Logic-based Probabilistic Modeling

Taisuke Sato

Tokyo Institute of Technology, Ookayama Meguro Tokyo Japan,
http://sato-www.cs.titech.ac.jp/

Abstract. After briefly mentioning the historical background of PLL/SRL, we examine PRISM, a logic-based modeling language, as an instance of PLL/SRL research. We first look at the distribution semantics, PRISM's semantics, which defines a probability measure on a set of possible Herbrand models. We then mention characteristic features of PRISM as a tool for probabilistic modeling.

1 Introduction

Logic has long been considered as a discipline concerning certainty with major attention on deductive inference. However recent developments in machine learning and other related areas are expanding the role of logic from a vehicle for deductive inference to the one for probabilistic knowledge representation and statistical inference, and has spawned an interdisciplinary subfield of machine learning called PLL (probabilistic logic learning) and/or SRL (statistical relational learning) that combine logic and probability for probabilistic modeling[1].

As its name suggests, PLL[1] has LP (logic programming) and ILP (inductive logic programming) as its backbone. PLL however adds to LP/ILP probability and makes logical formulas random variables taking true and false probabilistically. With such probabilistic formulas we can express logical yet probabilistic phenomena such as gene inheritance cleanly at predicate level. Moreover probabilities statistically learned from real data makes it possible to evaluate how probable a given formula is in the real world.

Furthermore by amalgamating with probability, LP/ILP, or more generally logic, acquires an ability to cope with missing information. Suppose we are going for a picnic tomorrow (P) if the weather is clear (W). Also suppose we have to prepare a lunch box tonight (L) if we go for a picnic tomorrow (P). Then it follows by deduction that we have to prepare a lunch box tonight if the weather is clear tomorrow ($L \Leftarrow W$). However we do not know the truth value of W, tomorrow's weather. Should we prepare a lunch box tonight? In such a case a purely logical inference would completely get stuck as we do not know the truth value of W. The amalgamated system on the other hand will be able to help our decision on whether we should prepare a lunch box or not by looking at the probability associated with W.

[1] SRL seems more often used than PLL now.

SRL [2], independently of PLL, originated from uncertainty reasoning in BNs (Bayesian networks)[3]. BNs are a graphical representation of joint distributions in terms of directed acyclic graphs. BNs are powerful. First they can visualize complex dependencies among random variables graphically as parents-child nodes. Second they allow us to perform various types of probabilistic inference and parameter learning such as maximum likelihood inference, MAP (maximum a posteriori) inference and Bayes inference. These inferences are efficiently carried out by well-developed algorithms including the celebrated BP (belief propagation) algorithm. Also there are ones for structure learning as well that learn a directed graph from data. As a result BNs are used as one of the standard tools for handling uncertainty and applied to bioinformatics, natural language processing, speech recognition, planning, diagnosis, datamining, decision support and so on.

Nevertheless from a viewpoint of knowledge representation in AI, BNs are primitive. Basic assertions available in BNs are limited to $X = x$ type such that X is a random variable and x is a value. In other words, BNs are a propositional system where $X = x$ is a propositional variable which is true with probability $P(X = x)$. BNs' lack of logical variables, relations and quantifiers, the hallmark of predicate calculus, implies we have no way to express (probabilistic) rules such as "a friend of a friend is a friend with probability 0.3." One might say BNs is a low level language just like machine code.

The low-levelness of BNs prompted the emergence of the KBMC (knowledge-based model construction) approach in the nineties. Initially KBMC built BNs from a knowledge base using a high level language mostly as a matter of convenience. But later it evolved to SRL which upgrades BNs themselves by explicitly introducing variables and relations and/or by embedding them in a richer programming language be it logical or otherwise. Variables in such a language recover an ability to express general patterns. More importantly the introduction of relations opened a new dimension for feature-based probabilistic modeling[2] by using relations themselves as new features [2].

Currently SRL and PLL are moving toward unification but it remains to be seen whether they are eventually unified or not. In this talk, we look into some details of PRISM [4–6][3] as an instance of ongoing PLL/SRL research. PRISM is a probabilistic extension of Prolog with a declarative semantics called *distribution semantics* which is a probabilistic generalization of the standard least model semantics of logic programs [7]. Despite the fact that PRISM adopts a single data structure, a single probability computation algorithm and a single parameter learning algorithm, all independent of target models, it universally covers generative models from Bayesian networks to probabilistic grammars with efficiency supported by the PRISM architecture. In what follows we review the

[2] Features are a function from an input to a real number that characterizes the input. In statistical natural language proceeding for example, often an individual word is used as a feature that returns 1 if the input contains the word, else 0.

[3] `http://sato-www.cs.titech.ac.jp/prism/index.html`

distribution semantics in detail and then illustrate some aspects of PRISM from a machine learning point of view.

2 Probabilistic semantics

2.1 The basic principle

The distribution semantics, PRISM's declarative semantics, is a probabilistic generalization of the least model semantics in logic programming. Before proceeding however, we look back on a fundamental theorem which underlies it.

When logic and probability are combined, probabilities $P(\varphi)$ are assigned to formulas φ. However since probabilities have to respect Kolmogorov's axioms while formulas obeys their own logical laws, it is not self-evident how to coherently assign probabilities to formulas[4]. We here review Fenstad's representation theorem [8] in our context which states a formal relationship between formulas and their probabilities.

We introduce some notations. Let $\mathcal{L}$ be a countable first order language without equality and ω be a model on a certain fixed domain $\mathbf{U}$[5]. We assume variables are indexed by natural numbers $\mathbf{N}$. We denote by $\varphi[\omega]$ $\{a \in \mathbf{U}^{\mathbf{N}} \mid \omega \models_a \varphi(x_i, \ldots, x_j)\}$ where a is an assignment for the variables such that $a(i)$ is assigned to x_i etc. $\varphi[\omega]$ denotes the set of sequences of elements from $\mathbf{U}$ whose substitution for the variables in φ makes φ true in the model ω.

Theorem 1 (Fenstad 67). *Let φ, ϕ be formulas in $\mathcal{L}$. Suppose probabilities assigned to formulas satisfy the following.*

$$\begin{aligned}
&(i)\ P(\varphi \vee \phi) + P(\varphi \wedge \phi) = P(\varphi) + P(\phi)\\
&(ii)\ P(\neg\varphi) = 1 - P(\varphi)\\
&(iii)\ P(\varphi) = P(\phi),\ \textit{if} \vdash \varphi \leftrightarrow \phi\\
&(iv)\ P(\varphi) = 1,\ \textit{if} \vdash \varphi
\end{aligned}$$

Then there is a σ-additive probability measure λ on the set Ω of models on $\mathbf{U}$ for $\mathcal{L}$. There is also for each $\omega \in \Omega$, a probability μ_ω on the sets of $\varphi[\omega]$'s such that

$$P(\varphi) = \int_S \mu_\omega(\varphi[\omega])\, d\lambda(\omega).$$

When the domain $\mathbf{U}$ is finite and φ is closed, the above theorem reduces to

$$P(\varphi) = \sum_{\omega \models \varphi} \lambda(\{\omega\}).$$

So in this case, the probability is given as the sum of probabilities of the models that satisfy the formula.

[4] In addition since probabilities are supposed to be used in a probabilistic model, they need to be computable and learnable from data.

[5] $\mathbf{U}$ can be taken as a Herbrand universe for $\mathcal{L}$.

Fenstad's theorem strongly suggests to us that if we assign probabilities to formulas in a reasonable way, we should first define a probability measure over models. The *distribution semantics*[4] we explain next follows this idea.

2.2 The distribution semantics

Formally a PRISM program DB is a set of definite clauses. We write it as $DB = F \cup R$ where F is a set of ground atoms corresponding to primitive probabilistic events such as coin tossing and R is a set of rules (definite clauses). We assume no atom in F unifies with a head appearing in R.

We consider the set of Herbrand models (we often call them just *worlds* for simplicity) Ω_{DB} for DB and define a probability measure P_{DB} over Ω_{DB} as follows. First enumerate ground atoms in F like $A_1, A_2, \ldots$ and identify a Herbrand interpretation of F with a binary infinite vector $(1, 0, \ldots)$ that specifies A_1 is true (1), A_2 is false (0) and so on. Let $\Omega_F = \prod_i \{0,1\}_i$ be the set of such binary vectors. We give Ω_F a product topology where $\{0,1\}$ has a discrete topology. Also let P_F be a *base* distribution which is any probability measure on the σ-algebra generated by open sets of Ω_F.

Consider an arbitrary subset F' of F. $DB' = F' \cup R$ has the least Herbrand model $\mathbf{M}(DB')$ defined as follows. Let T be the immediate consequence operator [7]. It is applied to a set I of ground atoms and defined by $T(I) = \{A \mid A \Leftarrow B_1 \wedge \cdots \wedge B_h (h \geq 0)$ is a ground instance of a clause in DB' and $\{B_1, \ldots, B_h\} \subseteq I\}$. Put $I_\infty = \cup_{k=0}^{\infty} T^k(\emptyset)$ and verify I_∞ is the least fixpoint of T, i.e. $T(I_\infty) = I_\infty$. Define a Herbrand model $\mathbf{M}(DB')$ by a ground atom A is true in $\mathbf{M}(DB')$ iff $A \in I_\infty$[6]. $\mathbf{M}(DB')$ is called the least Herbrand model of DB'[7]. Using this $\mathbf{M}(DB') = \mathbf{M}(F' \cup R)$ as a model parameterized by F', a subset of F, we can extend P_F by Kolmogorov's extension theorem to a probability measure P_{DB} on the σ-algebra generated by open sets in Ω_{DB} with a product topology (see [5] for details). We consider P_{DB} as the denotation of P_{DB} (distribution semantics).

From the construction of P_{DB}, it is easy to see every closed formula φ is measurable when considered as a function from Ω_{DB} to $\{0,1\}$ such that

$$\varphi(\omega) = \begin{cases} 1 & \text{if } \omega \models \varphi \\ 0 & \text{else} \end{cases}$$

and hence we define the probability $P_{DB}(\varphi)$ of φ as $P_{DB}(\varphi = 1)$. Probabilities thus defined satisfy the conditions from (i) to (iv) of Fenstad's representation theorem. Also we can see the distribution semantics is a generalization of the least model semantics because if the base distribution P_F puts all probability mass on one Herbrand model making $F' \subseteq F$ true, P_{DB} also will put all probability mass on the least Herbrand model of $F' \cup R$.

[6] Proof theoretically $DB' \vdash A$ iff $A \in I_\infty$ for every ground atom A.

[7] In logic programming, the least Herbrand model is considered as the canonical denotation of definite clause programs.

3 PRISM: from semantics to implementation

The *distribution semantics* considers P_{DB} as the denotation of a program $DB = F \cup R$. P_{DB} always exists, uniquely, for any set F of ground atoms, any base measure P_F on F and any set R of definite clauses. Such "semantic robustness" is one of the unique features of PRISM compared to other systems dealing with infinite domains and infinitely many random variables [2, 1].

However defining a semantics is one thing and implementing it is another. When implementing the distribution semantics as PRISM as an extension of Prolog, we fix F and restrict the base measure P_F to a denumerable product of of Bernoulli distributions to make P_{DB} computable in probabilistic modeling.

More concretely we introduce ground atoms called `msw` atoms[8] representing a probabilistic choice that take the form $\texttt{msw}(i, v)$ where i is a choice name and v is a chosen value and both are ground terms. We fix the set $F_{\texttt{msw}}$ of ground `msw` atoms and give a (-n infinite) joint distribution $P_{\texttt{msw}}(\cdot)$ in such a way that if a probabilistic choice named i has k choices $v_1, \ldots, v_k$, correspondingly, one of $\texttt{msw}(i,v_1), \ldots, \texttt{msw}(i,v_k)$, say $\texttt{msw}(i,v_j)$ is exclusively true with probability $\theta_{v_j} = P_{\texttt{msw}}(\texttt{msw}(i,v_j))$ ($\sum_j \theta_{v_j} = 1$). $\texttt{msw}(i,v)$ is the only probabilistic built-in predicate in PRISM and used to simulate simple probabilistic events such as coin flipping and dice throwing. The role of definite clauses in R then is to organize such simple events into a complex event corresponding to our observation in the real world. The following is a PRISM program describing the inheritance of ABO blood type. As you see a PRISM program is just liken an ordinary Prolog program[9].

```
values_x(gene,[a,b,o],[0.5,0.2,0.3]).

bloodtype(P) :-
     genotype(X,Y),
     ( X=Y -> P=X ; X=o -> P=Y ; Y=o -> P=X ; P=ab ).
genotype(X,Y) :-
     msw(gene,X), msw(gene,Y).
```

Fig. 1. ABO-blood type program DB_1

The first clause `values_x(gene,[a,b,o],[0.5,0.2,0.3])` is a PRISM declaration specifying $F_{\texttt{msw}}$ and $P_{\texttt{msw}}(\cdot)$. It introduces a set of mutually exclusive atoms $\{\texttt{msw}(\texttt{gene},\texttt{a}), \texttt{msw}(\texttt{gene},\texttt{b}), \texttt{msw}(\texttt{gene},\texttt{o})\}$ corresponding to a probabilistic choice named `gene` having three possible outcomes `a`, `b` and `o`, representing three genes determing one's ABO blood type. They are true with 0.5, 0.2 and 0.3 respectively as indicated by the `values_x` declaration. The second clause is

[8] `msw` stands for "multi-ary random switch."
[9] Some familiarity with Prolog is assumed here.

a rule specifying the relationship between genotypes (pair of genes) and phenotypes (ABO blood type, `a`, `b`, `o`, `ab`). For example if a genotype is (`a`,`b`), the blood type is `ab`. The last clause simulates the inheritance of two genes, one from each parent. Sampling `msw(gene,X)` returns `X` = `a` with probability 0.5 etc. In PRISM, textually different occurrences of `msw` atoms in a program are treated as independent. So `msw(gene,X)` and `msw(gene,Y)` are independent. This program as a whole describes how `bloodtype(P)`, our observation, is generated by a sequential choices made by `msw(gene,·)` atoms.

Once loaded into computer memory by the PRISM system, DB_1 can answer various questions such as the probability of `bloodtype(a)`. PRISM computes it by way of search and the resulting propositional AND/OR formula called an explanation graph. To be precise it first performs an exhaustive SLD search for `?- bloodtype(a)` and collects all conjunctions E_1, E_2 and E_3 such that $E_i, DB_1 \vdash \texttt{bloodtype(a)}$ $(i = 1, 2, 3)$ where $E_1 = \texttt{msw(gene,a)} \wedge \texttt{msw(gene,a)}$, $E_2 = \texttt{msw(gene,a)} \wedge \texttt{msw(gene,o)}$ and $E_3 = \texttt{msw(gene,o)} \wedge \texttt{msw(gene,a)}$. We call each E_i an *explanation* for `bloodtype(a)`. Since the search is exhaustive, $\texttt{bloodtype(a)} \Leftrightarrow E_1 \vee E_2 \vee E_3$ holds with probability one in terms of PRISM's semantics. In addition since E_1, E_2 and E_3 are obtained by mutually exclusive proof paths, they are mutually exclusive as well. Also recall that `msw` atoms are independent. Putting these together $P_{DB_1}(\texttt{bloodtype(a)})$ is calculated as

$$\begin{aligned} P_{DB_1}(\texttt{bloodtype(a)} \mid \theta_a, \theta_b, \theta_o) &= P_{DB_1}(E_1 \vee E_2 \vee E_3) \\ &= P_{DB_1}(E_1) + P_{DB_1}(E_2) + P_{DB_1}(E_3) \\ &= \theta_a^2 + \theta_a\theta_o + \theta_o\theta_a \\ &= 0.45 \end{aligned}$$

where $\theta_a = P_{\texttt{msw}}(\texttt{msw(gene,a)}) = 0.5$, $\theta_b = P_{\texttt{msw}}(\texttt{msw(gene,b)}) = 0.2$ and $\theta_o = P_{\texttt{msw}}(\texttt{msw(gene,o)}) = 0.3$[10] as specified by `values_x(gene,[a,b,o],[0.5,0.2,0.3])`.

4 Statistical abduction

Although PRISM is an extension of Prolog, their inferences are of different type. Prolog is a logical language for (controlled) deduction whereas PRISM is a logical language for (controlled) abduction. In general abduction refers to "inference to the best explanation." Given a knowledge base K, a set of formulas, and an observation O, we seek for the best explanation E in abduction such that $K \wedge E \vdash O$ and $K \wedge E$ is consistent. The exhaustive search for explanations for the given goal in PRISM is exactly an abductive inference. One of the problems in abduction is that there can be many explanations just like a student has many excuses for not doing homework. In the blood type example, PRISM abduces three explanations E_1, E_2, and E_3 for the observation `bloodtype(a)` but in

[10] $P_{DB_1}(\cdot)$ is an extension of $P_{\texttt{msw}}(\cdot)$, so $P_{DB_1}(E_1) = P_{\texttt{msw}}(\texttt{msw(gene,a)} \wedge \texttt{msw(gene,a)}) = P_{\texttt{msw}}(\texttt{msw(gene,}a\texttt{)})^2 = {\theta_a}^2$.

the case of parsing where observations are sentences, the knowledge base is a grammar and an explanations is a parse tree, we often have tens of thousands of parse trees for one sentence. In the face of multiple explanations, we need to choose somehow one of them as the best one.

Statistical abduction[9] resolves the problem of multiple explanations that arises in abduction by introducing a probabilistic model $P(\cdot)$ connecting explanations E, a knowledge base K and an observation O. Using $P(\cdot)$ we choose the most probable E giving the highest $P(E \mid O, K)$ such that $K \wedge E \vdash O$. In this sense PRISM is not just a language for abduction but a language for statistical abduction, and indeed the first one with an ability to perform statistical inference to our knowledge. In PRISM the knowledge base is a program DB for which the distribution semantics guarantees $P_{DB}(\mathrm{iff}(DB)) = 1$[11] [5]. Accordingly

$$\begin{aligned} E^* &= \mathrm{argmax}_E P_{DB}(E \mid O, \mathrm{iff}(DB)) \\ &= \mathrm{argmax}_E P_{DB}(E \wedge O \wedge \mathrm{iff}(DB)) \\ &= \mathrm{argmax}_E P_{DB}(E \wedge \mathrm{iff}(DB)) \\ &= \mathrm{argmax}_E P_{DB}(E) \end{aligned}$$

holds. Thus seeking for the best explanation in statistical abduction is equivalent to Viterbi inference implemented in PRISM, giving $E^* = E_1$ as the best explanation for `bloodtype(a)`.

5 Probabilistic modeling

So far we have been looking at theoretical aspects of PRISM. Here we examine PRISM as a practical tool for probabilistic modeling. As a modeling tool, the most salient feature of PRISM is model specification by (recursive) definite clauses[12], which results in

- universality (for generative models and their parameter learning)
- high level specification (small amount of coding) and
- interpretability (what the system does is readable to humans).

The first point is due to the fact that PRISM can simulate, as an extension of Prolog, a non-deterministic Turing machine in which non-determinacy is resolved by a probabilistic choice, and in addition, PRISM has a generic routine (the graphical EM algorithm [5]) for parameter learning. The universality covers PCFGs (probabilistic context free grammars) [11] as well as BNs [3]. A PCFG is a CFG with probabilities assigned to grammar rules. It generates a sentence by repeatedly making a probabilistic choice of a grammar rule and expanding a

[11] $\mathrm{iff}(DB)$ is the if-and-only-if completion of DB. Definite clauses with a common head such as $A \Leftarrow B$ and $A \Leftarrow C$ in a program are lumped together to the if-and-only-if form $A \Leftrightarrow B \vee C$ in $\mathrm{iff}(DB)$.

[12] Actually general clauses (those that may contain negative goals in the clause body) are allowed under a certain condition[10].

nonterminal with it until no nonterminal remains. Since there is no upper limit on the number of applications of grammar rules, if there is a recursive rule, we use a countably many iid random variables. Hence finite probabilistic models such as BNs or otherwise cannot express PCFGs though they are the most basic class of probabilistic grammars [12].

The second point owes to the power of first-order expressions such as variables, terms, relations and recursion. For example HMMs (hidden Markov models) which are a class of stochastic automata very popular in machine learning can be expressed in three lines (together with three line declarations) as shown in Fig. 3.

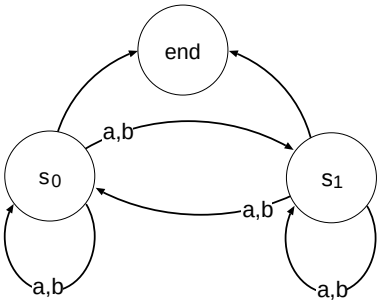

Fig. 2. An HMM

```
values_x(init,[s0,s1],[0.5,0.5]).
values_x(out(_),[a,b],[0.5,0.5]).
values_x(trans(_),[s0,s1,end],[0.7,0.2,0.1]).

hmm(L):- msw(init,S0),hmm(S0,L).
hmm(S,L):- msw(trans(S),NextS),
     (NextS=end -> L=[] ; msw(out(S),C), L=[C|Cs], hmm(NextS,Cs)).
```

Fig. 3. $DB_{\mathtt{hmm}}$ for the HMM in Fig.2

We hope the program $DB_{\mathtt{hmm}}$ in Fig. 2 is self-explanatory but add comments[13]. This program generates indefinitely long lists of `a` and `b`. There are two states `s0` and `s1` both of which can be an initial state. After choosing the initial state, it goes into infinite recursion, while outputting `a` or `b` on transition, until a choice of transition to `end`, the final state, is made probabilistically.

[13] `values_x(out(_),[a,b],[0.5,0.5])` is a template where the underscore _ can be replaced with any term.

```
?- prob(hmm([a,a,b])).
Probability of hmm([a,a,b]) is: 0.012345679012346

?- viterbif(hmm([a,a,b])).
hmm([a,a,b]) <= hmm(s0,[a,a,b]) & msw(init,s0)
hmm(s0,[a,a,b]) <= hmm(s1,[a,b]) & msw(trans(s0),s1) & msw(out(s0),a)
hmm(s1,[a,b]) <= hmm(s1,[b]) & msw(trans(s1),s1) & msw(out(s1),a)
hmm(s1,[b]) <= hmm(s1,[]) & msw(trans(s1),s1) & msw(out(s1),b)
hmm(s1,[]) <= msw(trans(s1),end)

?- learn([hmm([a,a,b]),hmm([b,a]),hmm([b,b]),hmm([a,b,a])]).
...
#em-iterations: 0.(18) (Converged: -12.908717468)
Switch init: s0 (p: 0.000005604) s1 (p: 0.999994396)
Switch out(s0): a (p: 0.499954371) b (p: 0.500045629)
Switch out(s1): a (p: 0.500019760) b (p: 0.499980240)
Switch trans(s0): s0 (p: 0.430352826) s1 (p: 0.000001065) end (p: 0.569646108)
Switch trans(s1): s0 (p: 0.573218835) s1 (p: 0.426780543) end (p: 0.000000623)
```

Fig. 4. Running $DB_{\texttt{hmm}}$

Fig. 4 is a sample session of $DB_{\texttt{hmm}}$. After loading, we issue `?-prob(hmm([a,a,b]))` to compute the probability of `hmm([a,a,b])`. Next we ask what is the most probable state transition sequence (Viterbi inference) for generating `hmm([a,a,b])`. The answer is given as a calling sequence (in Prolog) of subgoals. Finally, we learn parameters from a list of observations $\{\texttt{hmm}([\texttt{a},\texttt{a},\texttt{b}]), \texttt{hmm}([\texttt{b},\texttt{a}]), \texttt{hmm}([\texttt{b},\texttt{b}]), \texttt{hmm}([\texttt{a},\texttt{b},\texttt{a}])\}$ by invoking the graphical EM algorithm using `learn/1`. After 18 iterations it converged with log-likelihood -12.908717468, giving parameters as listed. "`Switch init`" signifies the parameters are for `msw(init,·)`.

We remark that it is straightforward to extend and modify, say merge with a PCFG, the above skeletal program to one's purpose. When the user modifies his probabilistic model, it often happens that he has to start from designing a new data structure all over again. Since PRISM adopts a single data structure (explanation graphs), there is no need for a new data structure. All you need to change when you change your model is the specification part alone, which is a labor saving aspect of PRISM.

The third point, interpretability, is of particular importance in practice. Eventually the outcome of our analysis by probabilistic modeling must be transferred to non-experts. However think of non-generative probabilistic models specified by "weights" w_i like $P(y \mid x) \propto \exp(\sum_i w_i f_i(x, y))$. It would be very hard to explain the meaning of those weights to non-experts, and especially so when there are a huge number of weights like in statistical natural language processing. On the contrary, logic-based probabilistic modeling uses logical formulas to

specify models, which seem more readable and more meaningful to non-experts than weights, though we admit logical formulas themselves might be an obstacle.

Last but not least, we comment on complexity. PRISM is a high-level modeling language and models can be described succinctly as a PRISM program. However one might ask if the ease of modeling sacrifices efficiency. The answer is possibly so but marginally. Due to the complexity analysis of representative models [5], HMMs, PCFGs and BNs can be computed in the same time complexity as specialized algorithms (the Baum-Welch algorithm for HMMs, the Inside-Outside algorithm for PCFGs, BP for BNs [13]). The reason is that explanation graphs, PRISM's data structure, realize structure-sharing and probabilities and expectations (needed for parameter learning) are computed by dynamic programming exploiting such structure-sharing. The real issue here is the trade off between general data structure for every model and specialized data structure for a specific model. PRISM lies on the general end of this scale. It constructs explanation graphs using pointers as their size depends on a model and data and is unknown beforehand in general. This causes a disadvantage in computational efficiency. We believe however the flexibility can compensate for such a disadvantage and implementation efforts can make it minimal.

6 Concluding remarks

We reviewed the historical background of PLL/SRL and examined PRISM as an instance of PLL/SRL research. It is a logic-based modeling language we have been developing in the past decade and provides a general tool for generative modeling in machine learning. As an extension of Prolog, it subsumes Prolog and furthermore has the ability to learn parameters from data based on an abductive framework called "statistical abduction." We omitted most of computational details but they can be reached by [4, 5, 14, 6]. Also omitted is variational Bayes which is the latest feature of PRISM for Bayesian inference [15].

References

1. De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming - Theory and Applications. Lecture Notes in Computer Science. Springer (2008) 1–27
2. Getoor, L., Taskar, B., eds.: Introduction to Statistical Relational Learning. MIT Press, Cambridge, MA (2007)
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
4. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP'95). (1995) 715–729
5. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15** (2001) 391–454

6. Sato, T., Kameya, Y.: New Advances in Logid-Based Probabilistic Modeling by PRISM. In De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S., eds.: Probabilistic Inductive Logic Programming. LNAI 4911, Springer (2008) 118–155
7. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag (1984)
8. Fenstad, J.E.: Representation of probabilities defined on first order languages. In Crossley, J.N., ed.: Sets, Models and Recursion Theory. North-Holland (1967) 156–172
9. Sato, T., Kameya, Y.: Statistical abduction with tabulation. In Kakas, A., Sadri, F., eds.: Computational Logic: Logic Programming and Beyond. LNAI 2408, Springer (2002) 567–587
10. Sato, T., Kameya, Y., Zhou, N.F.: Generative modeling with failure in PRISM. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). (2005) 847–852
11. Wetherell, C.S.: Probabilistic languages: a review and some open questions. Computing Surveys **12**(4) (1980) 361–379
12. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press (1999)
13. Sato, T.: Inside-Outside probability computation for belief propagation. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07). (2007) 2605–2610
14. Zhou, N.F., Sato, T., Shen, Y.D.: Linear tabling strategies and optimization. Theory and Practice of Logic Programming **8**(1) (2008) 81–109
15. Sato, T., Kameya, Y., Kurihara, K.: Variational bayes via propositionalized probability computation in prism. Annals of Mathematics and Artificial Intelligence, to appear.