reason over a large class of statistical models such as Hybrid Bayesian Networks with discrete child-discrete parent, continuous child-discrete parent (finite mixture model), and continuous child-continuous parent (Kalman filter), which was hitherto not possible in

PLP frameworks. It can also be used for hybrid models, e.g., models that mix discrete and Gaussian distributions. For instance, consider the finite mixture model example (Example 34) where `w(a)` is Gaussian but `w(b)` is a discrete distribution with values 1 and 2 with 0.5 probability each. The density of the mixture distribution can be written as

$$f(X) = 0.3\mathcal{N}_X(1.0, 0.2) + 0.35\delta_{1.0}(X) + 0.35\delta_{2.0}(X)$$

Example 34 (Hybrid Model)**.** *In this example, we encode the above density in extended PRISM.*

```
fmix(X) :- msw(m, M),
           msw(w(M), X).

% Ranges of RVs
values(m, [a,b]).
values(w(a), real).
values(w(b), [1, 2]).

% PDFs and PMFs
:- set_sw(m, [0.3, 0.7]),
   set_sw(w(a), norm(1.0, 0.2)),
   set_sw(w(b), [0.5, 0.5]).
```

□

Thus the language can be used to model problems that lie outside traditional Hybrid Bayesian Networks.

6.5.5 Lifting PRISM's restrictions

ProbLog and PITA [52], an implementation of LPAD, lift PRISM's mutual exclusion and independence restrictions. Their inference technique first materializes the set of explanations for each query, and represents this set as a BDD, where each node in the BDD is a (discrete) random variable. Distinct paths in the BDD are mutually exclusive and variables in a single path are all independent. Probabilities of query answers are computed with a simple dynamic programming algorithm using this BDD representation.

The technical development in this thesis is based on PRISM and imposes its restrictions. However, by materializing the set of symbolic derivations first (analogous to the inference procedure of ProbLog and PITA), representing them in a factored form (such as a BDD), and then computing success functions on this representation, the mutual exclusion and independence restrictions can be lifted for models with continuous variables as well.

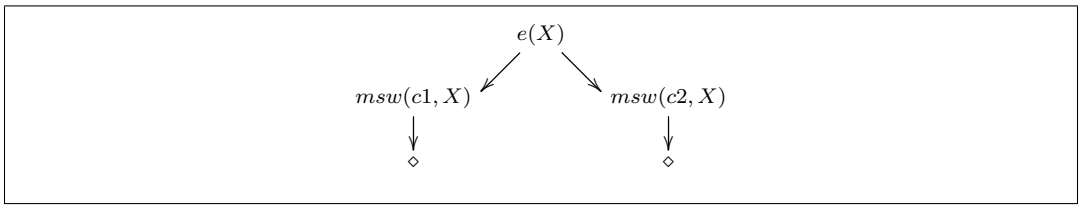


Figure 6.8: Derivation for goal `e(X)`

Example 35. *We will illustrate the idea of BDD-based success function computation using the following program, where $c1$ and $c2$ denote fair and biased coins respectively, and predicate $e(X)$ denotes the outcome of the coin tossing experiment.*

```
e(X) :-  msw(c1, X).
e(X) :-  msw(c2, X).

% Ranges of RVs
values(c1, [h,t]).
values(c2, [h,t]).

% PDFs and PMFs:
:- set_sw(c1, [0.5, 0.5]),
   set_sw(c2, [0.3, 0.7]).
```

□

Note that for the above program, PRISM will not be able to compute probabilities correctly as the derivations are not mutually exclusive (Figure 6.8).

Let us explain the BDD representation for goal $e(h)$ in ProbLog/PITA. The encoding requires ordering of the variables. Let $c1$ appears before $c2$. Then the first branch of the BDD checks whether the outcome of $c1$ is h. The second branch checks whether the outcome of $c2$ is h. The BDD construction for goal $e(t)$ follows the same approach.

We can follow the above mentioned approach to construct a symbolic BDD representation for goal $e(X)$. We represent the outcomes of the coin tosses $c1, c2$ using variables Y and Z respectively. In the BDD representation (Figure 6.9), we first toss coin $c1$ and check whether Y equals to X in the first branch. The second branch first checks $X \neq Y$, and then decides whether the outcome Z equals to X.

In order to compute success function on Figure 6.9, we need to define success function of dis-equality constraints. Success function of a dis-equality constraint $X \neq Y$ is defined as

$$\psi_{X \neq Y} = \langle 1, true \rangle - \langle 1, X = Y \rangle.$$

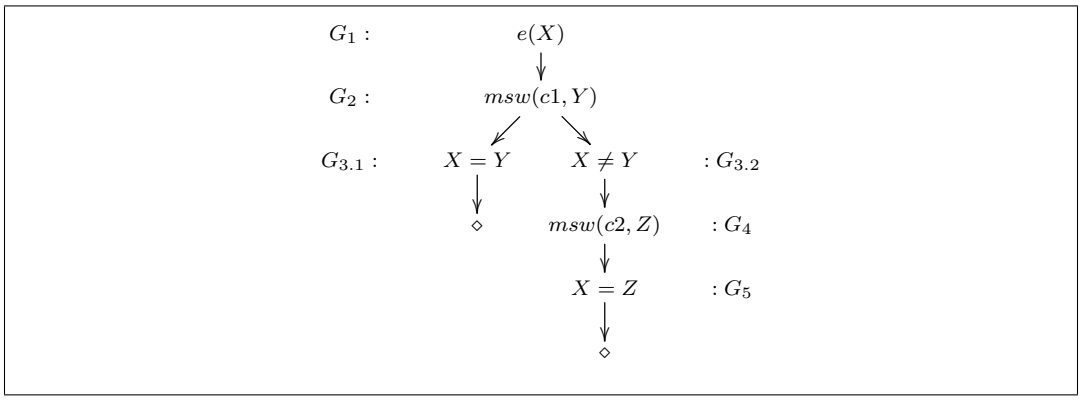


Figure 6.9: BDD representation for goal `e(X)`

Note that we do not pose any restriction over the constants in success function, i.e., constants can be positive/negative real values. Thus the negative constant in the above equation does not pose any additional restriction. Since dis-equality constraints are not used in success functions, the general form of constraints that we permit in our language remains the same. Thus the above function maintains the general form of success function, and the set of success functions are still closed under join and marginalize operations.

Example 36. *In this example, we show the success function computation of goal* $e(X)$ *in Figure 6.9. Success function of goal* $G_{3.1}$ *is* $\langle 1, X = Y \rangle$*. Similarly,*

$$\begin{aligned}
\psi_{G_5}(X, Z) &= \langle 1, X = Z \rangle \\
\psi_{G_4}(X, Z) &= \psi_{msw(c2)}(Z) * \psi_{G_5}(X, Z) \\
&= (0.3\delta_h(Z) + 0.7\delta_t(Z)) * \langle 1, X = Z \rangle \\
&= \langle 0.3\delta_h(Z), X = Z \rangle + \langle 0.7\delta_t(Z), X = Z \rangle
\end{aligned}$$

Now success function of goal $G_{3.2}$ *is*

$$\psi_{G_{3.2}}(X, Y) = \mathbb{M}(\psi_{X \neq Y} * \psi_{G_4}(X, Z), Z)$$

Note that $\psi_{X \neq Y} = \langle 1, true \rangle - \langle 1, X = Y \rangle$.

Thus

$$\begin{aligned}
\psi_{X \neq Y} * \psi_{G_4}(X, Z) = &\langle 0.3\delta_h(Z), X = Z \rangle + \langle 0.7\delta_t(Z), X = Z \rangle \\
&- \langle 0.3\delta_h(Z), X = Z, X = Y \rangle - \langle 0.7\delta_t(Z), X = Z, X = Y \rangle.
\end{aligned}$$

Marginalizing the above success function w.r.t Z yields

$$\begin{aligned}\psi_{G_{3.2}}(X,Y) &= \mathbb{M}(\psi_{X\neq Y} * \psi_{G_4}(X,Z), Z)\\ &= 0.3\delta_h(X) + 0.7\delta_t(X) - \langle 0.3\delta_h(X), X = Y\rangle - \langle 0.7\delta_t(X), X = Y\rangle\end{aligned}$$

Then success function of goal G_2 is

$$\begin{aligned}\psi_{G_2}(X,Y) &= \psi_{msw(c1)} * (\psi_{G_{3.1}} + \psi_{G_{3.2}})\\ &= (0.5\delta_h(Y) + 0.5\delta_t(Y)) * (\langle 1, X = Y\rangle\\ &+ 0.3\delta_h(X) + 0.7\delta_t(X) - \langle 0.3\delta_h(X), X = Y\rangle - \langle 0.7\delta_t(X), X = Y\rangle)\\ &= \langle 0.5\delta_h(Y), X = Y\rangle + \langle 0.5\delta_t(Y), X = Y\rangle + 0.15\delta_h(Y)\delta_h(X)\\ &+ 0.35\delta_h(Y)\delta_t(X) + 0.15\delta_t(Y)\delta_h(X) + 0.35\delta_t(Y)\delta_t(X)\\ &- \langle 0.15\delta_h(Y)\delta_h(X), X = Y\rangle - \langle 0.35\delta_h(Y)\delta_t(X), X = Y\rangle\\ &- \langle 0.15\delta_t(Y)\delta_h(X), X = Y\rangle - \langle 0.35\delta_t(Y)\delta_t(X), X = Y\rangle\end{aligned}$$

Finally, marginalizing $\psi_{G_2}(X,Y)$ w.r.t. Y yields $\psi_{G_1}(X)$.

$$\begin{aligned}\psi_{G_1}(X) &= 0.5\delta_h(X) + 0.5\delta_t(X) + 0.15\delta_h(X) + 0.35\delta_t(X)\\ &+ 0.15\delta_h(X) + 0.35\delta_t(X) - 0.15\delta_h(X)\delta_h(X)\\ &- 0.35\delta_h(X)\delta_t(X) - 0.15\delta_t(X)\delta_h(X) - 0.35\delta_t(X)\delta_t(X)\\ &= 0.65\delta_h(X) + 0.85\delta_t(X)\end{aligned}$$

Note that this function gives the correct probability of $e(h)$ and $e(t)$. □

Thus the success function computation remains the same. However, the main difficulty here is to construct a factored-form or BDD-based representation of symbolic derivation, which is a topic of future work. For simplicity, we used discrete variables in the above example. Continuous variables follow the same approach where the only difference is that delta functions and discrete probabilities are replaced by Gaussian functions.

6.6 Implementation

We implemented the extended inference algorithm presented in this thesis in the XSB logic programming system [59]. The system is available at `http://www.cs.sunysb.edu/~cram/contdist`. This proof-of-concept prototype is implemented as a meta-interpreter and currently supports discrete and Gaussian distributions. The meaning of various probabilistic predicates (e.g., `msw, values, set_sw`) in the system are similar to that of PRISM system. This implementation illustrates how the inference algorithm specializes to the specialized techniques that have been developed for several popular statistical

models such as HMM, FMM, Hybrid Bayesian Networks and Kalman Filters.

6.7 Closure of Success Functions: Proof of Propositions 6 and 7

This section presents proof of Propositions 6 and 7.

Proposition 6. *Integrated form of a PPDF function with respect to a variable V is a PPDF function, i.e.,*

$$\int_{-\infty}^{\infty} \prod_{k=1}^{m} \mathcal{N}_{(\overline{a_k}.\overline{X_k}+b_k)}(\mu_k, \sigma_k^2) dV = \alpha \prod_{l=1}^{m'} \mathcal{N}_{(\overline{a'_l}.\overline{X'_l})+b'_l}(\mu'_l, \sigma'^2_l)$$

where $V \in \overline{X_k}$ and $V \notin \overline{X'_l}$.

(Proof)

The above proposition states that integrated form of a product of Gaussian PDF functions with respect to a variable is a product of Gaussian PDF functions. We first prove it for a simple case involving two standard Gaussian PDF functions, and then generalize it for arbitrary number of Gaussians.

For simplicity, let us first compute the integrated-form of $\mathcal{N}_{V-X_1}(0,1).\mathcal{N}_{V-X_2}(0,1)$ w.r.t. variable V where X_1, X_2 are linear combination of variables (except V). We make the following two assumptions:

1. The coefficient of V is 1 in both PDFs.
2. Both PDFs are standard normal distributions (i.e., $\mu = 0$ and $\sigma^2 = 1$).

Let ϕ denote the integrated form, i.e.,

$$\begin{aligned}
\phi &= \int_{-\infty}^{\infty} \mathcal{N}_{V-X_1}(0,1).\mathcal{N}_{V-X_2}(0,1) dV \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp^{-\frac{(V-X_1)^2}{2}} . \frac{1}{\sqrt{2\pi}} \exp^{-\frac{(V-X_2)^2}{2}} dV \\
&= \int_{-\infty}^{\infty} \frac{1}{2\pi} \exp^{-\frac{1}{2}[(V-X_1)^2+(V-X_2)^2]} dV \\
&= \int_{-\infty}^{\infty} \frac{1}{2\pi} \exp^{-\frac{1}{2}\eta} dV
\end{aligned}$$

Now

$$\begin{aligned}\eta &= (V - X_1)^2 + (V - X_2)^2 \\ &= 2.V^2 - 2.V.(X_1 + X_2) + (X_1^2 + X_2^2) \\ &= 2[(V - \frac{X_1 + X_2}{2})^2 + (\frac{X_1^2 + X_2^2}{2}) - (\frac{X_1 + X_2}{2})^2] \\ &= 2[(V - \frac{X_1 + X_2}{2})^2 + g]\end{aligned}$$

where

$$\begin{aligned}g &= (\frac{X_1^2 + X_2^2}{2}) - (\frac{X_1 + X_2}{2})^2 \\ &= \frac{1}{4}(X_1 - X_2)^2\end{aligned}$$

Thus the integrated form can be expressed as

$$\begin{aligned}\phi &= \int_{-\infty}^{\infty} \frac{1}{2\pi} \exp^{-\frac{1}{2}.2[(V - \frac{X_1+X_2}{2})^2 + g]}\, dV \\ &= \int_{-\infty}^{\infty} \frac{1}{2\pi} \exp^{-\frac{1}{2}.2.(V - \frac{X_1+X_2}{2})^2} . \exp^{-\frac{1}{2}.2.g}\, dV \\ &= \frac{1}{2\sqrt{\pi}} exp^{-g} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2.\pi.\frac{1}{2}}} \exp^{-\frac{(V - \frac{X_1+X_2}{2})^2}{2.\frac{1}{2}}}\, dV \\ &= \frac{1}{2\sqrt{\pi}} exp^{-g} \\ &\texttt{(as integration over the whole area is 1)} \\ &= \frac{1}{\sqrt{2\pi.2}} exp^{-\frac{1}{4}(X_1 - X_2)^2} \\ &= \mathcal{N}_{X_1 - X_2}(0, 2)\end{aligned}$$

Thus integrated form of a PPDF function is another PPDF function. Notice that the integrated form is a constant when $X1 = X2$.

Generalization for arbitrary number of PDFs. Note that for any arbitrary number of PDFs in a PPDF function, $\eta = \sum(V - X_i)^2$ can be always written as $k[(V - \beta)^2 + g_n]$, where

$$g_n = \frac{1}{n} \sum_{i=1}^{n} X_i^2 - \frac{1}{n^2} (\sum_{i=1}^{n} X_i)^2$$

For any arbitrary number of PDFs, we will prove the property on g_n. In other words, we will show that g_n can be expressed as

$$g_n = \frac{1}{n}\sum_{i=1}^{n} X_i^2 - \frac{1}{n^2}(\sum_{i=1}^{n} X_i)^2 = \frac{1}{n^2}\sum_{i\neq j, i<j} (X_i - X_j)^2 \tag{6.4}$$

which means integrated form of n PDFs,

$$\phi_n = \int_{-\infty}^{\infty} \prod_{i=1}^{n} \mathcal{N}_{V-X_i}(0,1)dV$$

can be expressed as

$$\phi_n = \alpha \exp^{-g_n} = \alpha \prod_{i\neq j, i<j} \mathcal{N}_{X_i - X_j}$$

Proposition 11. *Let* $f_n = \sum_{i=1}^{n} X_i$. *Then,*

$$f_n^2 = \sum_{i=1}^{n} X_i^2 + \sum_{i\neq j} X_i X_j.$$

Proof. We prove the proposition using induction. Let us assume that the above equation holds for n variables. Now for $(n+1)^{th}$ variable X_{n+1},

$$\begin{aligned}
f_{n+1}^2 &= (\sum_{i=1}^{n+1} X_i)^2 \\
&= ((\sum_{i=1}^{n} X_i) + X_{n+1})^2 \\
&= (\sum_{i=1}^{n} X_i)^2 + X_{n+1}^2 + 2(X_1 + ... + X_n)X_{n+1} \\
&= \sum_{i=1}^{n} X_i^2 + \sum_{i\neq j, i=1}^{n} X_i X_j + X_{n+1}^2 + 2(X_1 + ... + X_n)X_{n+1} \\
&\texttt{(using induction hypothesis)} \\
&= \sum_{i=1}^{n+1} X_i^2 + \sum_{i\neq j} X_i X_j
\end{aligned}$$

□

Now going back to proving equation 6.4, we first show that g_n can be written in the following form

$$\begin{aligned} g_n &= \frac{1}{n}\sum_{i=1}^{n} X_i^2 - \frac{1}{n^2}(\sum_{i=1}^{n} X_i)^2 \\ &= \frac{1}{n^2}[(n-1)\sum_{i=1}^{n} X_i^2 - \sum_{i\neq j} X_i X_j] \end{aligned}$$

The above equation can be proved by induction. It is easy to see that for $n = 2$ the equation holds, as $g_2 = \frac{1}{2}\sum_{i=1}^{2} X_i^2 - \frac{1}{4}(\sum_{i=1}^{2} X_i)^2 = \frac{1}{4}[X_1^2 + X_2^2 - X_1X_2 - X_2X_1]$. Now

$$\begin{aligned} g_{n+1} &= \frac{1}{(n+1)}\sum_{i=1}^{n+1} X_i^2 - \frac{1}{(n+1)^2} f_{n+1}^2 \\ &= \frac{1}{(n+1)^2}[(n+1)\sum_{i=1}^{n+1} X_i^2 - f_{n+1}^2] \\ &= \frac{1}{(n+1)^2}[(n+1)\sum_{i=1}^{n+1} X_i^2 - \sum_{i=1}^{n+1} X_i^2 - \sum_{i\neq j} X_i X_j] \\ &\texttt{(using Proposition 11)} \\ &= \frac{1}{(n+1)^2}[n\sum_{i=1}^{n+1} X_i^2 - \sum_{i\neq j} X_i X_j] \end{aligned}$$

Thus $g_n = \frac{1}{n^2}[(n-1)\sum_{i=1}^{n} X_i^2 - \sum_{i\neq j} X_i X_j]$.
Finally, we will prove that

$$\begin{aligned} g_n &= \frac{1}{n^2}[(n-1)\sum_{i=1}^{n} X_i^2 - \sum_{i\neq j} X_i X_j] \\ &= \frac{1}{n^2}\sum_{i\neq j, i<j} (X_i - X_j)^2 \end{aligned}$$

Proposition 12. *Let* $h_n = \sum_{i\neq j, i<j}(X_i - X_j)^2$. *Then*

$$h_n = (n-1)\sum_{i=1}^{n} X_i^2 - \sum_{i\neq j} X_i X_j$$

Proof. We use induction to prove the above proposition. Let h_n holds for n

variables. Then for $(n+1)^{th}$ variable,

$$\begin{aligned}
h_{n+1} &= \sum_{i\neq j, i<j} (X_i - X_j)^2 \\
&= h_n + \sum_{i=1}^{n} (X_i - X_{n+1})^2 \\
&= (n-1)\sum_{i=1}^{n} X_i^2 - \sum_{i\neq j, i=1}^{n} X_i X_j + \sum_{i=1}^{n} X_i^2 + nX_{n+1}^2 - 2\sum_{i=1}^{n} X_i X_{n+1} \\
&= n\sum_{i=1}^{n+1} X_i^2 - \sum_{i\neq j, i=1}^{n+1} X_i X_j
\end{aligned}$$

□

Since $g_n = \frac{1}{n^2} h_n$

$$g_n = \frac{1}{n^2} \sum_{i\neq j, i<j} (X_i - X_j)^2$$

Thus ϕ_n can be expressed as

$$\phi_n = \alpha \exp^{-g_n} = \alpha \prod_{i\neq j, i<j} \mathcal{N}_{X_i - X_j}$$

Integrated-form with arbitrary constants: For any arbitrary mean, variance and coefficients of V,

$$\begin{aligned}
\phi_2 &= \int_{-\infty}^{\infty} \mathcal{N}_{a_1 V - X_1}(\mu_1, \sigma_1^2).\mathcal{N}_{a_2 V - X_2}(\mu_2, \sigma_2^2) dV \\
&= \mathcal{N}_{a_2 X_1 - a_1 X_2}(a_1\mu_2 - a_2\mu_1, a_2^2\sigma_1^2 + a_1^2\sigma_2^2)
\end{aligned}$$

Proposition 13 gives a proof of the above equation.

And

$$\phi_n = \alpha \prod_{i\neq j, i<j} \mathcal{N}_{a_j X_i - a_i X_j}(a_i\mu_j - a_j\mu_i, \sigma_{ij}^2)$$

where

$$\sigma_{ij}^2 = \frac{\sum_{k=1}^{n} a_k^2 \prod_{l\neq k, l=1}^{n} \sigma_l^2}{\prod_{k=1, k\neq i,j}^{n} \sigma_k^2}$$

Note that the normalization constant is also adjusted appropriately in the integrated form. The number of PDFs in the resultant PPDF function is quadratic in the number of PDFs in the original PPDF function. If there are n PDFs in the original PPDF function, then there are $\binom{n}{2}$ PDFs in the resultant PPDF function.

Proposition 13.

$$\begin{aligned}\phi_2 &= \int_{-\infty}^{\infty} \mathcal{N}_{a_1V-X_1}(\mu_1,\sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2,\sigma_2^2)dV \\ &= \mathcal{N}_{a_2X_1-a_1X_2}(a_1\mu_2 - a_2\mu_1, a_2^2\sigma_1^2 + a_1^2\sigma_2^2)\end{aligned}$$

Proof.

$$\begin{aligned}\phi_2 &= \int_{-\infty}^{\infty} \mathcal{N}_{a_1V-X_1}(\mu_1,\sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2,\sigma_2^2)dV \\ &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp^{-\frac{(a_1V-X_1-\mu_1)^2}{2\sigma_1^2}} . \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp^{-\frac{(a_2V-X_2-\mu_2)^2}{2\sigma_2^2}} dV \\ &= \int_{-\infty}^{\infty} \frac{1}{2\pi\sigma_1\sigma_2} \exp^{-\frac{[\sigma_2^2(a_1V-X_1-\mu_1)^2+\sigma_1^2(a_2V-X_2-\mu_2)^2]}{2\sigma_1^2\sigma_2^2}} dV \\ &= \int_{-\infty}^{\infty} \frac{1}{2\pi\sigma_1\sigma_2} \exp^{-\frac{1}{2}\eta} dV\end{aligned}$$

where

$$\begin{aligned}\eta &= \frac{\sigma_2^2(a_1V - X_1 - \mu_1)^2 + \sigma_1^2(a_2V - X_2 - \mu_2)^2}{\sigma_1^2\sigma_2^2} \\ &= \tfrac{1}{\sigma_1^2\sigma_2^2}[\sigma_2^2(a_1^2V^2 - 2a_1V(X_1+\mu_1) + (X_1+\mu_1)^2) + \sigma_1^2(a_2^2V^2 - 2a_2V(X_2+\mu_2) + (X_2+\mu_2)^2)] \\ &= \tfrac{1}{\sigma_1^2\sigma_2^2}[(\sigma_2^2a_1^2 + \sigma_1^2a_2^2)V^2 - 2V\{a_1\sigma_2^2(X_1+\mu_1) + a_2\sigma_1^2(X_2+\mu_2)\} + \sigma_2^2(X_1+\mu_1)^2 + \sigma_1^2(X_2+\mu_2)^2] \\ &= \frac{(\sigma_2^2a_1^2 + \sigma_1^2a_2^2)}{\sigma_1^2\sigma_2^2}[V - \frac{\{a_1\sigma_2^2(X_1+\mu_1) + a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2 + \sigma_1^2a_2^2)}]^2 + g\end{aligned}$$

where

$$\begin{aligned}g &= \frac{\sigma_2^2(X_1+\mu_1)^2 + \sigma_1^2(X_2+\mu_2)^2}{\sigma_1^2\sigma_2^2} - \frac{[a_1\sigma_2^2(X_1+\mu_1) + a_2\sigma_1^2(X_2+\mu_2)]^2}{\sigma_1^2\sigma_2^2(\sigma_2^2a_1^2 + \sigma_1^2a_2^2)} \\ &= \frac{1}{(\sigma_2^2a_1^2 + \sigma_1^2a_2^2)}[a_2^2(X_1+\mu_1)^2 + a_1^2(X_2+\mu_2)^2 - 2a_1a_2(X_1+\mu_1)(X_2+\mu_2)]\end{aligned}$$

$$g = \frac{1}{(\sigma_2^2 a_1^2 + \sigma_1^2 a_2^2)}[a_2(X_1+\mu_1) - a_1(X_2+\mu_2)]^2$$
$$= \frac{1}{(\sigma_2^2 a_1^2 + \sigma_1^2 a_2^2)}[(a_2 X_1 - a_1 X_2) - (a_1\mu_2 - a_2\mu_1)]^2$$

Thus

$$\phi_2 = \int_{-\infty}^{\infty} \frac{1}{2\pi\sigma_1\sigma_2} \exp^{-\frac{1}{2}\cdot\frac{(\sigma_2^2 a_1^2+\sigma_1^2 a_2^2)}{\sigma_1^2\sigma_2^2}[V - \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2 a_1^2+\sigma_1^2 a_2^2)}]^2 + g} dV$$
$$= \frac{1}{\sqrt{2\pi(\sigma_2^2 a_1^2 + \sigma_1^2 a_2^2)}} exp^{-\frac{1}{2}g}$$
$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2.\pi.\frac{\sigma_1^2\sigma_2^2}{(\sigma_2^2 a_1^2+\sigma_1^2 a_2^2)}}} \exp^{-\frac{1}{2}\cdot\frac{(\sigma_2^2 a_1^2+\sigma_1^2 a_2^2)}{\sigma_1^2\sigma_2^2}[V - \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2 a_1^2+\sigma_1^2 a_2^2)}]^2} dV$$
$$= \frac{1}{\sqrt{2\pi(\sigma_2^2 a_1^2 + \sigma_1^2 a_2^2)}} exp^{-\frac{1}{2}g}$$
`(as integration over the whole area is 1)`
$$= \frac{1}{\sqrt{2\pi(\sigma_2^2 a_1^2 + \sigma_1^2 a_2^2)}} exp^{-\frac{1}{2}\frac{1}{(\sigma_2^2 a_1^2+\sigma_1^2 a_2^2)}[(a_2 X_1 - a_1 X_2) - (a_1\mu_2 - a_2\mu_1)]^2}$$
$$= \mathcal{N}_{a_2X_1 - a_1X_2}(a_1\mu_2 - a_2\mu_1, a_2^2\sigma_1^2 + a_1^2\sigma_2^2)$$

□

Proposition 7. *The set of all success functions is closed under join and marginalize operations.*

Proof. We first show that success functions are closed under *join* operation.

Definition of success function for base predicates follows the general form of success functions. Now according to the definition of join operation, join of two success functions $\psi_1 = \sum_i \langle D_i, C_i\rangle$ and $\psi_2 = \sum_j \langle D_j, C_j\rangle$ is the success function $\sum_{i,j}\langle D_i D_j, C_i \wedge C_j\rangle$. Product of two sum of terms can be written as another summation of terms (e.g., product of $D_1 + D_2$ and $D_3 + D_4$ is $D_1D_3 + D_2D_3 + D_1D_4 + D_2D_4$). Each PPDF function in the new success function is a product of two PPDFs from ψ_1 and ψ_2, which preserves the general form of PPDF functions. Similarly, the constraint in the new constrained PPDF function is simply a conjunction of the two constraint sets from ψ_1 and ψ_2. Thus the join operation returns a function which follows the general form of success functions.

Next we prove that success functions are closed under *marginalize* operations. Marginalization w.r.t. a discrete variable is done by integrating the delta functions involving that discrete variable.

Marginalization operation w.r.t. a continuous variable involves first doing a projection and then doing an integration. According to the definition of projection operation, it does not alter the general form of success functions. Proposition 6 shows that PPDF functions are closed under integration operation. Thus the set of all success functions are closed under marginalize operation. □

Chapter 7

Learning

We present an EM-based learning algorithm which permits us to learn the distribution parameters of extended PRISM programs with discrete as well as Gaussian random variables. Similar to inference, our learning algorithm uses the *symbolic derivation* procedure to succinctly represent a large set of logical derivations; it also represents and computes Expected Sufficient Statistics (ESS) symbolically. In this chapter, we present our learning algorithm in detail along with illustrative examples on finite mixture models.

7.1 Learning Algorithm

We use the expectation-maximization algorithm [15] to learn the distribution parameters from data. First we show how to compute the expected sufficient statistics (ESS) of the random variables and then describe our algorithm.

The ESS of a discrete random variable is a n-tuple where n is the number of values that the discrete variable takes. Suppose that a discrete random variable V takes $v_1, v_2, ..., v_n$ as values. Then the ESS of V is $(ESS^{V=v_1}, ESS^{V=v_2}, ..., ESS^{V=v_n})$ where $ESS^{V=v_i}$ is the expected number of times variable V had valuation v_i in all possible proofs for a goal. The ESS of a Gaussian random variable X is a triple $(ESS^{X,\mu}, ESS^{X,\sigma^2}, ESS^{X,count})$ where the components denote the expected sum, expected sum of squares and the expected number of uses of random variable X, respectively. When derivations are enumerated, the ESS for each random variable can be represented by a tuple of reals. To accommodate *symbolic* derivations, we lift each component of ESS to a function, represented as described below.

Representation of ESS functions: For each component ν (discrete variable valuation, mean, variance, total counts) of a random variable, its *ESS*

function in a goal G is represented as follows:

$$\xi^{\nu}_{G} = \sum_{i} \langle \chi^{\nu}_{i} \phi_i, C_i \rangle.$$

where $\langle \phi_i, C_i \rangle$ is a constrained PPDF function and

$$\chi^{\nu}_{i} = \begin{cases} \overline{a}_i \cdot \overline{X}_i + b_i & \text{if } \nu = X, \mu \\ \overline{a}_i \cdot \overline{X}^2_i + b_i & \text{if } \nu = X, \sigma^2 \\ b_i & \text{otherwise} \end{cases}$$

Here $\overline{a}_i, b_i$ are constants, and $\overline{X}_i = V_c(G)$.

Expressions of the form $\chi_i \phi_i$ are called *EPDF* functions and a pair $\langle \chi_i \phi_i, C \rangle$ is called a *constrained EPDF function*. A sum of a finite number of constrained EPDF functions is called an *ESS function*. We use $C_i(\xi)$ to denote the constraints (i.e., C_i) in the i^{th} constrained EPDF function of ESS function ξ; and $E_i(\xi)$ to denote the i^{th} EPDF function of ξ. Note that the representation of ESS functions is same as that of success functions for discrete random variable valuations and total counts.

ESS functions of base predicates: The ESS function of the i^{th} parameter of a discrete random variable V is $P(V = v_i)\delta_{v_i}(V)$. The ESS function of the mean of a continuous random variable X is $X\mathcal{N}_X(\mu, \sigma^2)$, and the ESS function of the variance of a continuous random variable X is $X^2\mathcal{N}_X(\mu, \sigma^2)$. Finally, the ESS function of the total count of a continuous random variable X is $\mathcal{N}_X(\mu, \sigma^2)$.

Example 37. *In this example, we compute the ESS functions of the random variables (*`m`*,* `w(a)`*, and* `w(b)`*) in Example 34. According to the definition of ESS function of base predicates, the ESS functions of these random variables for goals* $msw(m, M)$ *and* $msw(w(M), X)$ *are*

ESS	*for* $msw(m, M)$	*for* $msw(w(M), X)$
$\xi^{M=a}$	$p_a\delta_a(M)$	0
$\xi^{M=b}$	$p_b\delta_b(M)$	0
ξ^{X,μ_a}	0	$X\delta_a(M)\mathcal{N}_X(\mu_a, \sigma^2_a)$
ξ^{X,μ_b}	0	$X\delta_b(M)\mathcal{N}_X(\mu_b, \sigma^2_b)$
ξ^{X,σ^2_a}	0	$X^2\delta_a(M)\mathcal{N}_X(\mu_a, \sigma^2_a)$
ξ^{X,σ^2_b}	0	$X^2\delta_b(M)\mathcal{N}_X(\mu_b, \sigma^2_b)$
$\xi^{X,count_a}$	0	$\delta_a(M)\mathcal{N}_X(\mu_a, \sigma^2_a)$
$\xi^{X,count_b}$	0	$\delta_b(M)\mathcal{N}_X(\mu_b, \sigma^2_b)$

□

ESS functions of user-defined predicates: If $G \rightarrow G'$ is a step in a derivation, then the ESS function of a random variable for G is computed bottom-up based on the its ESS function for G'. This computation is described below.

Join and *Marginalize* operations, defined earlier for success functions, can be readily extended for ESS functions as well. The computation of ESS functions for a goal, based on the symbolic derivation, uses the extended *join* and *marginalize* operations defined below.

Unlike the join operation over pairs of success functions, the join operation used in ESS computation is defined over pairs of ESS and success functions. The join operation returns an ESS function.

Definition 13 (Join)**.** *Let $\psi_1 = \sum_i \langle D_i, C_i \rangle$ be a success function and $\xi_2 = \sum_j \langle E_j, C'_j \rangle$ be an ESS function, then join of ψ_1 and ξ_2 represented as $\psi_1 * \xi_2$ is the ESS function $\sum_{i,j} \langle D_i E_j, C_i \wedge C'_j \rangle$.*

Marginalization w.r.t. a continuous variables involves two steps: first rewriting the ESS function in a projected form and then doing the required integration.

Definition 14 (Projection)**.** *Projection of an ESS function ξ w.r.t. a continuous variable V, denoted by $\xi \downarrow_V$, is an ESS function ξ' such that*
$\forall i.\ E_i(\xi') = E_i(\xi)[\overline{a} \cdot \overline{X} + b/V]$; and
$C_i(\xi') = (C_i(\xi) - C_{ip})[\overline{a} \cdot \overline{X} + b/V]$,
where C_{ip} is a linear constraint $(V = \overline{a} \cdot \overline{X} + b)$ on V in $C_i(\xi)$ and $t[s/x]$ denotes replacement of all occurrences of x in t by s.

Note that the replacement of V by $\overline{a} \cdot \overline{X} + b$ in PDFs and linear constraints does not alter the general form of ESS functions. Thus projection returns an ESS function. Notice that if ξ does not contain any linear constraint on V, then the projected form remains the same.

Proposition 14. *Integration of an EPDF function with respect to a variable V is an EPDF function, i.e.,*

$$\alpha \int_{-\infty}^{\infty} \chi \prod_{k=1}^{m} \mathcal{N}_{(\overline{a_k} \cdot \overline{X_k} + b_k)}(\mu_k, \sigma_k^2) dV = \alpha' \chi' \prod_{l=1}^{m'} \mathcal{N}_{(\overline{a'_l} \cdot \overline{X'_l}) + b'_l}(\mu'_l, \sigma'^2_l)$$

where $V \in \overline{X_k}$, $V \in \chi$ and $V \notin \overline{X'_l}$, $V \notin \chi'$.

Proof of this proposition is presented in Section 7.5.

For example, the integration of $V\mathcal{N}_{a_1V-X_1}(\mu_1,\sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2,\sigma_2^2)$ w.r.t. variable V is

$$\int_{-\infty}^{\infty} V\mathcal{N}_{a_1V-X_1}(\mu_1,\sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2,\sigma_2^2)dV$$
$$= \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}\mathcal{N}_{a_2X_1-a_1X_2}(a_1\mu_2-a_2\mu_1,a_2^2\sigma_1^2+a_1^2\sigma_2^2). \tag{7.1}$$

Here X_1, X_2 are linear combinations of variables (except V). Equation 7.1 is an instance of Proposition 14.

Definition 15 (Integration)**.** *Let ξ be an ESS function that does not contain any linear constraints on V. Then integration of ξ with respect to V, denoted by $\oint_V \xi$ is an ESS function ψ' such that $\forall i.E_i(\psi') = \int E_i(\psi)dV$.*

Note that if ξ does not contain any PDF on V, then the integrated form remains the same.

Example 38. *Let ξ_1 represent the following ESS function*

$$\xi_1 = 0.3Z\mathcal{N}_Z(2.0,1.0).\mathcal{N}_{X-Z}(0.5,0.1).$$

Then integration of ξ_2 w.r.t. Z yields

$$\oint_Z \xi_1 = \int 0.3Z\mathcal{N}_Z(2.0,1.0).\mathcal{N}_{X-Z}(0.5,0.1)dZ$$
$$= 0.15(X+2.5)\mathcal{N}_X(2.5,1.1). \tag{7.2}$$
(using Equation 7.1)

□

Definition 16 (Marginalize)**.** *Marginalization of an ESS function ξ with respect to a variable V, denoted by $\mathbb{M}(\xi,V)$, is an ESS function ξ' such that*

$$\xi' = \oint_V \xi \downarrow_V$$

We overload $\mathbb{M}$ to denote marginalization over a set of variables, defined such that $\mathbb{M}(\xi,\{V\}\cup\overline{X}) = \mathbb{M}(\mathbb{M}(\xi,V),\overline{X})$ and $\mathbb{M}(\xi,\{\}) = \xi$.

Proposition 15. *The set of all ESS functions is closed under the extended* `Join` *and* `Marginalize` *operations.*

Proof. The proof of this proposition depends on Proposition 14 and closely follows the proof of the closure of success functions (Proposition 7). □

The ESS function of a random variable component (discrete variable valuation, mean, variance, total counts) in a derivation is defined as follows.

Definition 17 (ESS functions in a derivation)**.** *Let $G \to G'$. Then the ESS function of a random variable component ν in the goal G, denoted by ξ^{ν}_{G}, is computed from that of G', based on the way G' was derived:*

PCR: $\xi^{\nu}_{G} = \sum_{G'} \mathbb{M}(\xi^{\nu}_{G'}, V(G') - V(G))$.

MSW: *Let* $G = \mathtt{msw}(\mathrm{rv}(\overline{X}), Y), G_1$. *Then* $\xi^{\nu}_{G} = \psi_{msw(rv(\overline{X}),Y)} * \xi^{\nu}_{G'} + \psi_{G'} * \xi^{\nu}_{msw}$.

CONS: *Let* $G = Constr, G_1$. *Then* $\xi^{\nu}_{G} = \psi_{Constr} * \xi^{\nu}_{G'}$.

Example 39. *Using the definition of ESS function of a derivation involving MSW, we compute the ESS function of the random variables in goal G_2 of Figure 7.1.*

	ESS functions for goal G_2
$\xi^{M=a}$	$p_a\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2)$
$\xi^{M=b}$	$p_b\delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)$
ξ^{X,μ_a}	$Xp_a\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2)$
ξ^{X,μ_b}	$Xp_b\delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)$
ξ^{X,σ_a^2}	$X^2p_a\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2)$
ξ^{X,σ_b^2}	$X^2p_b\delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)$
$\xi^{X,count_a}$	$p_a\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2)$
$\xi^{X,count_b}$	$p_b\delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)$

Notice the way $\xi^{M=a}_{G_2}$ is computed.

$$\begin{aligned}
\xi^{M=a}_{G_2} &= \psi_{msw(m,M)}\xi^{M=a}_{G_3} + \psi_{G_3}\xi^{M=a}_{msw(m,M)} \\
&= [p_a\delta_a(M) + p_b\delta_b(M)].0 \\
&+ [\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2) + \delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)].p_a\delta_a(M) \\
&= p_a\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2)
\end{aligned}$$

Finally, for goal G_1 we marginalize the ESS functions w.r.t. M.

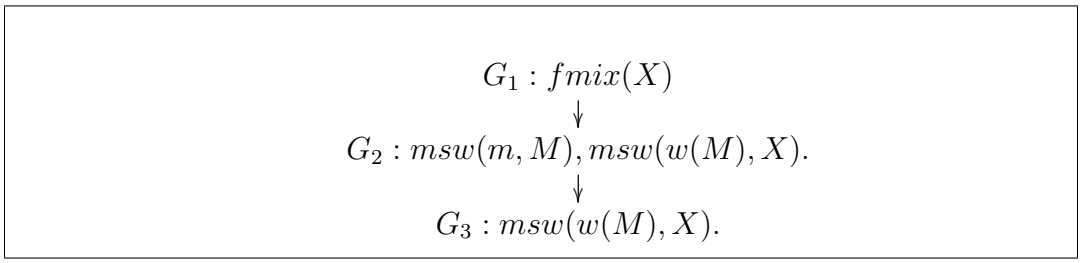


Figure 7.1: Symbolic derivation for goal `fmix(X)`

	ESS functions for goal G_1
$\xi^{M=a}$	$p_a \mathcal{N}_X(\mu_a, \sigma_a^2)$
$\xi^{M=b}$	$p_b \mathcal{N}_X(\mu_b, \sigma_b^2)$
ξ^{X,μ_a}	$X p_a \mathcal{N}_X(\mu_a, \sigma_a^2)$
ξ^{X,μ_b}	$X p_b \mathcal{N}_X(\mu_b, \sigma_b^2)$
ξ^{X,σ_a^2}	$X^2 p_a \mathcal{N}_X(\mu_a, \sigma_a^2)$
ξ^{X,σ_b^2}	$X^2 p_b \mathcal{N}_X(\mu_b, \sigma_b^2)$
$\xi^{X,count_a}$	$p_a \mathcal{N}_X(\mu_a, \sigma_a^2)$
$\xi^{X,count_b}$	$p_b \mathcal{N}_X(\mu_b, \sigma_b^2)$

□

Example 40. *Figure 7.1 shows the symbolic derivation for goal* `fmix(X)` *over the finite mixture model program in Example 34. Success function of goal* G_3 *is* $\psi_{msw(w(M),X)}(M, X)$*, hence* $\psi_{G_3} = \delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2) + \delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)$*.*

ψ_{G_2} *is* $\psi_{msw(m,M)}(M) * \psi_{G_3}(M, X)$ *which yields* $\psi_{G_2} = p_a\delta_a(M)\mathcal{N}_X(\mu_a, \sigma_a^2) + p_b\delta_b(M)\mathcal{N}_X(\mu_b, \sigma_b^2)$*. Note that* $\delta_b(M)\delta_a(M) = 0$ *as* M *can not be both* a *and* b *at the same time. Also* $\delta_a(M)\delta_a(M) = \delta_a(M)$*.*

Finally, $\psi_{G_1} = \mathbb{M}(\psi_{G_2}, M)$ *which is* $p_a\mathcal{N}_X(\mu_a, \sigma_a^2) + p_b\mathcal{N}_X(\mu_b, \sigma_b^2)$*. Note that* ψ_{G_1} *represents the mixture distribution [38] of mixture of two Gaussian distributions.*

Here $p_a = 0.3, p_b = 0.7, \mu_a = 2.0, \mu_b = 3.0$*, and* $\sigma_a^2 = \sigma_b^2 = 1.0$*.* □

The algorithm for learning distribution parameters (Θ) uses a fixed set of training examples $(t_1, t_2, ..., t_N)$. Note that the success and ESS functions for t_i's are constants as the training examples are variable free (i.e., all the variables get marginalized over).

Algorithm 1 (Expectation-Maximization)**.** *Initialize the distribution parameters* Θ*.*

1. *Evaluate the log likelihood,* $\lambda = \ln P(t_1, .., t_N|\Theta) = \sum_i \ln \psi_{t_i}$*.*
2. *For each training example* t_i *(*$1 \leq i \leq N$*), construct the symbolic derivations using current* Θ*.*

3. ***E-step:*** *Compute the ESS (ξ_{t_i}) of the random variables, and success probabilities ψ_{t_i} w.r.t. Θ.*
M-step: *Compute the MLE of the distribution parameters given the ESS and success probabilities (i.e., evaluate Θ'). Θ' contains updated distribution parameters. More specifically, for a discrete random variable V, its parameters are updated as follows:*

$$p'_{V=v} = \frac{\eta_{V=v}}{\sum_{u \in values(V)} \eta_{V=u}}$$

where

$$\eta_{V=v} = \sum_{i=1}^{N} \frac{\xi_{t_i}^{V=v}}{\psi_{t_i}}.$$

For each continuous random variable X, its mean and variances are updated as follows:

$$\mu'_X = \frac{\sum_{i=1}^{N} \frac{\xi_{t_i}^{X,\mu}}{\psi_{t_i}}}{N_X}$$

$$\sigma'^2_X = \frac{\sum_{i=1}^{N} \frac{\xi_{t_i}^{X,\sigma^2}}{\psi_{t_i}}}{N_X} - \mu'^2_X$$

where N_X is the expected total count of X.

$$N_X = \sum_{i=1}^{N} \frac{\xi_{t_i}^{X,count}}{\psi_{t_i}}$$

4. *Evaluate the log likelihood ($\ln P(t_1, .., t_N|\Theta') = \sum_i \ln \psi_{t_i}$) and check for convergence. Otherwise let $\Theta \leftarrow \Theta'$ and go to step 2.*

Example 41. *Let $x_1, x_2, ..., x_N$ be the observations. For a given training example $t_i = fmix(x_i)$, the ESS functions are (using Example 39)*

	ESS functions for goal $fmix(x_i)$
$\xi^{M=k}$	$p_k \mathcal{N}_X(x_i\|\mu_k, \sigma_k^2)$
ξ^{X,μ_k}	$x_i p_k \mathcal{N}_X(x_i\|\mu_k, \sigma_k^2)$
ξ^{X,σ_k^2}	$x_i^2 p_k \mathcal{N}_X(x_i\|\mu_k, \sigma_k^2)$
$\xi^{X,count_k}$	$p_k \mathcal{N}_X(x_i\|\mu_k, \sigma_k^2)$

where $k \in \{a, b\}$.

The E-step of the EM algorithm involves computation of the above ESS functions.

In the M-step, we update the model parameters from the computed ESS functions.

$$p'_k = \frac{\sum_{i=1}^{N} \frac{\xi^k_{t_i}}{\psi_{t_i}}}{\sum_{i=1}^{N} \frac{\xi^a_{t_i}}{\psi_{t_i}} + \frac{\xi^b_{t_i}}{\psi_{t_i}}} = \frac{\sum_{i=1}^{N} \frac{\xi^k_{t_i}}{\psi_{t_i}}}{N} \tag{7.3}$$

Similarly,

$$\mu'_k = \frac{\sum_{i=1}^{N} \frac{\xi^{\mu_k}_{t_i}}{\psi_{t_i}}}{N_k} \tag{7.4}$$

$$\sigma'^2_k = \frac{\sum_{i=1}^{N} \frac{\xi^{\sigma^2_k}_{t_i}}{\psi_{t_i}}}{N_k} - \mu'^2_k \tag{7.5}$$

where $k \in \{a, b\}$ *and*

$$N_k = \sum_{i=1}^{N} \frac{\xi^{count_k}_{t_i}}{\psi_{t_i}} \tag{7.6}$$

□

7.2 Correctness of the Learning Algorithm

Theorem 16. *Algorithm 1 correctly computes the MLE of the distribution parameters which are used to maximize the likelihood of the data.*

Proof. The main routine of Algorithm 1 for discrete case is same as the *learn-naive* algorithm of [55], except the computation of $\eta_{V=v}$ (Equation 4.1).

$$\eta_{V=v} = \sum_{\text{for each goal } g} \frac{1}{\psi_g} \sum_{S} P(S) N^v_S.$$

where S is an explanation for goal g and N^v_S is the total number of times $V = v$ in S.

We show that $\xi^{V=v}_g = \sum_S P(S) N^v_S$.

Let the goal g has a single explanation S where S is a conjunction of subgoals (i.e., $S_{1:n} = g_1, g_2, ..., g_n$). Thus we need to show that $\xi^{V=v}_g = P(S) N^v_S$.

We prove this by induction on the length of S. The definition of ξ for base predicates gives the desired result for $n = 1$. Let the above equation holds for length n i.e., $\xi^{V=v}_{g_{1:n}} = P(S_{1:n})N_{1:n}$. For $S_{1:n+1} = g_1, g_2, ..., g_n, g_{n+1}$,

$$\begin{aligned}
P(S_{1:n+1})N_{1:n+1} &= P(g_1, g_2, ..., g_n, g_{n+1})N_{1:n+1} \\
&= P(g_1,g_2,...,g_n)P(g_{n+1})(N_{1:n}+N_{n+1}) \\
&= P(S_{1:n})P(g_{n+1})N_{1:n}+P(S_{1:n})P(g_{n+1})N_{n+1} \\
&= P(g_{n+1})[P(S_{1:n})N_{1:n}]+P(S_{1:n})[P(g_{n+1})N_{n+1}] \\
&= P(g_{n+1})\xi^{V=v}_{g_{1:n}} + P(S_{1:n})\xi^{V=v}_{g_{n+1}} \\
&= \xi^{V=v}_{g_{1:n+1}}
\end{aligned}$$

The last step follows from the definition of ξ in a derivation.

Now based on the exclusiveness assumption, for disjunction (or multiple explanations) like $g = g_1 \vee g_2$ it trivially follows that $\xi^{V=v}_g = \xi^{V=v}_{g_1} + \xi^{V=v}_{g_2}$. □

EM for mixture model: The EM algorithm for standard mixture model is the following [4].

Algorithm 2 (EM for mixture model)**.** *Initialize the model parameters (mean μ_k, variance σ^2_k, mixing coefficient p_k).*
E-step: *Compute the posterior probabilities of each component.*

$$\gamma^i_k = \frac{p_k \mathcal{N}_k(x_i|\mu_k, \sigma^2_k)}{\sum_l p_l \mathcal{N}_l(x_i|\mu_l, \sigma^2_l)}$$

M-step: *Re-estimate the model parameters using the posterior probabilities.*

$$\mu_k = \frac{\sum_{i=1}^N x_i \gamma^i_k}{\sum_{i=1}^N \gamma^i_k} \tag{7.7}$$

$$\sigma^2_k = \frac{\sum_{i=1}^N (x_i - \mu_k)^2 \gamma^i_k}{\sum_{i=1}^N \gamma^i_k} \tag{7.8}$$

$$p_k = \frac{\sum_{i=1}^N \gamma^i_k}{N} \tag{7.9}$$

Update the log likelihood and repeat the above steps until convergence.

Example 42. *This example illustrates that for the mixture model example, our ESS computation does the same computation as standard EM learning*

algorithm for mixture models [4].

Notice that for Equation 7.3, $\frac{\xi^k_{t_i}}{\psi_{t_i}} = \frac{p_k \mathcal{N}_k(x_i|\mu_k,\sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i|\mu_l,\sigma_l^2)}$ *which is nothing but the posterior responsibilities* γ^i_k *(here* k *is either* a *or* b*).*

$$\begin{aligned} p'_k &= \frac{\sum_{i=1}^N \frac{\xi^k_{t_i}}{\psi_{t_i}}}{N} \\ &= \frac{\sum_{i=1}^N \frac{p_k \mathcal{N}_X(x_i|\mu_k,\sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i|\mu_l,\sigma_l^2)}}{N} \\ &= \frac{\sum_{i=1}^N \gamma^i_k}{N} \end{aligned}$$

Thus Equation 7.3 computes the same quantity as Equations 7.9.

Similarly for Equation 7.4, $\frac{\xi^{\mu_k}_{t_i}}{\psi_{t_i}} = \frac{x_i p_k \mathcal{N}_k(x_i|\mu_k,\sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i|\mu_l,\sigma_l^2)}$ *which equals* $x_i\gamma^i_k$ *of Equation 7.7.*

$$\begin{aligned} \mu'_k &= \frac{\sum_{i=1}^N \frac{\xi^{\mu_k}_{t_i}}{\psi_{t_i}}}{N_k} \\ &= \frac{\sum_{i=1}^N \frac{x_i p_k \mathcal{N}_X(x_i|\mu_k,\sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i|\mu_l,\sigma_l^2)}}{\sum_{i=1}^N \frac{p_k \mathcal{N}_X(x_i|\mu_k,\sigma_k^2)}{\sum_l p_l \mathcal{N}_l(x_i|\mu_l,\sigma_l^2)}} \\ &= \frac{\sum_{i=1}^N x_i\gamma^i_k}{\sum_{i=1}^N \gamma^i_k}. \end{aligned}$$

Variances are updated similarly. □

7.3 Complexity Analysis

Complexity: Let S_i denote the number of constrained EPDF terms in ξ_i; P_i denote the maximum number of product terms in any EPDF function in ξ_i; Q_i denote the maximum size of a constraint set (C_i) in ξ_i; and L_i denote the maximum number of terms in any χ_i in ξ_i. The time complexity of the two basic operations used in constructing a symbolic derivation is as follows.

Proposition 17 (Time Complexity)**.** *The worst-case time complexity of* $Join(\xi_i, \psi_j)$ *is* $O(S_i * S_j * (P_i * P_j + Q_i * Q_j))$.

The worst-case time complexity of $\mathbb{M}(\xi_g, V)$ *is* $O(S_g * P_g)$ *when* V *is discrete and* $O(S_g * (P_g * L_g + Q_g))$ *when* V *is continuous.*

Note that when computing the ESS function of a goal in a derivation, the join operation is limited to joining the success function of a single `msw` to

the ESS function of a goal and joining the ESS function of a single `msw` to the success function of a goal, and hence the parameters S_i, P_i, and Q_i are typically small. The complexity of the *size* of ESS functions is as follows.

Proposition 18 (ESS Function Size)**.** *For a random variable component ν in the goal G and its symbolic derivation, the following hold:*

1. *The maximum number of product terms in any EPDF function in ξ_G^ν is linear in $|V_c(G)|$, the number of continuous variables in G.*
2. *The maximum size of a constraint set in a constrained EPDF function in ξ_G^ν is linear in $|V_c(G)|$.*
3. *The maximum number of constrained EPDF functions in any entry of ξ_G^ν is potentially exponential in the number of discrete random variables in the symbolic derivation.*

Proof. The proof closely follows the proof of Proposition 10. □

Relation with PRISM's g-EM algorithm: Recall that (in Section 6.3) for programs with only discrete random variables, there is a one-to-one correspondence between the entries in the tabular representation of success functions and PRISM's answer tables. Instead of constructing a symbolic derivation for discrete variables, it is possible to construct the next step of a derivation by enumerating the values of discrete variables (similar to PRISM), i.e., creating different paths in a derivation based on discrete variable values and constructing symbolic derivation only for continuous variables (Figure 6.5). Similar to g-EM, we can use the idea of tabling to eliminate redundant computation. With this simplification, the complexity of the learning algorithm reduces to PRISM's graphical-EM algorithm for program with only discrete random variables.

7.4 Implementation

We implemented the learning algorithm presented in this thesis using the XSB logic programming system [59]. This proof-of-concept prototype is implemented as a meta-interpreter and currently supports discrete and Gaussian distributions. This implementation illustrates how the learning algorithm specializes to the specialized learning algorithms for several popular statistical models such as Hidden Markov Models and Finite Mixture Models.

7.5 Closure of ESS functions: Proof of Proposition 14

Property 19. *Integrated form of an ESS(mean) function with respect to a variable V is an ESS(mean) function, i.e.,*

$$\int_{-\infty}^{\infty}(\alpha V+\beta)\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV$$
$$=\alpha'\prod_{l=1}^{m'}\mathcal{N}_{(\overline{a'_l}\cdot\overline{X'_l})+b'_l}(\mu'_l,\sigma'^2_l)+\beta'\prod_{l=1}^{m''}\mathcal{N}_{(\overline{a''_l}\cdot\overline{X''_l})+b''_l}(\mu''_l,\sigma''^2_l)$$

where $V\in\overline{X_k}$, α is a constant and β is a linear combination of variables or constants except V.

(Proof)

The above integral can be written in the following form

$$\begin{aligned}I&=\int_{-\infty}^{\infty}(\alpha V+\beta)\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV\\&=\int_{-\infty}^{\infty}\alpha V\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV+\int_{-\infty}^{\infty}\beta\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV\\&=I_1+I_2\end{aligned}$$

where

$$I_1=\int_{-\infty}^{\infty}\alpha V\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV$$

and

$$I_2=\int_{-\infty}^{\infty}\beta\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV$$

Using Property 6.7, it can be proved that

$$I_2 = \int_{-\infty}^{\infty} \beta \prod_{k=1}^{m} \mathcal{N}_{(\overline{a_k}.\overline{X_k}+b_k)}(\mu_k, \sigma_k^2) dV$$
$$= \beta' \prod_{l=1}^{m''} \mathcal{N}_{(\overline{a_l''}.\overline{X_l''})+b_l''}(\mu_l'', \sigma_l''^2)$$

Therefore it is sufficient to prove the following equation

$$I_1 = \int_{-\infty}^{\infty} \alpha V \prod_{k=1}^{m} \mathcal{N}_{(\overline{a_k}.\overline{X_k}+b_k)}(\mu_k, \sigma_k^2) dV$$
$$= \alpha' \prod_{l=1}^{m'} \mathcal{N}_{(\overline{a_l'}.\overline{X_l'})+b_l'}(\mu_l', \sigma_l'^2)$$

For simplicity, let us first compute the integrated-form of $V\mathcal{N}_{V-X_1}(0,1).\mathcal{N}_{V-X_2}(0,1)$ w.r.t. variable V where X_1, X_2 are linear combination of variables (except V). We make the following two assumptions:

1. The coefficient of V is 1 in both PDFs.
2. Both PDFs are standard normal distributions (i.e., $\mu = 0$ and $\sigma^2 = 1$).

$$\begin{aligned}
\phi &= \int_{-\infty}^{\infty} V.\mathcal{N}_{V-X_1}(0,1).\mathcal{N}_{V-X_2}(0,1) dV \\
&= \int_{-\infty}^{\infty} V.\frac{1}{\sqrt{2\pi}} \exp^{-\frac{(V-X_1)^2}{2}} . \frac{1}{\sqrt{2\pi}} \exp^{-\frac{(V-X_2)^2}{2}} dV \\
&= \int_{-\infty}^{\infty} V.\frac{1}{2\pi} \exp^{-\frac{1}{2}[(V-X_1)^2+(V-X_2)^2]} dV \\
&= \int_{-\infty}^{\infty} V.\frac{1}{2\pi} \exp^{-\frac{1}{2}\eta} dV
\end{aligned}$$

Now

$$\begin{aligned}
\eta &= (V-X_1)^2 + (V-X_2)^2 \\
&= 2.V^2 - 2.V.(X_1+X_2) + (X_1^2+X_2^2) \\
&= 2[(V - \frac{X_1+X_2}{2})^2 + (\frac{X_1^2+X_2^2}{2}) - (\frac{X_1+X_2}{2})^2] \\
&= 2[(V - \frac{X_1+X_2}{2})^2 + g]
\end{aligned}$$

where

$$\begin{aligned} g &= (\frac{X_1^2 + X_2^2}{2}) - (\frac{X_1 + X_2}{2})^2 \\ &= \frac{1}{4}(X_1 - X_2)^2 \end{aligned}$$

Thus

$$\begin{aligned} \phi &= \int_{-\infty}^{\infty} V.\frac{1}{2\pi} \exp^{-\frac{1}{2}.2[(V-\frac{X_1+X_2}{2})^2+g]}\, dV \\ &= \int_{-\infty}^{\infty} V.\frac{1}{2\pi} \exp^{-\frac{1}{2}.2.(V-\frac{X_1+X_2}{2})^2} . \exp^{-\frac{1}{2}.2.g}\, dV \\ &= \frac{1}{2\sqrt{\pi}} exp^{-g} \int_{-\infty}^{\infty} V.\frac{1}{\sqrt{2.\pi.\frac{1}{2}}} \exp^{-\frac{(V-\frac{X_1+X_2}{2})^2}{2.\frac{1}{2}}}\, dV \end{aligned}$$

Now $\int_{-\infty}^{\infty} V.\frac{1}{\sqrt{2.\pi.\frac{1}{2}}} \exp^{-\frac{(V-\frac{X_1+X_2}{2})^2}{2.\frac{1}{2}}}\, dV$ is nothing but the expectation of normal random variable V whose distribution is $\mathcal{N}_V(\frac{X_1+X_2}{2}, \frac{1}{2})$. Since the expectation of a normal random variable is its mean, $\int_{-\infty}^{\infty} V.\frac{1}{\sqrt{2.\pi.\frac{1}{2}}} \exp^{-\frac{(V-\frac{X_1+X_2}{2})^2}{2.\frac{1}{2}}}\, dV = \frac{X_1+X_2}{2}$. Thus

$$\begin{aligned} \phi &= \frac{X_1 + X_2}{2} \frac{1}{2\sqrt{\pi}} exp^{-g} \\ &= \frac{X_1 + X_2}{2} \frac{1}{\sqrt{2\pi.2}} exp^{-\frac{1}{4}(X_1-X_2)^2} \\ &= \frac{X_1 + X_2}{2} \mathcal{N}_{X_1-X_2}(0, 2) \end{aligned}$$

Thus integrated form of an ESS(mean) function is another ESS(mean) function.

Now for any arbitrary number of PDFs, the mean is $\frac{1}{n}\sum_{i=1}^{n} X_i$. Similar to Property 6.7, it can be proved that the integrated form can be expressed as

$$\phi_n = \alpha \frac{1}{n} \sum_{i=1}^{n} X_i \prod_{i \neq j, i<j} \mathcal{N}_{X_i - X_j}$$

We can lift our assumptions as any arbitrary normal pdf can be expressed as standard normal distribution with a linear transformation on parameters

(Property 1).

For any arbitrary mean, variance and coefficients of V, the following proposition holds.

Proposition 20. *This Proposition is an instance of Proposition 14 which states that the integrated form of an ESS(mean) function with respect to a variable V is another ESS(mean) function.*

$$\int_{-\infty}^{\infty} V\mathcal{N}_{a_1V-X_1}(\mu_1,\sigma_1^2).\mathcal{N}_{a_2V-X_2}(\mu_2,\sigma_2^2)dV$$
$$= \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}\mathcal{N}_{a_2X_1-a_1X_2}(a_1\mu_2-a_2\mu_1,a_2^2\sigma_1^2+a_1^2\sigma_2^2).$$

Proof. It can be proved along the same lines on the proof of Hypothesis 13. Using similar approach, we get

$$\phi_2 = \int_{-\infty}^{\infty} V\frac{1}{2\pi\sigma_1\sigma_2}\exp^{-\frac{1}{2}.\frac{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}{\sigma_1^2\sigma_2^2}[V-\frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}]^2+g}dV$$
$$= \frac{1}{\sqrt{2\pi(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}}exp^{-\frac{1}{2}g}$$
$$\int_{-\infty}^{\infty} V\frac{1}{\sqrt{2.\pi.\frac{\sigma_1^2\sigma_2^2}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}}}\exp^{-\frac{1}{2}.\frac{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}{\sigma_1^2\sigma_2^2}[V-\frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}]^2}dV$$

Now the integral is nothing but the expectation of the normal random variable V. Since the expectation of a normal random variable is its mean, the above integral simplifies to $\frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}$.

$$\phi_2 = \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}\frac{1}{\sqrt{2\pi(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}}exp^{-\frac{1}{2}g}$$
$$= \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}$$
$$\frac{1}{\sqrt{2\pi(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}}exp^{-\frac{1}{2}\frac{1}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}[(a_2X_1-a_1X_2)-(a_1\mu_2-a_2\mu_1)]^2}$$
$$= \frac{\{a_1\sigma_2^2(X_1+\mu_1)+a_2\sigma_1^2(X_2+\mu_2)\}}{(\sigma_2^2a_1^2+\sigma_1^2a_2^2)}\mathcal{N}_{a_2X_1-a_1X_2}(a_1\mu_2-a_2\mu_1,a_2^2\sigma_1^2+a_1^2\sigma_2^2)$$

□

Property 21. *Integrated form of an ESS(variance) function with respect to a variable V is an ESS(variance) function, i.e.,*

$$\int_{-\infty}^{\infty}(\alpha V^2+\beta)\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV$$
$$=\alpha'\prod_{l=1}^{m'}\mathcal{N}_{(\overline{a_l'}\cdot\overline{X_l'})+b_l'}(\mu_l',\sigma_l'^2)+\sum\beta'\prod_{l=1}^{m''}\mathcal{N}_{(\overline{a_l''}\cdot\overline{X_l''})+b_l''}(\mu_l'',\sigma_l''^2)$$

where $V\in\overline{X_k}$, α is a constant and β is a linear/quadratic combination of variables/constants.

(Proof)

The above integral can be written in the following form

$$\begin{aligned}I&=\int_{-\infty}^{\infty}(\alpha V^2+\beta)\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV\\&=\int_{-\infty}^{\infty}\alpha V^2\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV+\int_{-\infty}^{\infty}\beta\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV\\&=I_1+I_2\end{aligned}$$

where

$$I_1=\int_{-\infty}^{\infty}\alpha V^2\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV$$

and

$$I_2=\int_{-\infty}^{\infty}\beta\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV$$

Note that β is in the form of $aV+b$. Thus using Property 19, it can be proved that

$$I_2=\int_{-\infty}^{\infty}\beta\prod_{k=1}^{m}\mathcal{N}_{(\overline{a_k}\cdot\overline{X_k}+b_k)}(\mu_k,\sigma_k^2)dV=\sum\beta'\prod_{l=1}^{m''}\mathcal{N}_{(\overline{a_l''}\cdot\overline{X_l''})+b_l''}(\mu_l'',\sigma_l''^2)$$

Therefore it is sufficient to prove the following equation

$$\begin{aligned} I_1 &= \int_{-\infty}^{\infty} \alpha V^2 \prod_{k=1}^{m} \mathcal{N}_{(\overline{a_k}.\overline{X_k}+b_k)}(\mu_k, \sigma_k^2) dV \\ &= \alpha' \prod_{l=1}^{m'} \mathcal{N}_{(\overline{a'_l}.\overline{X'_l})+b'_l}(\mu'_l, \sigma'^2_l) \end{aligned}$$

For simplicity, let us first compute the integrated-form of $V^2\mathcal{N}_{V-X_1}(0,1).\mathcal{N}_{V-X_2}(0,1)$ w.r.t. variable V where X_1, X_2 are linear combination of variables (except V). We make the following two assumptions:

1. The coefficient of V is 1 in both PDFs.
2. Both PDFs are standard normal distributions (i.e., $\mu = 0$ and $\sigma^2 = 1$).

$$\begin{aligned} \phi &= \int_{-\infty}^{\infty} V^2.\mathcal{N}_{V-X_1}(0,1).\mathcal{N}_{V-X_2}(0,1) dV \\ &= \int_{-\infty}^{\infty} V^2.\frac{1}{\sqrt{2\pi}} \exp^{-\frac{(V-X_1)^2}{2}} .\frac{1}{\sqrt{2\pi}} \exp^{-\frac{(V-X_2)^2}{2}} dV \\ &= \int_{-\infty}^{\infty} V^2.\frac{1}{2\pi} \exp^{-\frac{1}{2}[(V-X_1)^2+(V-X_2)^2]} dV \\ &= \int_{-\infty}^{\infty} V^2.\frac{1}{2\pi} \exp^{-\frac{1}{2}\eta} dV \end{aligned}$$

Now

$$\begin{aligned} \eta &= (V-X_1)^2 + (V-X_2)^2 \\ &= 2.V^2 - 2.V.(X_1+X_2) + (X_1^2+X_2^2) \\ &= 2[(V - \frac{X_1+X_2}{2})^2 + (\frac{X_1^2+X_2^2}{2}) - (\frac{X_1+X_2}{2})^2] \\ &= 2[(V - \frac{X_1+X_2}{2})^2 + g] \end{aligned}$$

where

$$\begin{aligned} g &= (\frac{X_1^2+X_2^2}{2}) - (\frac{X_1+X_2}{2})^2 \\ &= \frac{1}{4}(X_1-X_2)^2 \end{aligned}$$

Thus

$$\begin{aligned}\phi &= \int_{-\infty}^{\infty} V^2.\frac{1}{2\pi}\exp^{-\frac{1}{2}.2[(V-\frac{X_1+X_2}{2})^2+g]}\,dV \\ &= \int_{-\infty}^{\infty} V^2.\frac{1}{2\pi}\exp^{-\frac{1}{2}.2.(V-\frac{X_1+X_2}{2})^2}.\exp^{-\frac{1}{2}.2.g}\,dV \\ &= \frac{1}{2\sqrt{\pi}}exp^{-g}\int_{-\infty}^{\infty} V^2.\frac{1}{\sqrt{2.\pi.\frac{1}{2}}}\exp^{-\frac{(V-\frac{X_1+X_2}{2})^2}{2.\frac{1}{2}}}\,dV\end{aligned}$$

Let

$$\begin{aligned}I &= \int_{-\infty}^{\infty} V^2.\frac{1}{\sqrt{2.\pi.\frac{1}{2}}}\exp^{-\frac{(V-\frac{X_1+X_2}{2})^2}{2.\frac{1}{2}}}\,dV \\ &= \int_{-\infty}^{\infty} V^2.\frac{1}{\sqrt{2.\pi.\sigma^2}}\exp^{-\frac{(V-\mu)^2}{2\sigma^2}}\,dV.\end{aligned}$$

where $\mu = \frac{X_1+X_2}{2}$ and $\sigma^2 = 1/2$. We can do some variable transformation to simplify the above computation. Let $U = \frac{V-\mu}{\sigma}$. Thus the distribution of U is standard normal.

$$\begin{aligned}I &= \int_{-\infty}^{\infty} (\sigma U+\mu)^2.\frac{1}{\sqrt{2.\pi.\sigma^2}}\exp^{-\frac{U^2}{2}}\,dU \\ &= \int_{-\infty}^{\infty} (\sigma^2U^2+2\sigma\mu U+\mu^2).\frac{1}{\sqrt{2.\pi.\sigma^2}}\exp^{-\frac{U^2}{2}}\,dU \\ &= I_1+I_2+I_3\end{aligned}$$

where

$$I_1 = \int_{-\infty}^{\infty} \sigma^2U^2.\frac{1}{\sqrt{2.\pi.\sigma^2}}\exp^{-\frac{U^2}{2}}\,dU$$

$$I_2 = \int_{-\infty}^{\infty} 2\sigma\mu U.\frac{1}{\sqrt{2.\pi.\sigma^2}}\exp^{-\frac{U^2}{2}}\,dU$$

and

$$I_3 = \int_{-\infty}^{\infty} \mu^2.\frac{1}{\sqrt{2.\pi.\sigma^2}}\exp^{-\frac{U^2}{2}}\,dU.$$

Now I_1 and I_2 are nothing but the variance and mean of standard normal

distribution. Thus

$$\begin{aligned} I_1 &= \int_{-\infty}^{\infty} \sigma^2 U^2 . \frac{1}{\sqrt{2.\pi.\sigma^2}} \exp^{-\frac{U^2}{2}} dU \\ &= \sigma^2 \int_{-\infty}^{\infty} U^2 . \frac{1}{\sqrt{2.\pi.\sigma^2}} \exp^{-\frac{U^2}{2}} dU \\ &= \sigma^2 . 1 = \sigma^2 . \end{aligned}$$

$$\begin{aligned} I_2 &= \int_{-\infty}^{\infty} 2\sigma\mu U . \frac{1}{\sqrt{2.\pi.\sigma^2}} \exp^{-\frac{U^2}{2}} dU \\ &= 2\sigma\mu \int_{-\infty}^{\infty} U . \frac{1}{\sqrt{2.\pi.\sigma^2}} \exp^{-\frac{U^2}{2}} dU \\ &= 2\sigma\mu . 0 = 0. \end{aligned}$$

and

$$\begin{aligned} I_3 &= \int_{-\infty}^{\infty} \mu^2 . \frac{1}{\sqrt{2.\pi.\sigma^2}} \exp^{-\frac{U^2}{2}} dU. \\ &= \mu^2 . \end{aligned}$$

Thus ϕ becomes

$$\begin{aligned} \phi &= (\sigma^2 + \mu^2) \frac{1}{2\sqrt{\pi}} exp^{-g} \\ &= (\sigma^2 + \mu^2) \frac{1}{\sqrt{2\pi.2}} exp^{-\frac{1}{4}(X_1 - X_2)^2} \\ &= (\sigma^2 + \mu^2) \mathcal{N}_{X_1 - X_2}(0, 2) \end{aligned}$$

Thus integrated form of an ESS(variance) function is another ESS(variance) function.

Now for any arbitrary number of PDFs, the mean is $\mu = \frac{1}{n} \sum_{i=1}^{n} X_i$ and variance is $\sigma^2 = 1/n$. Similar to Property 6.7, it can be proved that the integrated form can be expressed as

$$\phi_n = \alpha(\sigma^2 + \mu^2) \prod_{i \neq j, i<j} \mathcal{N}_{X_i - X_j}$$

We can lift our assumptions as any arbitrary normal pdf can be expressed as standard normal distribution with a linear transformation on parameters (Property 1).

Chapter 8

Conclusion

In this dissertation, we developed algorithms for doing inference and learning in probabilistic logic programs with continuous random variables and linear equality constraints. In this final chapter, we summarize the key contributions of this thesis in Section 8.1, discuss possible future work directions in Section 8.2.

8.1 Contributions

The contributions of this thesis include: extension of PRISM with continuous random variable and linear equality constraints; the idea of symbolic derivation and computation; inference and learning algorithms; formal distribution semantics; and usable prototype or meta-interpreter for the extended language.

Extension of PRISM with continuous random variables and linear equality constraints. In this thesis, we extended PRISM with Gaussian/Gamma distributed random variables and linear equality constraints over reals. The use of linear equality constraints enables us to encode continuous child-continuous parent Bayesian network. Thus the extended framework can model a large class of hybrid Bayesian networks (e.g., HMM, FMM, KF) and Hybrid models.

In this dissertation, we reviewed some representative related work in the field of Statistical Relational Learning. Note that hybrid MLN and hybrid ProbLog also extended their base models with continuous random variables. Hybrid MLN supports description of continuous properties. But it does not provide any mechanism for creating a new random variable which is a linear combination of other random variables. Although hybrid ProbLog programs can use a continuous random variable, further processing can be based only on testing whether or not the variables value lies in a given interval. Thus certain classes of Hybrid Bayesian Networks (with continuous child with con-

tinuous parents), Kalman Filters, and Hybrid Models can not be encoded in hybrid MLN and ProbLog. The extension to PRISM described in this thesis encodes such statistical models. In addition, the complexity of our inference and learning techniques specializes to that of standard statistical models (e.g., HMM, FMM, KF).

Distribution semantics. PRISM's approach (e.g., enumerable set of facts, compatibility condition) of defining distribution semantics does not generalize naturally for continuous random variables as the Herbrand universe is infinite. Thus the formulation of the distribution semantics of the extended PRISM programs is an important problem. In this thesis, we formally define the distribution semantics of the extended PRISM programs.

Symbolic derivation. One of the key techniques used in this thesis is the idea of *symbolic derivation*, which compactly represents sets of explanations without enumerating over the outcomes of the random variables. Symbolic derivation is the key aspect of our inference and learning algorithms.

Inference and learning algorithms. In this thesis, we developed an inference algorithm in the context of PRISM; however the procedures core ideas can be easily applied to other PLP languages as well. An interesting aspect of our inference algorithm is that PRISMs query evaluation process becomes a special case in the absence of any continuous random variables in the program. The symbolic inference procedure enables us to reason over complex probabilistic models such as large sub-class of Hybrid Bayesian Networks (e.g., HMM, FMM, Kalman filter) and hybrid models that were hitherto not possible in PLP frameworks.

Our EM-based learning algorithm permits us to learn the distribution parameters of extended PRISM programs with discrete as well as Gaussian random variables. The learning algorithm naturally generalizes the ones used for PRISM and Hybrid Bayesian Networks.

Prototype implementation. We implemented the inference and learning algorithms presented in this thesis in the XSB logic programming system [59]. This proof-of-concept prototype illustrates how the inference and learning algorithms specialize to the specialized techniques that have been developed for several popular statistical models such as HMM, FMM, Hybrid Bayesian Networks and Kalman Filters.

Finally, this work opened up new directions of research in area of statistical relational learning, more specifically in probabilistic logic programming.

8.2 Future Work

Discrete child-continuous parent Hybrid Bayesian Network. One avenue for future research would be to support discrete child-continuous parent hybrid Bayesian network. As mentioned in the background section (Section 2.3.4), this type of hybrid Bayesian networks are represented using *softmax/logistic* functions. Inclusion of logistic function in success or ESS function computation does not satisfy the closure properties. Recent research [41] also show that without numeric integration, exact inference is not possible for this kind of hybrid Bayesian networks. Lerner et al. [35] proposed an exact inference algorithm which does numerical integration. Thus it remains to be investigated whether PLP frameworks can model this type of hybrid Bayesian networks.

Approximate Inference. Approximate inference and sampling based techniques can be used to perform inference when exact inference is not possible. For example, approximate inference techniques can be used to support discrete child-continuous parent hybrid Bayesian networks.

Many real world problems (e.g., decision making tasks) involve inequality constraints (e.g., $X < 1.5$). These problems involve making a decision based on continuous variable values (e.g., $temperature > 60$, $length < 5$, etc). Since inequality constraints can be represented using discrete child-continuous parent Bayesian network, support of this kind of Bayesian networks will also enable the support of inequality constraints.

In this thesis, we extended PRISM with Gaussian/Gamma distributed random variables. One avenue of future research would be to support other continuous distributions, and approximate inference techniques can be used to support those distributions.

Relationship with CLP. Constraint logic programming (CLP) supports linear equality/inequality constraints. The operations performed in CLP are very similar to our join and marginalize operations. For example, the join operation in CLP joins two constraint stores. Thus we can extend the CLP language to compute success functions. The idea is to have a distribution constraint store in addition to the traditional variable/value constraint store. The join operation in CLP will remain the same, whereas marginalize operation will perform projection/integration w.r.t. a continuous variable.

Performance Evaluation and Optimization. We implemented the inference and learning algorithms as a meta-interpreter in the XSB logic programming system [59]. A lower level integration with XSB and performance evaluation is a topic of future research. Note that various optimization techniques (e.g., tabling, dynamic programming) can be used to minimize the

computation time.

Application. Although the framework presented in this thesis can model a large sub-class of hybrid Bayesian networks, we have not thoroughly investigated all the application areas. As mentioned in the introduction, PRISM has been applied in the area of bioinformatics, sensor networks, and probabilistic planning; along the same lines, our extension permits natural encoding of those problems when data is continuous. However, we do not support various decision making tasks where the decision is made based on comparing the continuous value with a constant (e.g., turning a sensor on when the temperature goes below $60F$). Note that this an example of discrete child-continuous parent hybrid Bayesian network. Thus supporting discrete child-continuous parent hybrid Bayesian networks will give a much broader level of applicability of this framework in real-world applications.

Bibliography

[1] Ahrendt, P. The multivariate Gaussian probability distribution. Tech. rep., jan 2005.

[2] Apt, K. R., and van Emden, M. H. Contributions to the theory of logic programming. *Journal of ACM 29*, 3 (1982), 841–862.

[3] Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J. D. Magic sets and other strange ways to implement logic programs. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)* (1986), pp. 1–15.

[4] Bishop, C. *Pattern recognition and Machine Learning.* Springer (Information Science and Statistics), 2006.

[5] Bratko, I. *Prolog Programming for Artificial Intelligence.* Addison-Wesley Longman Ltd, 2000.

[6] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transaction of Computers 35*, 8 (1986), 677–691.

[7] Choi, J., Amir, E., and Hill, D. Lifted inference for relational continuous models. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)* (2010), pp. 126–134.

[8] Choi, J., Guzman-Rivera, A., and Amir, E. Lifted relational Kalman filtering. In *Proceedings of International Joint Conference in Artificial Intelligence (IJCAI)* (2011), pp. 2092–2099.

[9] Christiansen, H., Have, C. T., Lassen, O. T., and Petit, M. Inference with constrained hidden Markov models in PRISM. *Theory and Practice of Logic Programming (TPLP) 10*, 4-6 (2010), 449–464.

[10] Christiansen, H., and Lassen, O. T. Preprocessing for optimization of probabilistic-logic models for sequence analysis. In *Proceeding of International Conference in Logic Programming (ICLP)* (2009), pp. 70–83.

[11] Chu, D., Popa, L., Tavakoli, A., Hellerstein, J. M., Levis, P., Shenker, S., and Stoica, I. The design and implementation of a declarative sensor network system. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)* (2007), pp. 175–188.

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

[13] de Salvo Braz, R., Amir, E., and Roth, D. Lifted first-order probabilistic inference. In *Proceedings of International Joint Conference in Artificial Intelligence (IJCAI)* (2005), pp. 1319–1325.

[14] Dehaspe, L., Toivonen, H., and King, R. D. Finding frequent substructures in chemical compounds. In *Proceedings of Knowledge Discovery and Data mining (KDD)* (1998), pp. 30–36.

[15] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B 39*, 1 (1977), 1–38.

[16] Dzeroski, S. From inductive logic programming to relational data mining. In *JELIA* (2006), pp. 1–14.

[17] Dzeroski, S. Relational data mining. In *Data Mining and Knowledge Discovery Handbook*. 2010, pp. 887–911.

[18] Finn, P. W., Muggleton, S., Page, D., and Srinivasan, A. Pharmacophore discovery using the inductive logic programming system progol. *Machine Learning 30*, 2-3 (1998), 241–270.

[19] Forney, G. D. The Viterbi algorithm. In *Proceedings of the IEEE* (1973), pp. 268–278.

[20] Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. Learning probabilistic relational models. In *Proceedings of International Joint Conference in Artificial Intelligence (IJCAI)* (1999), pp. 1300–1309.

[21] Gelfand, A. E., and Smith, A. F. M. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association* (1990), 398–409.

[22] Getoor, L., and Taskar, B. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[23] Goswami, A., Ortiz, L. E., and Das, S. R. WiGEM: A learning-based approach for indoor localizatio. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)* (2011).

[24] Gutmann, B., Jaeger, M., and Raedt, L. D. Extending ProbLog with continuous distributions. In *Proceedings of Inductive Logic Programming(ILP)* (2010), pp. 76–91.

[25] Gutmann, B., Kimmig, A., Kersting, K., and De Raedt, L. Parameter learning in probabilistic databases: A least squares approach. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)* (2008), pp. 473–488.

[26] Gutmann, B., Thon, I., and De Raedt, L. Learning the parameters of probabilistic logic programs from interpretations. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)* (2011), pp. 581–596.

[27] Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., and De Raedt, L. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming (TPLP) 11*, 4-5 (2011), 663–680.

[28] Heckerman, D. A tutorial on learning with Bayesian networks. In *Innovations in Bayesian Networks*. 2008, pp. 33–82.

[29] Ishihata, M., Kameya, Y., Sato, T., and ichi Minato, S. An EM algorithm on bdds with order encoding for logic-based probabilistic models. *Journal of Machine Learning Research - Proceedings Track 13* (2010), 161–176.

[30] Jaffar, J., Maher, M. J., Marriott, K., and Stuckey, P. J. The semantics of constraint logic programs. *Journal of Logic Programming 37*, 1-3 (1998), 1–46.

[31] Kersting, K., and Raedt, L. D. Bayesian logic programs. In *Inductive Logic Programming(ILP) Work-in-progress reports* (2000).

[32] Kersting, K., and Raedt, L. D. Adaptive Bayesian logic programs. In *Proceedings of Inductive Logic Programming(ILP)* (2001), pp. 104–117.

[33] Kersting, K., Raedt, L. D., and Raiko, T. Logical hidden Markov models. *Journal of Artificial Intelligence Research (JAIR) 25* (2006), 425–456.

[34] Lari, K., and Young, S. J. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language 4* (1990), 3556.

[35] Lerner, U., Segal, E., and Koller, D. Exact inference in networks with discrete children of continuous parents. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)* (2001), pp. 319–328.

[36] Lloyd, J. W. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[37] Lowd, D., and Domingos, P. Efficient weight learning for Markov logic networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)* (2007), pp. 200–211.

[38] McLachlan, G., and Peel, D. *Finite mixture models.* Wiley Series in Probability and Statistics, 2000.

[39] Muggleton, S. Stochastic logic programs. In *Advances in inductive logic programming* (1996).

[40] Murphy, K. P. Inference and learning in hybrid Bayesian networks. *Technical Report UCB/CSD-98-990* (1998).

[41] Murphy, K. P. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)* (1999), pp. 457–466.

[42] Närman, P., Buschle, M., König, J., and Johnson, P. Hybrid probabilistic relational models for system quality analysis. In *Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC)* (2010), pp. 57–66.

[43] Pearl, J. *Probabilistic reasoning in intelligent systems - networks of plausible inference.* Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

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

[45] Poole, D. The independent choice logic and beyond. In *Probabilistic Inductive Logic Programming(ILP)* (2008), pp. 222–243.

[46] Pourret, O., Naim, P., and Marcot, B. *Bayesian Networks: A Practical Guide to Applications.* UK, Wiley, 2008.

[47] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recoginition. In *Proceedings of the IEEE* (1989), vol. 77, pp. 257–286.

[48] Raedt, L. D., and Kersting, K. Probabilistic logic learning. *Proceedings of the ACM SIGKDD Explorations, International Conference on Knowledge Discovery and Data Mining 5*, 1 (2003), 31–48.

[49] Raedt, L. D., and Kersting, K. Probabilistic inductive logic programming. In *Proceedings of the International Conference on Algorithmic Learning Theory (ALT)* (2004), pp. 19–36.

[50] Raedt, L. D., Kimmig, A., and Toivonen, H. ProbLog: A probabilistic prolog and its application in link discovery. In *Proceedings of International Joint Conference in Artificial Intelligence (IJCAI)* (2007), pp. 2462–2467.

[51] Richardson, M., and Domingos, P. Markov logic networks. *Machine Learning 62*, 1-2 (2006), 107–136.

[52] Riguzzi, F., and Swift, T. Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions. In *Technical Communications of the International Conference on Logic Programming* (2010), pp. 162–171.

[53] Russell, S., and Norvig, P. *Arficial Intelligence: A Modern Approach.* Prentice Hall Series in AI, 2003.

[54] Sato, T., and Kameya, Y. PRISM: a symbolic-statistical modeling language. In *Proceedings of International Joint Conference in Artificial Intelligence (IJCAI)* (1997), pp. 1330–1339.

[55] Sato, T., and Kameya, Y. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)* (1999), 391–454.

[56] Singh, A., Ramakrishnan, C. R., Ramakrishnan, I. V., Warren, D., and Wong, J. A methodology for in-network evaluation of integrated logical-statistical models. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)* (2008), pp. 197–210.

[57] Singla, P., and Domingos, P. Discriminative training of Markov logic networks. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)* (2005), pp. 868–873.

[58] Srinivasan, A., King, R. D., Muggleton, S., and Sternberg, M. J. E. Carcinogenesis predictions using ilp. In *Proceedings of Inductive Logic Programming (ILP)* (1997), pp. 273–287.

[59] Swift, T., Warren, D. S., et al. The XSB logic programming system, Version 3.3. Tech. rep., Department of Computer Science, SUNY, Stony Brook, 2011.

[60] Tamaki, H., and Sato, T. OLD resolution with tabulation. In *Proceeding of International Conference in Logic Programming (ICLP)* (1986), pp. 84–98.

[61] Vennekens, J., Denecker, M., and Bruynooghe, M. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming (TPLP)* (2009).

[62] Vennekens, J., Verbaeten, S., and Bruynooghe, M. Logic programs with annotated disjunctions. In *Proceedings of International Conference on Logic Programming (ICLP)* (2004), pp. 431–445.

[63] Wang, J., and Domingos, P. Hybrid Markov logic networks. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)* (2008), pp. 1106–1111.

[64] Warren, D. S. Memoing for logic programs. *Communications of ACM 35*, 3 (1992), 93–111.

[65] W.R., G., S., R., and D.J., S. *Markov chain Monte Carlo in practice*. London, UK: Chapman and Hall, 1996.

[66] Zhou, N.-F., and Sato, T. Efficient fixpoint computation in linear tabling. In *Proceedings of the International Symposium on Principles and Practice of Declarative Programming (PPDP)* (2003), pp. 275–283.