A New Perspective of Statistical Modeling by PRISM

Taisuke SATO
Tokyo Institute of Technology / CREST, JST
2-12-1 Ôokayama Meguro-ku Tokyo Japan 152-8552
Neng-Fa Zhou
The City University of New York
2900 Bedford Avenue, Brooklyn, NY 11210-2889

Abstract

PRISM was born in 1997 as a symbolic statistical modeling language to facilitate modeling complex systems governed by rules and probabilities [Sato and Kameya, 1997]. It was the first programming language with EM learning ability and has been shown to be able to cover popular symbolic statistical models such as Bayesian networks, HMMs (hidden Markov models) and PCFGs (probabilistic context free grammars) [Sato and Kameya, 2001]. Last year, we entirely reimplemented PRISM based on a new tabling mechanism of B-Prolog [Zhou and Sato, 2002]. As a result, we can now deal with much larger data sets and more complex models. In this paper, we focus on this recent development and report two modeling examples in statistical natural language processing. One is a declarative PDCG (probabilistic definite clause grammar) program which simulates top-down parsing. The other is a left-corner parsing program which describes a bottom-up parsing that manipulates a stack. The fact that these rather different types of modeling and their EM learning are uniformly possible through PRISM programming shows the versatility of PRISM.[1]

1 Introduction

PRISM[2] was born in 1997 as a symbolic statistical modeling language to facilitate modeling complex systems governed by rules and probabilities [Sato and Kameya, 1997; 2001]. The basic idea is to incorporate a statistical learning mechanism into logic programs for embedded parameters. The result is a unique programming language for symbolic statistical modeling. Actually it was the first programming language with EM learning ability and has been shown to be able to cover popular symbolic statistical models. Theoretically it is an embodiment of Turing machines with learning ability, but the real consequence is that it enables us to build arbitrarily complex symbolic statistical models that may go beyond existing statistical models.

PRISM's power comes from three independent yet interrelated ingredients.

- firm mathematical semantics (*distribution semantics*) [Sato, 1995]
- all solution search using memoizing (OLDT [Tamaki and Sato, 1986] and *linear tabling* [Zhou and Sato, 2002])
- EM learning of parameters embedded in a program by *the graphical EM algorithm* [Kameya and Sato, 2000]

We will not go into the detail of each ingredient, but PRISM has proved to cover most popular statistical models such as HMMs (hidden Markov models) [Rabiner, 1989; Rabiner and Juang, 1993], PCFGs (probabilistic context free grammars) [Wetherell, 1980; Manning and Schütze, 1999] and Bayesian networks [Pearl, 1988; Castillo *et al.*, 1997] with the same time complexity [Sato and Kameya, 2001]. Moreover, we have experimentally confirmed that the learning speed of the graphical EM algorithm [Kameya and Sato, 2000], an EM algorithm for ML (maximum likelihood) estimation employed in PRISM for parameter learning outperforms that of the standard Inside-Outside algorithm for PCFGs by two or three orders of magnitude [Sato *et al.*, 2001].

From the view point of statistical modeling, one of the significant achievements of PRISM is the elimination of the need for deriving new EM algorithms for new applications. When a user constructs a statistical model with hidden variables, all he or she needs is to write a PRISM program using probabilistic built-ins such as `msw/2` predicate representing a parameterized random switch. The remaining work, namely parameter estimation (learning), is taken care of by the graphical EM algorithm quite efficiently thanks to dynamic programming. Furthermore, as long as certain modeling principles are observed, it is mathematically assured that the program correctly performs EM learning (this is *not* self-evident when the model gets complicated). One may say that PRISM is a generic tool for ubiquitous EM learning.

The development of PRISM was gradual because we attempted to fulfill two rather conflicting requirements; exploiting the generality of the semantics and achieving reasonable efficiency for real applications. After all we decided to compromise the generality of semantics and to assume some in-

[1] This paper is partly based on [Sato and Motomura, 2002].
[2] URL= `http://sato-www.cs.titech.ac.jp/prism/index.html`

dependence conditions on programs because while these conditions somewhat restrict the class of acceptable programs, they greatly simplify probability computations thereby making fast EM learning possible.

Our EM learning consists of two phases. In the first preprocessing phase, all solutions are searched for a given goal with respect to a program, yielding a hierarchical graph called an *explanation graph (support graph)*. In the second learning phase, we run the graphical EM algorithm on the explanation graph to train parameters in the program. The graphical EM algorithm is efficient in the sense that it runs in time linear in the size of the explanation graph in each iteration [Sato and Kameya, 2001]. In this learning scheme, compared to the efficiency of the graphical EM algorithm in the learning phase, all solution search in the preprocessing phase could be a bottleneck. A naive search by backtracking would take exponential search time. The key technology to efficiency is memoizing, i.e. to table calls and returns of predicates for later reuse which often reduces exponential time complexity to polynomial time complexity. However, the early versions of PRISM were built on top of SICStus Prolog and it was practically impossible to directly incorporate a full tabling mechanism.

Last year, we replaced the underlying Prolog with B-Prolog and reimplemented PRISM with a full linear tabling mechanism [Zhou and Sato, 2002]. As a result, we can now deal with much larger data sets and more complex models. In this paper, we focus on this recent development and report two modeling examples in statistical natural language processing. One is a declarative PDCG (probabilistic definite clause grammar) program which simulates top-down parsing. The other is a left-corner parsing program which procedurally describes a bottom-up parsing that manipulates a stack. The fact that these rather different types of modeling and their EM learning are uniformly possible through PRISM programming shows the versatility of PRISM.

2 Preliminaries

2.1 A quick review of PRISM

PRISM is a probabilistic extension of Prolog [Sterling and Shapiro, 1986]. A Prolog program is a set of logical formulas called *definite clause*s which take the form $H \,\texttt{:-}\, B_1, \ldots, B_k$ ($k \geq 0$). H is an atom called the *head*, and $B_1, \ldots, B_k$ is a conjunction of atoms called the *body*. The clause says if B_1 and $\cdots$ and B_k hold, then H holds (declarative reading). In the context of top-down computation however, it should be read that to achieve goal H, achieve subgoals B_1 and ... and B_k (procedural reading). This twofold reading i.e. bottom-up declarative reading, vs. top-down procedural reading, makes it possible to write declarative but executable programs that encodes both declarative and procedural knowledge in a unified way. When $k = 0$, the clause is called a unit clause. It represents a fact that holds unconditionally. Hence, a collection of ground unit clauses is considered as a relational database.

The surface syntax of PRISM is just Prolog augmented with built-in probabilistic predicates, but the semantics is substantially extended in order to comply with the need of subsuming statistical information in programs. Our semantics guarantees the existence of a unique probability measure, treating every ground atom as a binary random variable.

A PRISM program DB is a set of definite clauses. We write it as $DB = F \cup R$ where F is a set of facts (unit clauses) and R is a set of rules (non-unit clauses) to emphasize the difference of role between facts and rules. One of the unique features of PRISM is that F has a *basic joint probability distribution* P_F.[3] Put it differently, the truth of ground unit clauses $A_1, A_2, \ldots$ in F is probabilistic and their statistical behavior is specified by P_F. Here we consider ground unit clauses as random variables taking on 1 (true) or 0 (false).

What distinguishes our approach from existing approaches to probabilistic semantics is that our semantics admits infinite domains and allows us to use infinitely many random variables (probabilistic ground atoms). Consequently we need not make a distinction between Bayesian networks where a finite number of random variables appear and PCFGs where a countably infinite number of random variables are required. They are just two classes of PRISM programs. Another consequence is that we can implement a variety of EM algorithms as PRISM programs as long as they express, roughly speaking, Turing machines with probabilistic choices.

2.2 Grass_wet example

To put the idea of PRISM across, we show a propositional PRISM program $DB_p = R_p \cup F_p$ in Figure 1.[4] It dose not include any first-order features of PRISM such as logical variables and function symbols.

$$R_p = \begin{cases} \texttt{g\_wet} & \texttt{:-} \quad \texttt{s\_on.} \\ \texttt{g\_wet} & \texttt{:-} \quad \texttt{s\_off, w\_rain.} \\ \texttt{g\_dry} & \texttt{:-} \quad \texttt{s\_off, w\_clear.} \end{cases}$$

$$F_p = \begin{cases} \texttt{s\_on.} \quad \texttt{s\_off.} \\ \texttt{w\_rain.} \quad \texttt{w\_clear.} \end{cases}$$

Figure 1: Wet grass program DB_p

R_p expresses our causal knowledge on the events represented by six propostions: `g_wet` ("grass is wet"), `g_dry` ("grass is dry"), `s_on` ("sprinkler is on"), `s_off` ("sprinkler is off"), `w_rain` ("it is rainy") and `w_clear` ("it is clear"). The first clause says the grass is wet if the sprinkler is on. The second clause says the grass is wet also if the sprinkler is off but the weather is rain. The last clause says the grass is dry if the sprinkler is off and the weather is clear. We assume these rules hold without uncertainty.

In addition to the causal knowledge described above, we know that the states of weather and the sprinkler are probabilistic and are statistically independent. We put this knowledge into the program by setting a probability distribu-

[3] P_F actually is a probability measure over the Herbrand interpretations of F, but for presentation purpose we prefer to use the term "distribution."

[4] This is for the explanatory purpose and not a complete PRISM program. We furthermore need various declarations to run the program.

tion P_{F_p} over random variables s_on, s_off, w_rain and w_clear. When doing so we notice that either s_on or s_off is always true but not both, and this is true of w_rain and w_clear as well. We therefore introduce *parameters* $\theta_s = \text{prob}(\texttt{s\_on} = 1)$ and $\theta_w = \text{prob}(\texttt{w\_rain} = 1)$ and define P_{F_p} as

$$P_{F_p}(\texttt{s\_on} = x_1, \texttt{s\_off} = x_2, \texttt{w\_rain} = x_3, \texttt{w\_clear} = x_4) = \begin{cases} \theta_s^{x_1}(1-\theta_s)^{x_2}\theta_w^{x_3}(1-\theta_w)^{x_4} \\ \qquad \text{if } x_1 + x_2 = 1, x_3 + x_4 = 1 \\ 0 \quad \text{o.w.} \end{cases}$$

Here $x_i \in \{0\,(\text{false}), 1\,(\text{true})\}$ $(1 \le i \le 4)$.

Once P_{F_p} is given, the program DB_p defines a joint distribution P_{DB_p} for the six events as follows. Imagine a sample from P_{F_p} and let it be

$$\langle \texttt{s\_on} = 1, \texttt{s\_off} = 0, \texttt{w\_rain} = 0, \texttt{w\_clear} = 1 \rangle.$$

Since the set of true facts F'_p is $\{\texttt{s\_on}, \texttt{w\_clear}\}$, it follows that $F'_p \cup R_p \vdash$ g_wet and $F'_p \cup R_p \not\vdash$ g_dry. In other words, we have $\langle \texttt{g\_wet} = 1, \texttt{g\_dry} = 0 \rangle$. Now we generalize. Let $\langle x_1, x_2, x_3, x_4 \rangle$ be a truth value vector for $\langle \texttt{s\_on}, \texttt{s\_off}, \texttt{w\_rain}, \texttt{w\_clear} \rangle$ sampled from P_{F_p}. Likewise let $\langle y_1, y_2 \rangle$ be a truth value vector for $\langle \texttt{g\_wet}, \texttt{g\_dry} \rangle$. As we saw above, $\langle x_1, x_2, x_3, x_4 \rangle$ determines $\langle y_1, y_2 \rangle$ uniquely, i.e. $\langle y_1, y_2 \rangle$ is a function of $\langle x_1, x_2, x_3, x_4 \rangle$. We denote this function as $\varphi_{DB_p}(\langle x_1, x_2, x_3, x_4 \rangle) = \langle y_1, y_2 \rangle$. Define a joint distribution P_{DB_p} by

$$\begin{aligned} & P_{DB_p}(\texttt{g\_wet} = y_1, \texttt{g\_dry} = y_2, \texttt{s\_on} = x_1, \texttt{s\_off} = x_2, \\ & \qquad \texttt{w\_rain} = x_3, \texttt{w\_clear} = x_4) \\ & \stackrel{\text{def}}{=} \begin{cases} P_{F_p}(x_1, x_2, x_3, x_4) \\ \qquad \text{if } \varphi_{DB_p}(\langle x_1, x_2, x_3, x_4 \rangle) = \langle y_1, y_2 \rangle \\ 0 \quad \text{otherwise} \end{cases} \end{aligned}$$

With P_{DB_p} defined now, DB_p becomes a statistical model incorporating logical knowledge. We can calculate whatever probability we need using P_{DB_p}. The parameters θ_s and θ_w are estimated by ML (maximum likelihood) estimation from random observations of s_on, s_off, w_rain and w_clear.

In general, PRISM programs include function symbols, variables and recursion. As a result, the Herbrand domain is infinite and defining P_{DB} is more involved. P_{DB} should be understood as a probability measure over the set of Herbrand interpretations of DB, whose cardinality by the way is that of real numbers. Also since parameter learning is ML estimation from incomplete data, we rely on the EM algorithm [Dempster *et al.*, 1977; McLachlan and Krishnan, 1997] for parameter estimation (learning). Mathematical details are explained in [Sato and Kameya, 2001].

3 PDCG

One of the most notable phenomena in natural language processing over the past decade is the adaptation of statistical techniques applied to various corpora [Manning and Schütze, 1999]. In particular probabilistic parsing methods have been developed to tackle the otherwise intractable problem of identifying most plausible parses for a given sentence. Although there are many statistical language models usable for probabilistic parsing, PCFGs have been appreciated as the most basic one due to their simplicity. So we first explain briefly PCFGs [Wetherell, 1980; Manning and Schütze, 1999].

A PCFG (probabilistic context free grammar) is a probabilistic extension of CFG where a CFG rule has a probability. If there are N rules $A \to \alpha_1, \ldots, A \to \alpha_N$ for a non-terminal A, a probability θ_i^A is associated with each rule $A \to \alpha_1$ $(1 \le i \le N)$ such that $\sum_{i=1}^N \theta_i^A = 1$. These probabilities are called *parameters* in this paper. Then the probability $p(t)$ of a parse tree t is equal to the product of parameters of rules which are used in the (leftmost) derivation of t. Let T be the set of parse trees for a sentence s. We define the probability $p(s)$ of the sentence s as $p(s) = \sum_{t \in T} p(t)$. When we emphasize $p(s)$ is dependent on the parameters of rules, we write $p(s \mid \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ denotes the set of all parameters.

Below is a simple probabilistic top-down parser written in PRISM a la DCG which is intended to illustrate how easily we can build PCFG like language models (and perform EM learning). The program defines a distribution of provable ground atoms over the form pdcg([s],[w_1,...,w_n],[]) which corresponds to a sentence [w_1,...,w_n]. target(pdcg,3) declares what we observe is a predicate pdcg/3.

values/2 declares possible choices for each non-terminal on sentence derivation[5]. For example, values(s,[[ap,vp],[pp,v]]) tells us that the top category s, sentence, has two choices (rules) i.e. s $\to$ apvp and s $\to$ pp v such that s $\to$ ap vp is assigned probability θ_1 and s $\to$ pp v probability θ_2 $(\theta_1 + \theta_2 = 1)$ respectively. v,n,c,p are terminals and terminal(Wd) says Wd is a terminal whereas first(A,Wd) says Wd is in the first set of the category A. A probabilistic choice is simulated by a built-in predicate msw/2 according to the assigned parameters. For example, when msw(s,RHS) is called in execution mode, s $\to$ ap vp will be chosen with probability θ_1. Note that this program is left recursive and would go into an infinite recursion if run by Prolog, but the tabling mechanism of PRISM prevents infinite recursion and realizes CFG parsing with $O(n^3)$ time complexity where n is the sentence length [Sato and Kameya, 2001].

Since the precision of probabilistic parsing by a PCFG is largely determined by the quality of parameters associated with rules in the backbone CFG, their estimation is quite important. Usually it is done by ML estimation from a labeled corpus, i.e. a collection of parse trees). If the corpus is just a collection of sentences (or POS(part of speech) tag sequences), sentences become incomplete data, and it is customarily to appeal to the Inside-Outside algorithm [Baker, 1979; Pereira and Schabes, 1992; Schabes *et al.*, 1993]. In PRISM, the parameters in the above program are estimated by learn/1 built-in predicate. It automatically estimates parameters associated with msw atoms from raw data given

[5] values(s,[v_1,...,v_k]) declares that a probabilistic switch named s has k choices [v_1,...,v_k] where s and v_i $(1 \le i \le k)$ are terms. We use this switch s like msw(s,X) in a program when we make a probabilistic choice from [v_1,...,v_k].

```
target(pdcg,3).
values(s,[[ap,vp],[pp,v]]).
values(vp,[[ap,v],[pp,v]]).
values(np,[[ap,np],[n],
           [np,c,np],[v,n],[vp,n]]).
values(pp,[[n,p],[np,p]]).
values(ap,[[adv],[adv,adv],[adv,np]]).

pdcg([Wd|R],[Wd|L0],L2):-
    terminal(Wd),
    pdcg(R,L0,L2).

pdcg([A|R],[Wd|L0],L2):-
    first(A,Wd),
    ( values(A,[ RHS ])
    ; values(A,[_,_|_]), msw(A,RHS) ),
    pdcg(RHS,[Wd|L0],L1),
    pdcg(R,L1,L2).
pdcg([],L1,L1).
```

Figure 2: A PDCG parser

a list of goals of the form `pdcg([s],[w1,...,wn],[])` by first constructing explanation graphs using tabled search and second running the graphical EM algorithm on them.

The graphical EM algorithm is a generic EM algorithm for PRISM programs and calculates probabilities from explanation graphs, obeying the principle of dynamic programming. It is quite fast. When implemented in C and applied to explanation graphs generated from PCFGs, it runs by far faster than the Inside-Outside algorithm which has been the de facto standard EM algorithm for PCFGs and also runs faster than the Stolcke's EM learning algorithm [Stolcke, 1995], a much more refined EM algorithm based on the Earley parsing model. Experimentally, we observed that when all programs are written in C, the speed ratio[6] of the graphical EM algorithm to the Inside-Outside algorithm is about 1,000:1 and that to the Stolcke's EM learning algorithm is 10:1, depending on grammars [Sato *et al.*, 2001].[7] Unfortunately these speed ratios do not carry over to the graphical EM algorithm implemented in PRISM. This is because the data structure used in PRISM is Prolog terms and hence, we should not expect EM learning by PRISM can match a specialized EM algorithm implemented in C. Nonetheless, just for the record, we report that PRISM installed on a PC (Pentium IV 2.4GHz, 1GB memory, Windows XP) can learn parameters for the ATR grammars (861 CFG rules, 168 nonterminals, 446 terminals) from explanation graphs (95MB in memory) generated from 2,000 sentences of the ATR corpus [Uratani *et al.*, 1994] at a speed of 21 seconds/iteration and the whole learning takes 6,470 seconds (600 seconds for search) in total. We must add however that in this EM learning experiment, we did not use the program in Figure 2 but compiled it to take advantage of Prolog's indexing mechanism for clause invocation. By compilation, the specialized clause for the grammar rule $\mathtt{s} \rightarrow \mathtt{pp}\,\mathtt{v}$ looks like

```
pdcg(s,[A|B],C) :-
    first(s,A), msw(s,[pp,v]),
    pdcg(pp,[A|B],D),D=[v|C].
```

We feel that PRISM is becoming competitive with the Inside-Outside algorithm written in C now as far as learning speed is concerned. This is a bit surprising if one considers the fact that PRISM is a much higher level programming language than C. As our implementation still has room for improvement (see our companion paper [Zhou and Sato, 2003] for implementation details), we are expecting to be able to enhance the competitiveness considerably in the near future.

4 Declarative distributions vs. procedural distributions

The language model described by a PCFG is declarative in the sense that the probability of a sentence is directly related to CFG rules, and procedural aspects such as how a parse tree is constructed play no role in calculating the probability of the sentence. This declarative property makes it relatively easy to derive an EM algorithm for PCFGs (and their various extensions like lexicalized PCFGs) and apply it to existing CFG parsers [Stolcke, 1995; Charniak, 1997; Carroll and Rooth, 1998].

When it comes to procedurally defined stochastic CFG parsers, or procedurally defined distributions in general, little work has been done on their EM learning. For example, the GLR (generalized LR(k)) parser [Tomita, 1986] is undoubtedly one of the most sophisticated parsers for natural language processing which performs a sequence of complex stack manipulations while looking up a LR(k) table. Although its probabilistic extension, the PGLR (probabilistic GLR) parser has been proposed in the past [Briscoe and Carroll, 1994; Inui *et al.*, 1997], no EM algorithm is known so far.

This notable contrast can be presumably attributed to the difficultly of formalizing a distribution in terms of operations and their data types employed in the parsing procedure such as stacks, tables, list etc. In the following we present a PRISM program for probabilistic LC (left corner) parsing [Manning, 1997; Roark and Johnson, 1999; Van Uytsel *et al.*, 2001] as an example of the affinity of PRISM programming for procedurally defined distributions. Since PRISM is equipped with a formal semantics and the semantics of a PRISM program is mathematically well-defined, we can be sure of the correctness of EM learning performed by the program no matter how syntactically complicated it is.

5 Probabilistic LC parser

5.1 LC parsing

LC (left corner) parsing is sequential bottom-up parsing for CFG grammars which, like LR(k) parsing, manipulates a

[6]The speed ratio is measured in terms of time required for one iteration.

[7]Theoretically the speed gap is anticipated to widen as the grammars becomes less ambiguous.

stack to reduce subtrees to a larger tree. A program in Figure 3 is a skeletal Prolog LC parser[8]. The top goal is `lc(Ws)` and parsing starts with the subgoal `lc(Ws,[goal(s)])` in the first clause such that `Ws` is a list of words and `s` the starting symbol (sentence). The actual parsing is carried out by `process(Stack0,Stack,L0,L)` in the body of second clause which is tail-recursive.

The parser performs three operations. The shift operation reads a word from the input sentence and pushes it onto a stack which holds nonterminals whose subtrees are completed and subgoals waiting for their corresponding subtrees to complete. The attach operation attaches a completed subtree to the waiting subgoal indicated by `goal/1`. So if a subtree for `B` is completed and if it is waited by a subgoal `goal(B)` at the stack top, `B` is attached to `goal(B)` and the `goal(B)` is removed from the stack. The projection operation treats the completed `B` differently. When `B` is completed, it looks for a CFG rule that has `B` as the left corner category like $A \rightarrow B\,C\,D$ (see `rule(LHS,[B|Rest])` in the third `process/3` clause) and pushes `A`, `goal(D)` and `goal(C)` onto the stack in this order using `predict/3`. Usually top-down pruning is combined with projection and the operation is performed only when `A` is waited for by some subgoal in the stack (this part is not included in the program for simplicity).

```
lc(Ws) :- lc(Ws,[goal(s)]).
lc(L0,Stack0) :-
  process(Stack0,Stack,L0,L),
  lc(L,Stack).

% shift operation
process([goal(C)|Rest],
          [Wd,goal(C)|Rest], [Wd|L], L).

% attach operation
process([B,goal(B)|Stack], Stack, L, L).

% project operation
process([B|Goals], Stack, L, L) :-
  rule(LHS,[B|Rest],
  predict(Rest,[LHS|Goals],Stack).

predict([],L,L).
predict([A|Ls],L2,[goal(A)|NewLs]):-
   predict(Ls,L2,NewLs).
```

Figure 3: A non-probabilistic LC parser

5.2 Probabilistic LC parsing

Probabilistic LC parsing is just a probabilistic version of LC parsing but the point is that it parameterizes CFG rules differently from PCFGs. It assigns probabilities to three operations (shift operation, attach operation and projection operation) in LC parsing. Hence the resulting language distributions form a different class of distributions from those allowed by PCFGs and are expected to be more context sensitive.

Since PRISM programs can be arbitrary Prolog programs, writing a probabilistic LC parser as a PRISM program presents no difficulty to us. Furthermore once we finish writing, it means we have obtained an EM algorithm for LC parsing, provided "due care" is taken to ensure mathematical correctness. That is, the program is written so that it expresses a sequential probabilistic sentence generation process in which every choice is exclusive, independent and made by `msw/2` built-in and once a choice is made, it never leads to failure [Sato and Kameya, 2001].

```
% shift operation
process([goal(A)|Rest],Stack,[Wd|L],L):-
  ( terminal(A),
      A=Wd,Stack=Rest
  ; \+ terminal(A),
      ( values(first(A),[Wd])
      ; values(first(A),[_,_|_]),
          msw(first(A),Wd) ),
      Stack=[Wd,goal(A)|Rest]  ).

% attach or project operation
process([A|Rest],Stack,L,L):-
  \+ A=goal(_),
  Rest=[goal(C)|Stack0],
   ( A==C,
     % goal(A) waits for an A-tree
     ( values(lc(A,A),_),
       % attach and project are possible
         msw(attach(A),Op),
         ( Op==attach,Stack=Stack0
         ; Op==project,
             next_Stack(A,Rest,Stack) )
     ; \+ values(lc(A,A),_),
       % A is forcibly attached
         Stack = Stack0 )
   ; A\==C,
       next_Stack(A,Rest,Stack)  ).

% project operation
next_Stack(A,[goal(C)|Rest2],Stack) :-
    % subtree A is waited for by g(C)
  ( values(lc(C,A),[_,_|_]),
      msw(lc(C,A),rule(LHS,[A|RHS2]))
  ; values(lc(C,A),[rule(LHS,[A|RHS2])]) ),
  predict(RHS2,[LHS,goal(C)|Rest2],Stack).
```

Figure 4: A probabilistic LC parser

[8] This program is taken from [Manning, 1997] with a slight modification.

With this in mind, we replace clauses in Figure 3 for three operations with corresponding ones as in Figure 4 (PRISM declarations for `target/1` and `values/2` are omitted). Since we have to avoid failure in the generation process, program codes are more complicated than the non-probabilistic LC parser.

In a generation process, the shift operation for which the first clause is responsible has two cases depending on whether `A` in `goal(A)` on the stack top is terminal or not. If `A` is a nonterminal and if it has a non-singleton first set, we use `msw(first(A),Wd)` to probabilistically select a word `Wd` to shift[9].

The second clause handles the case where a subtree for nonterminal `A` is completed. There are two cases. The first case is where a subgoal `goal(A)` is waiting on the stack. The other case is where the subtree for `A` has no such waiting subgoal on the stack. The first case is further subdivided into two subcases. In the first subcase, projection is possible[10] as well as attachment. We check this possibility by `values(lc(A,A),_)` and when possible, make a probabilistic choice of the operation. The second subcase is where no such projection is possible and only attach operation is possible.

The project operation is executed in the third clause. When `C` in `goal(C)` on the stack has left-corner relationship with the completed `A` subtree, and if there is more than one rule of the form $\mathtt{A} \rightarrow \mathtt{B}\cdots$, we probabilistically choose one of such rules by `msw(lc(C,A),rule(LHS,[A|RHS2]))`.

The probabilistic LC parser in Figure 4 has no side effects and never fails when used as a sentence generator. It logically describes a sequential decision process where decisions are made by `msw/2` built-in. Consequently, we are sure that the EM learning performed by the program is mathematically correct[11].

We have successfully tested EM learning by the probabilistic LC parser with a small number of data randomly generated from the program itself, but a large scale learning experiment seems difficult because of huge memory requirement. We are developing yet another way to reduce memory requirement using a different formulation of probabilistic LC parsing.

We remark that although there is a formulation of probabilistic LC parsing [Manning, 1997; Roark and Johnson, 1999], the parameter learning there assumes a fully annotated corpus. The only literature we found on the EM learning of LC parsing is [Van Uytsel *et al.*, 2001] in which a specialized EM algorithm for (extended) LC parsing is sketched.

9

`+X` is Prolog's negation which succeeds if and only if the goal `X` fails. `A==B` succeeds if `A` and `B` are identical Prolog terms whereas `A=B` denotes the unification of `A` and `B`.

10 In this subcase `A` must have left-corner relationship with itself. In general `A` is said to have left-corner relationship with `A'` if there is a sequence of CFG rules such that $\mathtt{A} \rightarrow \mathtt{B}_1\beta_1, \mathtt{B}_1 \rightarrow \mathtt{B}_2\beta_2, \ldots, \mathtt{B}_n \rightarrow \mathtt{A}'\beta_n$.

11 To be precise, we need to add a condition "if there is no loss of probability mass to infinite generation process," which is difficult to verify except simple models like PCFGs.

6 Conclusion

Relational learning for uncertainty modeling at first order level is a natural extension of many, if not all, probabilistic approaches and has been developed over the past decade [Breese, 1992; Sato, 1995; Muggleton, 1996; Sato and Kameya, 1997; Koller and Pfeffer, 1997; Cussens, 1999; Friedman *et al.*, 1999; Jaeger, 2001; Sato and Kameya, 2001; Kersting and De Raedt, 2002]. Yet, there seems little work that exploits the full power of the generality of predicate logic combined with statistical learning. Most of work descended from Bayesian networks assumes domains are finite, and dynamic Bayesian networks remain a repetition of the same template. When logic programs are used as an underlying vehicle, range-restrictedness is often imposed which excludes common logic programs such as one for `member` predicate.

PRISM [Sato and Kameya, 1997; 2001] is a general programming language with EM learning ability for modeling symbolic-statistical phenomena. Syntactically it is Prolog augmented with parameterized probabilistic built-ins and accepts *any* programs regardless of whether they are range-restricted or not. Semantically it is the first programming language that can formally define distributions (probability measures) over infinite Herbrand domains. Practically, recent reimplementation of PRISM [Zhou and Sato, 2002] has brought about fast and robust EM learning based on tabled search. The adaptation of B-Prolog's linear tabling mechanism considerably shortens search time and also allows us to use recursive clauses which would otherwise cause infinite recursion, thereby providing us with far more freedom of modeling than previous implementations.

In this paper, we have reported two programming examples in the area of statistical natural language processing that take advantage of this new perspective offered by the latest PRISM. The first one in Figure 2 is a probabilistic DCG program for top-down parsing. It uses difference lists as data structures and accepts left recursive CFG rules. The second one in Figure 4 defines a bottom-up shift-reduce parser, probabilistic LC parser that manipulates a stack. Note that both programs are not range-restricted as logic programs, thus cannot be expressed by those approaches that inhibit non-range-restricted programs. They are not expressible by a fixed size network either because we need an indefinitely many number of random variables that have no upper bound.

Last but not least while we observe that the learning speed of PDCG in PRISM is catching up with the Inside-Outside algorithm implemented in C, it is obvious that we have a lot to do to put PRISM into real use.

References

[Baker, 1979] J. K. Baker. Trainable grammars for speech recognition. In *Proceedings of Spring Conference of the Acoustical Society of America*, pages 547–550, 1979.

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

[Briscoe and Carroll, 1994] T. Briscoe and J. Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59, 1994.

[Carroll and Rooth, 1998] G. Carroll and M. Rooth. Valence induction with a head-lexicalized PCFG. In *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP 3)*, 1998.

[Castillo *et al.*, 1997] E. Castillo, J. M. Gutierrez, and A. S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer-Verlag, 1997.

[Charniak, 1997] E. Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, 1997.

[Cussens, 1999] J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 126–133, 1999.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Royal Statistical Society*, B39(1):1–38, 1977.

[Friedman *et al.*, 1999] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 1300–1309, 1999.

[Inui *et al.*, 1997] K. Inui, V. Sornlertlamvanich, H. Tanaka, and T. Tokunaga. A new probabilistic lr language model for statistical parsing. Technical Report (Dept. of CS) TR97-0004, Tokyo Institute of Technology, 1997.

[Jaeger, 2001] J. Jaeger. Complex probabilistic modeling with recursive relational bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32(1-4):179–220, 2001.

[Kameya and Sato, 2000] Y. Kameya and T. Sato. Efficient EM learning for parameterized logic programs. In *Proceedings of the 1st Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 269–294. Springer, 2000.

[Kersting and De Raedt, 2002] K. Kersting and L. De Raedt. Basic principles of learning bayesian logic programs. Technical Report Technical Report No. 174, Institute for Computer Science, University of Freiburg, 2002.

[Koller and Pfeffer, 1997] D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1316–1321, 1997.

[Manning and Schütze, 1999] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[Manning, 1997] C.D. Manning. Probabilistic parsing using left corner language models. In *Proceedings of the Fifth International Conference on Parsing Technologies (IWPT-97)*, pages 147–158. MIT Press, 1997.

[McLachlan and Krishnan, 1997] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley Interscience, 1997.

[Muggleton, 1996] S. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

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

[Pereira and Schabes, 1992] F. C. N. Pereira and Y. Schabes. Inside-Outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL'92)*, pages 128–135, 1992.

[Rabiner and Juang, 1993] L. R. Rabiner and B. Juang. *Foundations of Speech Recognition*. Prentice-Hall, 1993.

[Rabiner, 1989] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[Roark and Johnson, 1999] B. Roark and M. Johnson. Efficient probabilistic top-down and left-corner parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 421–428, 1999.

[Sato and Kameya, 1997] T. Sato and Y. Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1330–1335, 1997.

[Sato and Kameya, 2001] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

[Sato and Motomura, 2002] T. Sato and Y. Motomura. Toward logical-probabilistic modeling of complex systems. In *Proceedings of the International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*, 2002.

[Sato *et al.*, 2001] T. Sato, S. Abe, Y. Kameya, and K. Shirai. A separate-and-learn approach to EM learning of PCFGs. In *Proceedings of the 6th Natural Language Processing Pacific Rim Symposium (NLRPS2001)*, pages 255–262, 2001.

[Sato, 1995] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729, 1995.

[Schabes *et al.*, 1993] Y. Schabes, M. Roth, and R. Osborne. Parsing the wall street journal with the inside-outside algorithm. In *Proceedings of the 6th European ACL Conference*, pages 341–347, 1993.

[Sterling and Shapiro, 1986] L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, 1986.

[Stolcke, 1995] A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201, 1995.

[Tamaki and Sato, 1986] H. Tamaki and T. Sato. OLD resolution with tabulation. In *Proceedings of the 3rd International Conference on Logic Programming (ICLP'86)*, volume 225 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 1986.

[Tomita, 1986] M. Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, 1986.

[Uratani *et al.*, 1994] N. Uratani, T. Takezawa, H. Matsuo, and C. Morita. ATR integrated speech and language database. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories, 1994. In Japanese.

[Van Uytsel *et al.*, 2001] D.H. Van Uytsel, D. Van Compernolle, and P. Wambacq. Maximum-likelihood training of the plcg-based language model. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop 2001 (ASRU'01)*, 2001.

[Wetherell, 1980] C. S. Wetherell. Probabilistic languages: a review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.

[Zhou and Sato, 2002] Neng-Fa Zhou and T. Sato. Toward a High-performance System for Symbolic and Statistical Modeling. Technical Report (Computer Science) TR-200212, City University of New York, 2002.

[Zhou and Sato, 2003] Neng-Fa Zhou and T. Sato. Toward a High-performance System for Symbolic and Statistical Modeling. In *Proceedings of IJCAI-03 workshop on Learning Statistical Models from Relational Data (SRL2003)*, 2003.