# Negation Elimination for Finite PCFGs

Taisuke Sato[1] and Yoshitaka Kameya[2] *

Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo, Japan,
CREST/JST
[1] sato@mi.cs.titech.ac.jp, [2] kameya@mi.cs.titech.ac.jp
WWW home page: http://sato-www.cs.titech.ac.jp

**Abstract.** We introduce negation to a symbolic-statistical modeling language PRISM and propose to eliminate negation by program transformation called negation technique which is applicable to probabilistic logic programs. We also introduce finite PCFGs (probabilistic context free grammars) as PCFGs with finite constraints as part of generative modeling of stochastic HPSGs (head-driven phrase structure grammars). They are a subclass of log-linear models and allow exact computation of normalizing constants. We apply the negation technique to a PDCG (probabilistic definite clause grammar) program written in PRISM that describes a finite PCFG with a height constraint. The resulting program computes a normalizing constant for the finite PCFG in time linear in the given height. We also report on an experiment of parameter learning for a real grammar (ATR grammar) with the height constraint. We have discovered that the height constraint does not necessarily lead to a significant decrease in parsing accuracy.

## 1 Introduction

### 1.1 Background

Symbolic-statistical modeling is a discipline where symbolic reasoning and statistical inference cooperate to identify the underlying structure of our observations of interest such as genome sequences, disease pedigrees and documents in a natural language that consist of structured symbols with various types of uncertainty. There are several formalisms already developed. HMMs (hidden Markov models) are a kind of stochastic automata used to identify for instance genes in genome sequences (and in many other areas) [1]. PCFGs (probabilistic context free grammars) are CFGs such that rule selection in a string derivation is probabilistic and they are applied to parsing and scene analysis [2, 3]. The most popular one is BNs (Bayesian networks) that can represent finite distributions of any type [4]. Recently they were applied to linkage analysis and beat competitors [5].

However, while HMMs, PCFGs and BNs can express uncertainty in terms of probabilities, they are all at propositional level and their logical power is limited. They do not have logical variables or quantifiers. There is no explicit treatment

of negation either. Naturally there have been efforts for upgrading these formalisms to the first-order level in various communities including the LP (logic programming) community, the ILP (inductive logic programming) community and the BN community among which is PRISM, a symbolic-statistical modeling language, we have been developing.

PRISM[1] is a probabilistic extension of Prolog augmented with a built-in EM learning routine[2] for statistical inference of parameters embedded in PRISM programs [7]. It is intended for modeling complex systems governed by rules and probability. Theoretically PRISM is a probabilistic Turing machine with a parameter learning mechanism which subsumes HMMs, PCFGs and discrete BNs in terms of expressive power, probability computation and parameter learning [8]. But what is genuinely innovative about it is that it opens a way to use programs as statistical models (programs are statistical models themselves in PRISM) and frees the user of having to derive a new EM algorithm for parameter learning for a new statistical model everytime he/she invents it.

Unfortunately the current PRISM lacks negation, which narrows the class of definable distributions and also causes inconveniences in modeling. To overcome this limitation, we propose to deal with negation by program transformation. The point is that we allow negated PRISM programs but eliminate negation by program transformation, thus recover negation-free PRISM programs.

There are two deterministic algorithms available for negation elimination of source programs. A general one is FOC (first order compiler), a deterministic program transformation algorithm originally developed for non-probabilistic logic programs containing universally quantified implications[3] [9]. It uses continuation[4] to compile universally quantified implications into executable form. FOC is general and can deal with large programs but tends to generate complicated and less efficient programs from the viewpoint of the PRISM's tabled search[5] [10, 11]. In this paper, we alternatively propose to use the negation technique [12]. It is a deterministic transformation algorithm to synthesize a logic

[1] URL = `http://sato-www.cs.titech.ac.jp/prism/`

[2] EM learning here means parameter learning by the EM algorithm which is an iterative algorithm for maximum likelihood estimation of parameters associated with a probabilistic model with hidden variables [6]. Hidden variables are those that are not directly observable like a disease in contrast to symptoms thereof.

[3] Universally quantified implications are formulas of the form $\forall x\,(\phi \Rightarrow \varphi)$ and negation is a special case ($\neg\phi$ is equal to $\phi \Rightarrow \texttt{false}$).

[4] Continuation is a data to represent the *rest* of computation. Usually it is a higher order object in functional programming but here we just use a first order term called continuation term, representing the next goal to be executed with the help of auxiliary clauses.

[5] Tabling is a search technique to record calling patterns of goals and their solutions for later use not to repeat the same search. It can avoid exponential explosion in the search space by sharing computation paths and brings about the same effect as dynamic programming in top-down search. Compilation by FOC introduces continuation terms that can be an obstacle to tabled search as they differentiate similar goals syntactically.

program that traces failed computation paths of the original program. While the negation technique is only applicable to the negation of definite clause programs, synthesized programs do not carry continuation terms and hence are more preferable in view of the tabled search in PRISM. The original negation technique was intended for non-probabilistic programs but we use here an extended version the use of which is justified by the *distribution semantics* [13], the formal semantics of PRISM programs.

### 1.2 Generative modeling and failure

Negation significantly expands the applicability of PRISM modeling, far beyond HMMs and PCFGs. We here detail our statistical motivation behind the introduction of negation. In the following we do not make a distinction between negation (logical notion) and failure (procedural notion) for brevity as we deal only with cases where they coincide.

Statistical models defined by PRISM are basically *generative*. By generative we mean PRISM programs describe a sequential stochastic process of generating observations such as one for the left-most derivation of sentences by a PCFG, where rules are probabilistically chosen to expand non-terminals. The implicit assumption is that the generation process never fails regardless of whether it is finite or infinite. Popular probabilistic models such as HMMs, PCFGs and BNs are considered generative and belong to the failure-free class.

We now allow a generative process to fail. So PRISM programs may fail. If failure occurs after a probabilistic choice is made, we lose probability mass placed on the choice and the sum of probabilities of all successful computation paths will be less than one. Statistically this implies that we have to renormalize probabilities by computing a normalizing constant $P(\texttt{success})$ where `success` denotes an event of occurrence of successful computation. Also we have to assume that what is observed is conditional probabilities $P(x \mid \texttt{success})$ where $x$ is an observation. In other words, by introducing failure, we shift to a class of log-linear models in general[6] [14]. They are quite flexible but the computational burden, especially computing a normalizing constant, is sometimes so high as to make their use prohibitive.

Despite such difficulty, we allow failure in our modeling because it enables us to use complex constraints for precise modeling. We impose constraints on each computation path that possibly generates an observation and filter out those paths that fail to satisfy the constraints. The probability mass is distributed over the remaining successful paths. The mathematical correctness of this modeling, i.e. probabilities sum to unity, is guaranteed by renormalizing success probabilities. This approach looks naive but in reality unavoidable when constraints are too complex for human beings to check their consistency.

In our case, we are aiming to model generative stochastic HPSGs (head-driven phrase structure grammars) [15] as one of the PRISM targets. Stochastic

[6] A distribution has the form $p(x) = Z^{-1} \exp(\sum_i \nu_i f_i(x))$ where $\nu_i$ is a coefficient, $f_i(x)$ a *feature* and $Z$ a normalizing constant. HMMs and PCFGs correspond to the special case where $Z = 1$.

HPSGs are a class of highly sophisticated unification grammars where lexical constraints and a few linguistic principles interact to specify a distribution of sentences. There was an attempt to formalize generative stochastic HPSGs by Brew [16] but faced with theoretical difficulties due to failure caused by conflicting constraints. As a result researchers in the area turned to non-generative log-linear models and their parameter learning [17–19]. Notwithstanding we, appreciating the simplicity and understandability of generative models, decided to pursue a generate-and-test approach using failure to generative stochastic HPSGs. As a concrete step toward this end, we introduce finite PCFGs which we explain next.

### 1.3 Finite PCFGs

*Finite PCFG*s are PCFGs with finite constraints that make them generate only a finite number of sentences. We for example impose an upper bound of the height of parse trees as a finite constraint. As long as the tree being derived is within the limit, we allow free derivation but once a constraint is violated we force the derivation to fail. Other types of finite constraint are possible such as the number of rule applications but we use the height constraint as a canonical one in this paper.

As a result of the height constraint, the number of sentences licensed by a PCFG becomes finite and we can, at least in theory, exactly compute a normalizing constant $P(\texttt{success})$. Once this is done, it is possible to statistically infer parameters associated with the PCFG from data by applying a new EM algorithm proposed for generative models with failure [11]. The new EM algorithm requires a failure program which simulates failed computations of the finite PCFG program. We synthesize it by applying the negation technique [12] to a PDCG (probabilistic definite clause grammar) program describing the PCFG.

Our contributions are as follows. We allow negation of probabilistic logic programs and propose negation elimination by the negation technique at compile time. We then apply it to a specific case of finite PCFGs which play an important role in our approach to generative stochastic HPSG modeling and show that computations concerning finite PCFGs with a height constraint can be done in polynomial time, not in exponential time. We also show by a learning experiment that the difference in parsing tasks between a finite PCFG with a height constraint and the corresponding non-finite PCFG is small.

In what follows, we first give an overview of PRISM [7]. We then review the negation technique by an example and show how we should modify it to accommodate probabilistic primitives in PRISM while keeping its semantics. We then apply the negation technique to finite PCFGs. Finally we report an experiment of parameter learning with a finite PCFG applied to a real corpus.

Our work lies at the borders of probabilistic semantics, negation, tabling and statistical natural language processing. Due to space limitation however, an in-depth treatment of each topic is difficult and our explanations will be example-based to save space for formal definitions. The formal description of

the semantics of PRISM and the analysis of its statistical learning are detailed in [8]. Most of the related work concerning first-order probabilistic modeling is omitted. The reader is referred to [7, 20, 14, 21–24, 8, 25–28]. He/she is assumed to be familiar with basics of statistical language models [2] as well as logic programming [29].

## 2 Preliminaries

Hereafter we use logic programs and follow Prolog conventions. A logic program $DB$ is a set of definite clauses $A$ `:-` $B_1, \ldots, B_n$ $(n \geq 0)$ where $A$ is a head and $B_i$ $(1 \leq i \leq n)$, an atom, is a subgoal. Unit clauses are those with $n = 0$ and goal clauses are those without a head. Variables in a clause are assumed to be universally quantified at the clause head. They are expressed by a string beginning with an upper case letter such as `X1` or just by underscore '`_`' in case of anonymous variables. Expressions (clauses, atoms, terms) without variables are said to be ground.

PRISM is a symbolic-statistical modeling language which is a probabilistic extension of Prolog such that unit clauses have a parameterized distribution. So unit clauses are probabilistically true. PRISM has been used to describe (and perform parameter learning of) a variety of probabilistic models including naive Bayes classifiers, Bayesian networks, HMMs, PCFGs, probabilistic left corner parsing models, probabilistic graph grammars, linkage analysis programs etc. Now we show in this paper that PRISM can also describe finite PCFGs and learn their parameters. Before proceeding we have a quick review of PRISM for self-containedness.

A PRISM program $DB'$ is the union of a set of definite clauses and a set of ground atoms $F = \{\mathtt{msw}_1, \mathtt{msw}_2, \ldots\}$. Each $\mathtt{msw} = \mathtt{msw}(id,v)$ represents a probabilistic choice $v$ by a trial of random switch $id$. A value declaration $\mathtt{value}(id,[v_1, \ldots, v_k])$ attached to $DB'$ specifies that $v$ is one of $v_1, \ldots, v_k$. (The Herbrand interpretations of) $F$ has a parameterized probability measure $P_{\mathtt{msw}}$ (*basic distribution*)[7]. PRISM has a formal semantics called *distribution semantics* in light of which $DB'$ denotes a probability measure extending $P_{\mathtt{msw}}$ over the set of possible Herbrand interpretations of the program. The execution of a PRISM program is just an SLD derivation except that a PRISM primitive $\mathtt{msw}(id,\mathtt{V})$ returns a probabilistically chosen value $v$ for `V`. Because PRISM semantics is a generalization of the standard logic programming semantics, in an extreme case of assigning probabilities 0 or 1 to `msw` atoms, PRISM is reduced to Prolog.

We write a PRISM program to define a distribution such as the distribution of sentences. Statistical inference of parameters associated with the basic distribution $P_{\mathtt{msw}}$ is carried out by special EM algorithms developed for PRISM programs. The gEM (graphical EM) algorithm incorporates the idea of dynamic

[7] We interchangeably use a probability measure and a probability distribution for the sake of familiarity. $P_{\mathtt{msw}}$ is a direct product of infinitely many Bernoulli trials of finitely many types specified by the user.

programming and is applicable to non-failing PRISM programs [30]. Also there is an enhanced version [11] which amalgamates the gEM algorithm and the FAM algorithm [24] to efficiently deal with PRISM programs that may fail.

Here is an example of PRISM program reproduced from [11]. This program simulates a sequence of Bernoulli trials and gives a distribution over ground atoms of the form `ber(`$n$`,`$l$`)` such that $l$ is a list of outcomes of $n$ coin tosses.

```
target(ber,2).
values(coin,[heads,tails]).
:- set_sw(coin,0.6+0.4).

ber(N,[R|Y]):-
    N>0,
    msw(coin,R),  % probabilistic choice
    N1 is N-1,
    ber(N1,Y).    % recursion
ber(0,[]).
```

**Fig. 1.** Bernoulli program

We use `target(ber,2)` to declare that we are interested in the distribution of `ber/2` atoms[8]. To define a Bernoulli trial we declare `values(coin,[heads,tails])` which introduces a discrete binary random variable named `coin` whose range is {`heads, tails`} in the disguise of exclusively true atoms `msw(coin,`$v$`)` where $v$ is either `heads` or `tails`. In PRISM a program is executed like Prolog, i.e. in a top-down left-to-right manner in the sampling mode (there are two other execution modes) and a call to `msw(coin,R)` returns a sampled value in `R`. `msw` atoms are primitives to make a probabilistic choice and their probabilities are called *parameters*.

`:- set_sw(coin,0.6+0.4)` is a directive on loading this program. It sets parameters of `msw(coin,`$\cdot$`)`, i.e. the probability of `msw(coin,heads)` to 0.6 and that of `msw(coin,tails)` to 0.4, respectively. Next two clauses about `ber/2` should be self-explanatory. Clauses behave just like Prolog clauses except that `R` works as a random variable such that $P(\texttt{R} = \texttt{heads}) = 0.6$ and $P(\texttt{R} = \texttt{tails}) = 0.4$. The query `:- ber(3,L)` will return for instance `L = [heads,heads,tails]`.

[8] `p/n` means that a predicate `p` has `n` arguments. We call an atom A `p` atom if the predicate symbol of A is `p`.

## 3 Finite PDCG program and the success probability

We here closely examine a PRISM program defining a finite PCFG. The PRISM program in Figure 2 is a probabilistic DCG (definite clause grammar) program written as a meta interpreter.

```
pdcg(L):-
    start_symbol(A), max_height(N), pdcg2(A,L,N).

pdcg2(A,[A],N):-
    N>=0, terminal(A).
pdcg2(A,L,N):-
    N>=0, \+terminal(A), msw(A,RHS), N1 is N-1, pdcg3(RHS,L,N1).

pdcg3([],[],_).
pdcg3([X|R],L3,N):-
    pdcg2(X,L1,N), pdcg3(R,L2,N), app(L1,L2,L3).

app([],A,A).
app([A|B],C,[A|D]):- app(B,C,D).
```

**Fig. 2.** A PDCG program with a height constraint

This program succinctly specifies a finite PCFG with a height constraint and simulates the leftmost derivation of sentences. CFG rules are supplied in the form of PRISM's value declarations. We for example declare `value(np,[[n],[s,np]])` to say that `np` has two rules `np -> n` and `np -> s np`[9]. `max_height(N)` says that the height of a parse tree must be at most `N`. `msw(A,RHS)` represents a probabilistic choice in the derivation. When `msw(A,RHS)` is executed with `A` = `np`, one of `[n]` or `[s,np]` is probabilistically chosen as `RHS`. `start_symbol(A)` returns in `A` a start symbol such as '`s`' corresponding to the category of sentence.

The counter `N` holds the allowed height of the parse trees and is decremented by one whenever a production rule is applied. When `N` becomes less than `0`, the derivation fails. So the program never generates a sentence whose height is more than `N` asserted by `max_height(N)`.

EM Learning from observed sentences of parameters associated with this finite PCFG is performed by a new EM algorithm for generative models with

[9] We also accept left recursive rules such as `s -> s s`. An infinite loop caused by them in top-down parsing is detected and properly handled by the PRISM's tabling mechanism.

failure proposed in [11]. Unlike the Inside-Outside algorithm[10] however, it needs a program that traces failed computation paths of the PDCG program, which is a challenging task. As an intermediate step, we derive a program specialized to computing the success probability. We transform the PDCG program to the success program shown in Figure 3 by dropping the arguments holding a partial sentence as a list.

```
success:-       % success:- pdcg(_).
    start_symbol(A),
    max_height(N),
    success2(A,N).

success2(A,N):-
    N>=0,
    ( terminal(A)
    ; \+terminal(A),
        msw(A,RHS),
        N1 is N-1,
        success3(RHS,N1) ).

success3([],_).
success3([A|R],N):-
    success2(A,N),
    success3(R,N).
```

**Fig. 3.** `success` program

This program is obtained by applying unfold/fold transformation to `success:-pdcg(L)` [33]. In the transformation we used a special property (law) of the append predicate such that $\forall$ `L1`, `L2` $\exists$ `L3` `app`(`L1`, `L2`, `L3`) holds for lists `L1`, `L2` and `L3`. The correctness of unfold/fold transformation, i.e. the source program and the transformed program define the same probability measure, is proved from the fact that the distribution semantics is an extension of the least model semantics. We here present a sketch of the proof.

Let $R_{\texttt{success}}$ be the clauses in Figure 3. Theoretically the `success` program $DB_{\texttt{success}}$ is the union of $R_{\texttt{success}}$ and the set of probabilistic ground atoms $F = \{\texttt{msw}_1, \texttt{msw}_2, \ldots\}$ with a basic distribution $P_{\texttt{msw}}$. To prove that the transformation

[10] The Inside-Outside algorithm is a standard EM algorithm for PCFGs [31]. It takes $O(n^3)$ time for each iteration where $n$ is a sentence length. Compared to the gEM (graphical EM) algorithm employed by PRISM however, it is experimentally confirmed that it runs much slower, sometimes hundred times slower than the gEM algorithm depending on grammars [32].

preserves the distribution semantics, it is enough to prove that for any true atoms $F' = \{\mathtt{msw}'_1, \mathtt{msw}'_2, \ldots\} (\subseteq F)$ sampled from $P_{\mathtt{msw}}$, the transformation preserves the least model of $R \cup F'$. However this is apparent because our transformation is unfold/fold transformation (using a 'law' about the append predicate) that preserves the least model semantics.

Since the computation by the query `:- success` w.r.t. the success program faithfully traces all successful paths generated by `:- pdcg(_)` and vice versa, we have

$$\sum_{x:\text{sentence}} P(x) = P(\mathtt{pdcg(\_)}) = P(\mathtt{success}).$$

Note that the success program runs in time linear in the maximum height `N` thanks to the PRISM's tabling mechanism [10] as is shown in Figure 7 (left). The graph is plotted using the ATR grammar, a CFG grammar for the ATR corpus[11] [34]. In the probability computation, we employed a uniform distribution, i.e. every rule is chosen with the same probability for each nonterminal. The `success` program is further transformed to derive a special program necessary for maximum likelihood estimation.

## 4 Negation technique

In order to perform EM learning of parameters associated with the PDCG program in the previous section, we have to know not only the probability of derivation failure, but have to know how production rules are used in the failed derivation [11]. To obtain such information is not a trivial task. We have to record each occurrence of `msw` atoms in every computation path regardless of whether it leads to success or not, which, naively done, would take exponential time.

Fortunately we can suppress the exponential explosion by sharing partial computation paths even for failed computations. As far as successful computations are concerned, it has been proved to be possible by the tabled search mechanism of PRISM [8]. Hence we have only to synthesize an ordinary PRISM program whose successful computation corresponds to the failed computation of the original program and apply the tabled search to the synthesized program. We here employ the *negation technique* [12] to synthesize such a negated program. We give a short synthesis example in Figure 4 in place of the formal description.

We negate a familiar logic program `mem/2` program by the negation technique. The source program is placed on the top layer in Figure 4. First we take the iff form of the source program (middle layer). The iff form is a canonical representation of the source program and `exists([V,W],[X,Y]=[V,[V|W]])` is a Prolog representation of `∃V,W([X,Y]=[V,[V|W]])`. We then negate both sides of the iff form. The left hand side `mem(X,Y)` is negated to `not(mem(X,Y))`. On the right hand side, the first disjunct `exists([V,W],[X,Y]=[V,[V|W]])`

[11] The ATR corpus is a collection of 10,995 Japanese conversational sentences and their parses. The ATR grammar is a manually developed CFG grammar for the ATR corpus. It contains 861 CFG rules.

```
mem(V,[V|W]).
mem(V,[U|W]):- mem(V,W).
%-----------------------------------------
mem(X,Y) :-
 ( exists([V,W],[X,Y]=[V,[V|W]])
 ; exists([V,U,W],[X,Y]=[V,[U|W]], mem(V,W)) )
%-----------------------------------------
not_mem(X,Y) :-
  \+([X,Y]=[V,[V|W]]),
  ( \+([X,Y]=[V,[U|W]])
  ; [X,Y]=[V,[U|W]],not_mem(V,W) ).
```

**Fig. 4.** Negation example

is negated to `all([V,W],not([X,Y]=[V,[V|W]]))`, a Prolog term representing $\forall$`V,W`$\neg$`([X,Y]=[V,[V|W]])`.

Likewise the second disjunct `exists([V,U,W],([X,Y]=[V,[U|W]],mem(V,W)))` is negated to `all([V,U,W],([X,Y]=[V,[U|W]]` $\Rightarrow$ `not(mem(V,W))))`. This is further transformed to `all([V,U,W],not([X,Y]=[V,[U|W]]);exists([V,U,W],([X,Y]=[V,[U|W]],not(mem(V,W))))` by using the property of '=' predicate such that $\forall$`X(Y` $= t$`[X]` $\Rightarrow \phi) \Leftrightarrow \forall$`X(Y` $\neq t$`[X])` $\vee \exists$`X(Y` $= t$`[X]` $\wedge \neg\phi$) holds for any $\phi$ in the Herbrand universe[12]. Finally we replace `not(mem(`$\cdot,\cdot$`))` with a new predicate `not_mem(`$\cdot,\cdot$`)` and `not(`$s$ `=` $t$`)` with `\+(`$s$ `=` $t$`)` to be executable. The bottom program computes exactly the complement of `mem` relation defined by the top layer `mem` program.

Let *DB* be the source program and $DB^c$ the negated program. A logic program is said to be *terminating* if an SLD derivation using a fair selection rule for `:-`$A$ w.r.t. the program terminates successfully or finitely fails for every ground atom $A$. A relation $q(x)$ is said to be complementary to $r(x)$ if $q(x) \vee r(x)$ is true for every $x$ and there is no $x$ such that $q(x) \wedge r(x)$.

**Theorem 1.** *[12] Suppose $DB^c$ is terminating. Relations over the Herbrand universe defined by $DB^c$ through its least model are complementary to those defined by DB.*

**Proof:** The least model of *DB* defines relations over the Herbrand universe for an interpretation of each predicate $q(x)$. They satisfy the if-and-only-if definition $q(x) \Leftrightarrow W[x]$. Hence the complementary relations satisfy the negated if-and-only-if definition $\neg q(x) \Leftrightarrow \neg W[x]$. Since operations on $\neg W[x]$ used in the negation technique are *substitution of equals for equals* in the Herbrand universe, these complementary relations satisfy iff($DB^c$), i.e. the collection of the if-and-only-if

[12] The reason is that for the given `Y`, the equation `Y` $= t$`[X]` determines at most one `X` occurring in $t$.

definition for each predicate, thereby giving a fixed point of iff($DB^c$) which must coincide with the least model of $DB^c$ because iff($DB^c$) is terminating, and hence has only one fixed point of the immediate consequence operator. Q.E.D.

## 5 Negating 'success' program

We apply an extended negation technique to the `success` program in Figure 3 and obtain the PRISM program for failure shown in Figure 5 after simplifications. We extend the original negation technique in two points. First noticing that $\forall y(q(x,y) \Rightarrow \psi)$ is equivalent to $\forall y\, \neg q(x,y) \vee \exists y(q(x,y) \wedge \psi)$ provided there exists at most one $y$ satisfying $q(x,y)$ for given $x$, we use this equivalence to rewrite the program in the negation process. The use of this equivalence does not invalidate the proof of Theorem 1 as long as the definition of $q(x,y)$ remains intact. Second we apply the negation technique to programs containing `msw` atoms which are a basic probabilistic primitive in PRISM. $\neg(\exists\, \mathtt{RHS}(\mathtt{msw}(\mathtt{A},\mathtt{RHS}) \wedge \psi))$ is transformed to $\forall\, \mathtt{RHS}(\mathtt{msw}(\mathtt{A},\mathtt{RHS}) \Rightarrow \neg\psi)$, and further transformed to $\exists\, \mathtt{RHS}(\mathtt{msw}(\mathtt{A},\mathtt{RHS}) \wedge \neg\psi)$. This transformation is justified by the PRISM's distribution semantics according to which `msw(A,RHS)` should be treated as a normal user-defined predicated defined by a single ground atom. So we may assume in the transformation there exists at most one `RHS` for a given `A`. We also use the fact that during the computation of `:- failure`, when `msw(A,RHS)` is called with `A` ground, it never fails.

```
failure:-        % failure:- not(success).
   start_symbol(A),
   max_height(N),
   failure2(A,N).

failure2(A,N):-  % failure2(A,N):- not(success2(A,N)).
   N>=0,
   \+terminal(A),
   msw(A,RHS),
   N1 is N-1,
   failure3(RHS,N1).
failure2(_,N):- N<0.

failure3([A|R],N):-
   ( failure2(A,N)
   ; success2(A,N), failure3(R,N) ).
```

**Fig. 5.** `failure` program

The `failure` program in Figure 5 is terminating. We prove using Theorem 1 and the definition of the formal semantics of PRISM programs that the probability of `failure` is exactly $1 - P(\texttt{success})$[13].

**Proposition 1.** $P(\texttt{success}) + P(\texttt{failure}) = 1$.

**Proof:** Suppose $F' = \{\texttt{msw}'_1, \texttt{msw}'_2, \ldots\}$ is an arbitrary set of `msw` atoms. Let $DB_{\texttt{success}}$ (resp. $DB_{\texttt{failure}}$) be a program consisting of $F'$ and the clauses in Figure 3 (resp. the clauses in Figure 5) respectively. Since $DB_{\texttt{failure}}$ is terminating, it follows from Theorem 1 that relations defined by $DB_{\texttt{success}}$ and those by $DB_{\texttt{failure}}$ are complementary, in particular `success` and `failure` are complementary. As $F'$ is arbitrary, it follows from the definition of the distribution semantics [8] that $1 = P(\texttt{success} \vee \texttt{failure}) = P(\texttt{success}) + P(\texttt{failure})$. Q.E.D.

To confirm Proposition 1, we let each program compute the success probability and the failure probability respectively, using a real grammar, the ATR grammar. We use a uniform distribution for rule selection probabilities for this test. The maximum height is set to 20. As the snapshot in Figure 6 testifies, probabilities for `success` and `failure` exactly sum to one[14].

```
?- prob(success,Ps),prob(failure,Pf),X is Ps+Pf.
X = 1.0
Pf = 0.295491045124576
Ps = 0.704508954875424
```

**Fig. 6.** Probabilities sum to one

The `failure` program runs in time linear in the maximum height `N` though we do not prove it (see Figure 7). We thus have reached an efficient PRISM program for computing failure required by EM learning.

## 6 Learning example: the ATR grammar

To gauge the effect of the height constraint in finite PCFGs, we conducted a small learning experiment with real data, the ATR corpus and the ATR grammar [34]. In the experiment, as a training corpus and a test corpus, we first randomly picked up 2,500 and 1,000 sentences from the original corpus, respectively. For the maximum height $N$, the sentences which have only parse trees higher than $N$ are excluded from the training and test corpora[15]. For each test sentence,

[13] The generalization of Proposition 1 for negated programs which are terminating is easy.

[14] `prob/2` is a PRISM built-in to compute the probability of a given atom under the current parameter values.

[15] The sizes of the training and test corpus are as follows:

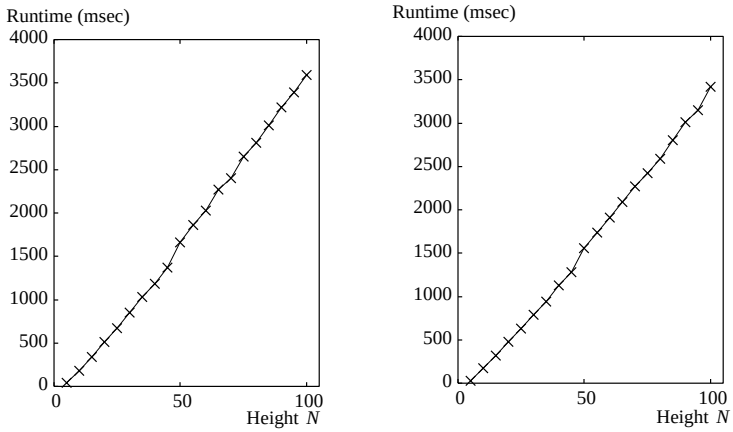


**Fig. 7.** Time for computing $P(\texttt{success})$ (left) and $P(\texttt{failure})$ (right) for the ATR grammar

we compared the height of the Viterbi parse, i.e. most likely parse based on the pure PCFG (whose parameters are learned by the Inside-Outside algorithm) and that based on the finite PCFG using a learning algorithm for finite PCFGs is described in [11].

The results are shown in Table 1. In the headers, $h_1$ (resp. $h_2$) indicates the height of the Viterbi parse based on the pure PCFG (resp. the finite PCFG). The column headed by '$h_1 > h_2$' shows the percentages of test sentences which hold $h_1 > h_2$, and so on. Table 1 shows that the finite PCFG model prefers shorter parse trees compared to the pure PCFG, hence we may say that we can add a height preference for parses by finite PCFGs, which is not easily realizable solely by pure PCFGs.

**Table 1.** Comparison on the height on Viterbi parses.

| $N$ | $h_1 > h_2$ | $h_1 < h_2$ | $h_1 = h_2$ |
|---|---|---|---|
| 15 | 11.2% | 2.1% | 86.7% |
| 18 | 14.2% | 1.8% | 84.0% |
| 20 | 6.5% | 3.6% | 89.9% |

| height $N$ | #training | #test |
|---|---|---|
| 15 | 2,252 | 913 |
| 18 | 2,465 | 990 |
| 20 | 2,492 | 996 |

Furthermore we evaluate the parsing accuracy with the finite PCFG based on the traditional criteria[16] [2]. The measured accuracy is given in Table 2. As the size of learning corpus is not large enough compared to the grammar size, we cannot make a definite comment on the performance differences between the pure PCFG and and the finite PCFG. However we may say that the parameters learned only from the parse trees with finite size does not necessarily lead to a significant decrease in parsing accuracy.

**Table 2.** Parsing accuracy with the pure PCFG and the finite PCFG.

| | LT | | BT | | 0-CB | |
|---|---|---|---|---|---|---|
| $N$ | Pure | Finite | Pure | Finite | Pure | Finite |
| 15 | 73.9% | 73.7% | 75.1% | 75.3% | 85.2% | 84.2% |
| 18 | 73.4% | 72.5% | 75.2% | 74.5% | 85.6% | 83.5% |
| 20 | 73.6% | 73.2% | 75.8% | 75.2% | 86.2% | 85.3% |

## 7 Conclusion

We have introduced negation to a symbolic-statistical modeling language PRISM and proposed to synthesize positive PRISM programs from negated ones by using the negation technique. The synthesized programs are used for PRISM to perform statistical parameter learning of generative models with failure.

The negation technique in this paper is more general than the original one presented in [12]. It allows us to use clauses that have internal variables[17] as long as they are uniquely determined by the (left-to-right) execution of the body[18]. We have shown in Section 5 that the synthesized PRISM program can exactly compute the probabilities of complementary relations, in particular the failure probability.

We also introduced finite PCFGs as PCFGs with finite constraints as part of generative modeling of stochastic HPSGs. They are a subclass of log-linear models and allow exact computation of normalizing constants. We have applied the negation technique to a PDCG program written in PRISM that describes a finite

[16] The criterion LT (labeled tree) is the ratio of test sentences in which the Viterbi parse completely matches the answer, i.e. the parse annotated by human. BT (bracketed tree) is the ratio of test sentences in which the Viterbi parse matches the answer ignoring nonterminal labels in non-leaf nodes. 0-CB (zero crossing brackets) is the ratio of test sentences in which the Viterbi parse does not conflict in bracketing with the answer.

[17] Internal variables are those occurring only in a clause body.

[18] For example the negation technique is applicable to a clause such as `p(X):-length(X,Y),q(X,Y)` where $Y$ is the length of a list $X$.

PCFG with a height constraint. The resulting program can compute a normalizing constant for the finite PCFG in time linear in the given height. Although we have shown only one example of finite PCFG, we have tested two other types of finite PCFG and found that their normalizing constants are computable in polynomial time.

Finally we conducted an EM learning experiment using the ATR corpus and the ATR grammar with a height constraint. We discovered that the height constraint does not heavily affect the performance of parsing tasks. Such comparison of finite and non-finite grammars is unprecedented in statistical natural language processing to our knowledge, though to what extent this result is generalized remains a future research topic.

## References


1. Rabiner, L.R., Juang, B.: Foundations of Speech Recognition. Prentice-Hall (1993)
2. Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press (1999)
3. Ivanov, Y., Bobick, A.: Recoginition of visual activities and interactions by stochastic parsing. IEEE Trans. Pattern Aanl. and Mach. Intell. **22** (2000) 852–872
4. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
5. Fishelson, M., Geiger, D.: Exact genetic linkage computations for general pedigrees. Bioinformatics **18 Suppl. 1** (2002) S189–S198
6. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions. Wiley Interscience (1997)
7. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). (1997) 1330–1335
8. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15** (2001) 391–454
9. Sato, T.: First order compiler: A deterministic logic program synthesis algorithm. Journal of Symbolic Computation **8** (1989) 605–627
10. Zhou, N.F., Sato, T.: Efficient Fixpoint Computation in Linear Tabling. In: Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP2003). (2003) 275–283
11. Sato, T., Kameya, Y.: A dynamic programming approach to parameter learning of generative models with failure. to be presented at ICML 2004 workshop SRL2004 (2004)
12. Sato, T., Tamaki, H.: Tansformational logic program synthesis. In: Proceedings of the International Conferenece on Fifth Generation Computer Systems FGCS84. (1984) 195–201
13. 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
14. Cussens, J.: Loglinear models for first-order probabilistic reasoning. In: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99). (1999) 126–133
15. Sag, I., Wasow, T.: Syntactic Theory: A Formal Introduction. Stanford: CSLI Publications (1999)

16. Brew, C.: Stochastic HPSG. In: Proceedings of the 7th Conference of European Chapter of the Association for Computational Linguistics (EACL'95). (1995) 83–89
17. Abney, S.: Stochastic attribute-value grammars. Computational Linguistics **23** (1997) 597–618
18. Riezler, S.: Probabilistic Constraint Logic Programming. PhD thesis, Universität Tübingen (1998)
19. Johnson, M., Geman, S., Canon, S., Chi, Z., Riezler, S.: Estimators for stochastic unification-based grammars. In: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99). (1999) 535–541
20. Koller, D., Pfeffer, A.: Learning probabilities for noisy first-order rules. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). (1997) 1316–1321
21. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99). (1999) 1300–1309
22. Muggleton, S.: Learning stochastic logic programs. In Getoor, L., Jensen, D., eds.: Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data. (2000)
23. Getoor, L., Friedman, N., Koller, D.: Learning probabilistic models of relational structure. In: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 01). (2001) 170–177
24. Cussens, J.: Parameter estimation in stochastic logic programs. Machine Learning **44** (2001) 245–271
25. Kersting, K., De Raedt, L.: Basic principles of learning bayesian logic programs. Technical Report Technical Report No. 174, Institute for Computer Science, University of Freiburg (2002)
26. De Raedt, L., Kersting, K.: Probabilistic logic learning. ACM-SIGKDD Explorations, special issue on Multi-Relational Data Mining **5** (2003) 31–48
27. Marthi, B., Milch, B., Russell, S.: First-order probabilistic models for information extraction. In: Proceedigs of IJCAI 2003 Workshop on Learning Statistical Models from Relational Data (SRL03). (2003)
28. Jaeger, J.: Complex probabilistic modeling with recursive relational bayesian networks. Annals of Mathematics and Artificial Intelligence **32** (2001) 179–220
29. Doets, K.: From Logic to Logic Programming. The MIT Press (1994)
30. Kameya, Y., Sato, T.: 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., Springer (2000) 269–294
31. Baker, J.K.: Trainable grammars for speech recognition. In: Proceedings of Spring Conference of the Acoustical Society of America. (1979) 547–550
32. Sato, T., Abe, S., Kameya, Y., Shirai, K.: A separate-and-learn approach to EM learning of PCFGs. In: Proceedings of the 6th Natural Language Processing Pacific Rim Symposium (NLRPS2001). (2001) 255–262
33. Tamaki, H., Sato, T.: Unfold/fold transformation of logic programs. In: Proceedings of the 2nd International Conference on Logic Programming (ICLP'84). Lecture Notes in Computer Science, Springer (1984) 127–138
34. Uratani, N., Takezawa, T., Matsuo, H., Morita, C.: ATR integrated speech and language database. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories (1994) In Japanese.