# Prefix and infix probability computation in PRISM

Ryosuke Kojima and Taisuke Sato

Tokyo Institute of Technology

**Abstract.** This paper presents the recent progress concerning prefix and infix probability for PCFGs in a logic-based modeling language PRISM. A prefix is an initial substring of a sentence and likewise an infix is a substring that occurs within a sentence. The prefix probability computation is already introduced to PRISM but applications are still scarce. We describe a new application to web data that identifies visitors' intentions, or goals visiting a website from observed sequences of their actions using prefix probability. We also discuss infix probability computation that generalizes prefix probability computation. Unlike previous approaches, we compute it through parsing followed by solving a set of non-linear equations.

**Keywords:** prefix probability, infix probability, PCFG, session log

## 1 Introduction

Probability computation based on explanation graphs in a logic-based modeling language PRISM has a rather long history [1, 2] where it is assumed that explanation graphs are acyclic and finite so that dynamic programming is applicable. Thanks to this dynamic programming feature, PRISM's probability computation and parameter learning for standard probabilistic models such as Bayesian networks (BNs), hidden Markov models (HMMs) and probabilistic context free grammars (PCFGs) are highly efficient and can be performed in the same time complexity as those algorithms specialized for each model class.

However there are cases where probability computation requires an infinite sum of probabilities, typically probabilistic model checking using Markov chains [3] and prefix probability computation in PCFGs [4, 5]. They are already made possible in probabilistic logic programming (PLP) [6, 7].

We present here two developments of our previous work [7] which introduces a basic mechanism of prefix probability computation by cyclic explanation graphs to PRISM. One is an application. We apply prefix probability computation to the problem of estimating visitors' goals who visit a website from their session log data. In our setting, visitors are a mixture of various groups with different goals. We analyze their action sequence as a sentence in some PCFG specialized for each group or goal, and represent the whole session log data as a mixture of

PCFGs, i.e., a combination of such specialized PCFGs[1]. The task is to estimate the visitor's goal as a most likely start symbol in the mixture of PCFGs. Our analysis will give us two kinds of information. One is the visitor's goal that is useful, for example, for web marketing. The other is a most likely parse tree for the visitor's action sequence which also helps us understand the visitor's behavior and will give us a clue to improve the website.

Nevertheless the above approach has two problems. A bigger one is that we cannot obtain any information until visitors complete their actions as sentences in a PCFG. So it is impossible to obtain visitor information online and guide them to a target page, say, by displaying affiliate links. The second one is the problem of *unachieved visitor*s. *Unachieved* visitors are those who quit the site for some reason before they achieve their goals or purposes. Obviously their action sequences should be considered as incomplete, not as complete sentences. We solve these two problems by generalizing sentences to prefixes and by considering action sequences as prefixes in a PCFG. A prefix is an initial substring of a sentence and since any incomplete sequence by an unachieved visitor can be considered as a prefix, this generalization makes it possible to identify a visitor's goal online and estimate the most likely parse tree for it.

The other development is a further generalization of prefixes to infixes, or prefix probability computation to infix probability computation. Unlike the former computation, infix probability computation is much harder and early attempts put some restrictions on it. However Nederhof and Satta recently proposed a completely general method to compute infix probability [8]. One thing we have to note is that their method only gives us probability. It never gives parse trees of prefixes or infixes, and hence Viterbi inference, obtaining a most likely tree for a prefix or an infix is impossible by their method. So we propose a new method, computing infix probability via parse trees expressed by cyclic explanation graphs in PRISM.

In the following, we first review prefix probability computation in PRISM and then look at the application of prefix probability computation to web session log data and then discuss infix probability computation implementable in PRISM. We assume the reader has a basic knowledge of PCFGs.

## 2 Prefix probability computation

A PCFG $\mathsf{G}_{\mathbf{\Theta}}$ is a CFG $\mathsf{G}$ augmented with a parameter set $\mathbf{\Theta} = \bigcup_{N^i \in \mathbf{N}} \{\theta_r\}_{N^i}$ where $\mathbf{N}$ is a set of nonterminals, and $N^1$ a start symbol and $\{\theta_r\}_{N^i}$ the set of parameters associated with rules $\{r \mid r = N^i \to \zeta\}$ for a nonterminal $N^i$ where $\zeta$ is a sequence of nonterminal and terminal symbols . We assume that the $\theta_r$'s satisfy $0 < \theta_r < 1$ and $\sum_{\zeta : N^i \to \zeta} \theta_{N^i \to \zeta} = 1$.

[1] A mixture of PCFGs is a combination of component PCFGs. It specifies a distribution $P(\mathbf{w} \mid N^1)$ of sentences $\mathbf{w}$ derived from the start symbol $N^1$ as $P(\mathbf{w} \mid N^1) = \sum_A P^A(\mathbf{w} \mid A) P(A \mid N^1)$ where $P^A(\mathbf{w} \mid A)$ is a distribution defined by a component PCFG with a start symbol $A$.

There are a couple of methods to compute prefix probabilities in PCFGs [4]. We briefly describe prefix probability computation in PRISM based on explanation graphs [7]. A *prefix* $\mathbf{v}$ is an initial substring of a sentence and the prefix probability $P_{\mathrm{pre}}^{N^1}(\mathbf{v})$ of $\mathbf{v}$ is defined as an infinite sum of probabilities of sentences extending $\mathbf{v}$:

$$P_{\mathrm{pre}}^{N^1}(\mathbf{v}) = \sum_{\mathbf{w}} P_{\mathsf{G}}(\mathbf{vw})$$

where $\mathbf{w}$ ranges over strings such that $\mathbf{vw}$ is a sentence in $\mathsf{G}$. Prefix probabilities are computed in PRISM by way of cyclic explanation graphs. Due to space limitations, we only sketch our prefix probability computation following [7]. We use a PCFG $\mathsf{G}_0 = \{\ \mathtt{s} \to \mathtt{s}\,\mathtt{s} : 0.4,\ \mathtt{s} \to \mathtt{a} : 0.3,\ \mathtt{s} \to \mathtt{b} : 0.3\ \}$ where "s" is a start symbol and "a" and "b" are terminals and consider to compute the prefix probability $P_{\mathrm{pre}}^{\mathtt{s}}(\mathtt{a})$ of prefix a. To compute $P_{\mathrm{pre}}^{\mathtt{s}}(\mathtt{a})$, we first parse "a" as a prefix by the PRISM program below.

```
values(s,[[s,s],[a],[b]]).
:- set_sw(s,[0.4,0.3,0.3]).

pre_pcfg(L):- pre_pcfg([s],L,[]).          % (1) L is a prefix
pre_pcfg([A|R],L0,L2):-                    % (2) L0 is ground when called
   ( get_values(A,_) -> msw(A,RHS),        % (3) if A is a nonterminal
       pre_pcfg(RHS,L0,L1)                 % (4) select rule A->RHS
   ;  L0=[A|L1] ),                         % (5) else consume A in L0
   ( L1=[] -> L2=[]                        % (6) (pseudo) success
   ; pre_pcfg(R,L1,L2) ).                  % (7) recursion
pre_pcfg([],L1,L1).                        % (8) termination
```

**Fig. 1.** Prefix parser $DB_0$

$DB_0$ in Fig. 1 is a prefix parser for $\mathsf{G}_0$. As can be seen from the comments, it runs exactly like a standard top-down CFG parser except *pseudo success* at line (6). *pseudo success* means an immediate return with success on the consumption of the input prefix L1 ignoring the remaining nonterminals in R at line (2)[2].

By running a command ?-probf(pre_pcfg([a])), we obtain an explanation graph in Fig. 2 (left) for pre_pcfg([a]). It is a set of boolean formulas defining intermediate goals by equivalence formulas. It represents all possible derivation paths of the top-goal pre_pcfg([a]) in terms of AND/OR formulas comprised

[2] This is justified because we assume the consistency of PCFGs [9] that implies the probability of remaining nonterminals in R yielding some terminal sequences is 1.

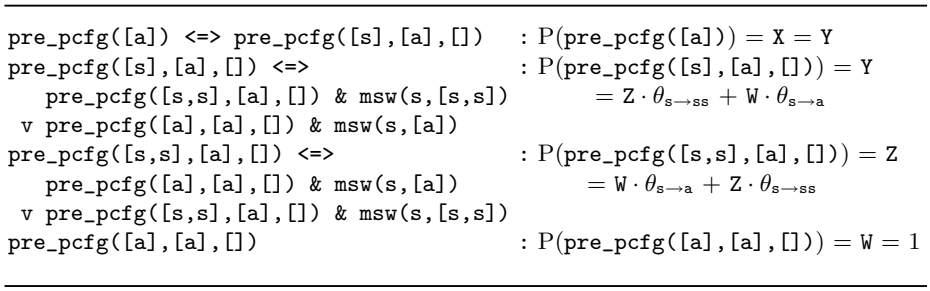


**Fig. 2.** Explanation graph for prefix "a" (left) and associated probability equations (right)

of msw atoms representing probabilistic choices. Note a cycle in the explanation graph: pre_pcfg([s,s],[a],[]) calls itself in the third equivalence formula.

Then we convert the equivalence formulas to a set of probability equations about X, Y, Z and W as in Fig. 2 (right). We use the fact that goals are independent ($P(A \wedge B) = P(A)P(B)$) and discjuntions are exclusive ($P(A \vee B) = P(A) + P(B)$). By solving them using parameter values $\theta_{\mathtt{s}\to\mathtt{ss}} = 0.4$ and $\theta_{\mathtt{s}\to\mathtt{a}} = 0.3$ set by :-set_sw(s,[0.4,0.3,0.3]) in the program $DB_0$, we eventually obtain $\mathtt{X} = \mathtt{Y} = \mathtt{Z} = 0.5$[3]. So we have $P^{\mathtt{s}}_{\mathrm{pre}}(\mathtt{a}) = \mathtt{X} = 0.5$. In general, a set of probability equations generated from a prefix in a PCFG using $DB_0$ is always linear and solvable by matrix operation [7].

## 3 Action sequences as prefixes in a PCFG

Here we tackle the problem of identifying visitors' purposes, or goals who visit a website from their session logs. We first abstract a visitor's session log into a sequence of actions comprised of five basic actions: up, down, sibling, reload and move. The first two, up and down, describe that a visitor moves respectively to a page in the parent directory and a subdirectory in the site's directory structure. An action sibling says that a visitor moves to a page in a subdirectory of the parent directory. An action reload means that a visitor requests the same page. An action move categorizes remaining miscellaneous actions. Moving between web pages is expressed by a sequence of basic actions. For example moving from /top/index.html to /top/child/a.html is a down action and moving from /top/index.html to /top/b.html is a sibling action.

We consider an action sequence generated by a visitor who has achieved the intended goal as a sentence in a PCFG. We parse it using rules like those in Table 1 and obtain a parse tree illustrated in Fig. 3. The CFG rules in Table 1 describe the visitors' goal-subgoal structure behind the action sequence. For example, the second and fourth rules say that when visitors' intention is Survey,

[3] W = 1 because pre_pcfg([a],[a],[]) is logically proved without involving msws.

**Table 1.** CFG rules

| |
|---|
| S → Survey |
| Survey → Search Destination |
| Search → Down Up Search \| Down |
| Destination → `sibling` Destination \| `sibling` |
| Down → Down `down` \| `down` |
| Up → Up `up` \| `up` |

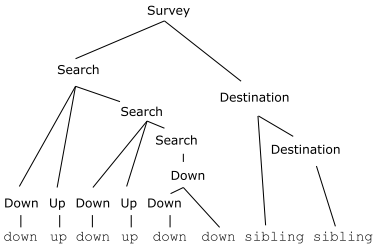


**Fig. 3.** Example of parse tree using rules in Table 1

they perform Search for a target page, reach a Destination page and looks around the Destination page.

Since diverse visitors visit a website with different goals, we capture action sequences $\mathbf{w}$ made by those diverse visitors in terms of a mixture of PCFGs $P(\mathbf{w} \mid N^1) = \sum_A P^A(\mathbf{w} \mid A)P(A \mid N^1)$ where $P^A(\mathbf{w} \mid A)$ is the probability of $\mathbf{w}$ being generated by a visitor whose goal is represented by a nonterminal $A$ and $P(A \mid N^1)$ is the probability of $A$ being derived from the start symbol $N^1$ respectively. We call such $A$ a *goal-nonterminal* and assume that there is a unique rule $N^1 \to A$ for each goal-nonterminal $A$ and also assume that it has a parameter $\theta_{N^1 \to A} = P(A \mid N^1)$.

Finally to be able to estimate visitor goals online and also from action sequences generated by unachieved visitors, we replace a sentence probability $P^A(\mathbf{w} \mid A)$ in a mixture of PCFGs $P(\mathbf{w} \mid N^1) = \sum_A P^A(\mathbf{w} \mid A)P(A \mid N^1)$ by a prefix probability $P^A_{\mathrm{pre}}(\mathbf{w} \mid A)$.

Hence given a prefix $\mathbf{w}_k$ with length $k$ as an action sequence, we estimate the most likely goal-nonterminal $A^\dagger$ for $\mathbf{w}_k$ by

$$A^\dagger = \operatorname*{argmax}_A P^A_{\mathrm{pre}}(\mathbf{w}_k)\theta_{N^1 \to A} \tag{1}$$

where $A$ ranges over possible goal-nonterminals. $P^A_{\mathrm{pre}}(\mathbf{w}_k)$ is computed just like $P^{N^1}_{\mathrm{pre}}(\mathbf{w})$ in the previous section.

## 4 Comparative experiment

In this section, we empirically evaluate our method, the *prefix method*, and compare it to two other methods: the PCFG method and logistic regression. The PCFG method naively uses a PCFG. It applies a mixture of PCFG to action sequences $\mathbf{w}_k$. So every sequence is considered as a sentence and the most likely goal-nonterminal $A^*$ is estimated from $\mathbf{w}_k$ by

$$A^* = \underset{A}{\operatorname{argmax}}\, P^A(\mathbf{w}_k)\theta_{N^1 \to A} \tag{2}$$

where $A$ ranges over possible goal-nonterminals. $P^A(\mathbf{w}_k)$ is the probability of $\mathbf{w}_k$ being derived by a component PCFG from goal-nonterminal $A$.

We also compare the prefix method with logistic regression which is a standard discriminative model that does not assume any structure behind data like the prefix and PCFG methods. For a fixed length $k$, the most likely visitor goal is estimated from $\mathbf{w}_k$ considered as a feature vector where features are five basic visitor actions introduced in Section 3.

### 4.1 Data sets and the universal session grammar

We prepare three data sets of action sequences by preprocessing web server logs of U of S (University of Saskatchewan), ClarkNet and NASA [10] in the Internet Traffic Archive [11]. We consider solely for convenience action sequences with length greater than 20 as sentences and exclude those with length greater than 30 as the latter's computation is too costly. As a result we obtain three data sets of action sequences referred to here as U of S, ClarkNet and NASA, each containing 652, 4523 and 2014 action sequences respectively.

With data sets prepared, we next specify a CFG to build a mixture of PCFGs applied to the data sets. To do so in turn requires to determine the number of goal-nonterminals. In other words, we have to know how many goals or intentions visitors have who visit a web site. So we perform clustering on action sequences assuming that one cluster corresponds to one goal, i.e., the number of clusters gives the number of goal-nonterminals.

We use a mixture of PCFGs again for clustering[4] in such a way that although component PCFGs share a common CFG, their parameters are independent. While varying the number of component PCFGs, we choose the clustering giving the highest Bayesian information criterion (BIC) [12]. We construct a small CFG for clustering, containing 30 rules and 12 nonterminals because clustering by a mixture of PCFGs tends to suffers very high memory usage. As a result of this clustering, we got five clusters which are listed in Table 2. Visitors belonging to Survey cluster survey wide area of a website and visitors belonging to News cluster find news and updates of a website. Survey(SpecificAreas) and News(SpecificAreas) clusters are similar to these clusters but visitors belonging

[4] Clustering is done by PRISM.

to them tend to stay in a specific area of a website. Other visitors including those who don't move in a web site along directory structure belong to Other cluster.

**Table 2.** Result of clustering

| Cluster (goal-nonterminal) | Features and major action |
|---|---|
| Survey | `up`/`down` moves in the hierarchy of a website |
| News | `up`/`down` moves in the hierarchy of a website + reload a same page |
| Survey(SpecificAreas) | access to same layer |
| News(SpecificAreas) | access to same layer + reload a same page |
| Other | other |

Finally we manually expand the small CFGs, which we use for clustering, into a large CFG called the *universal session grammar* that has five goal-nonterminals corresponding to five visitor clusters in Table 2. Some rules relating to Survey are listed Table 3. The universal session grammar contains 102 rules and 32 non-terminals and reflects our observation that visitors have different action patterns between initial, middle and end parts in a session.

**Table 3.** Part of the *universal session grammar*

| |
|---|
| S → Survey |
| Survey → InitialSearch Destination EndSearch \| Destination EndSearch |
| Destination→UpDownSearch Search \| UpDownSearch |
| Search→InternalSearch Destination \| InternalSearch |

### 4.2 Measuring accuracy

We apply the three methods to the task of estimating visitors' goals from prefixes of action sequences and compare their accuracy while varying prefix length. We also include a mixture of hidden Markov models (HMMs) in the comparison as a reference method.

To prepare a teacher data set to measure accuracy, we need to label each action sequence generated by a visitor by the visitor's true intention or goal, which is impossible. As substitution, we define a *correct top-goal* for an action sequence in a data set to be the most likely goal-terminal for the sequence estimated by a mixture of PCFGs with the universal session grammar whose

parameters are learned by MLE from the data set. This strategy seems to make sense as long as the universal session grammar is reasonably constructed.

In the experiment[5], accuracy is measured by 5-fold cross-validation for each prefix length $k$ ($2 \leq k \leq 20$). After parameter learning by a training data set, prefixes with length $k$ are cut out from action sequences in the test set and their most likely goal-nonterminals are compared against correct top-goals labeling them. Fig. 4 shows accuracy for each $k$ with standard deviation.

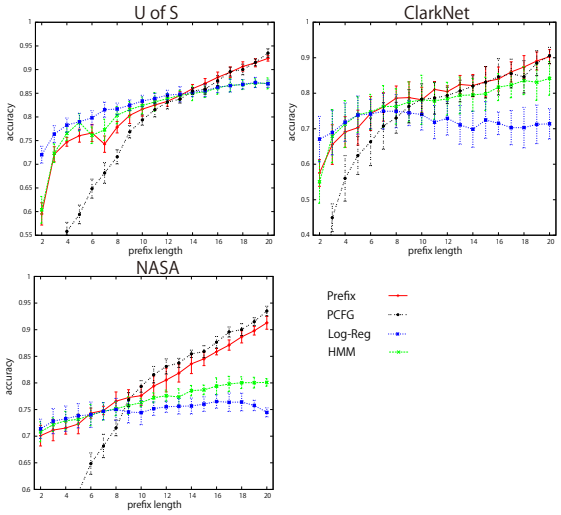


**Fig. 4.** Accuracy for U of S, ClarkNet and NASA

Here Prefix denotes the prefix method, PCFG the PCFG method[6] and Log-Reg logistic regression respectively. We also add HMM for comparison which uses a mixture of HMMs instead of a mixture of PCFGs[7] [8].

Fig. 4 clearly demonstrates that the prefix method and the PCFG method outperform logistic regression and HMM, a standard discriminative model and

[5] It is conducted on a PC with Core i7 Quad 2.67GHz OpenSUSE 11.4 and 72GB main memory.

[6] We apply a PCFG to prefixes by regarding them as sentences. In this experiment, the universal session grammar fails to parse at most two sequences for each dataset, so we ignore these sequences.

[7] We use a left-to-right HMM where the number of states is varied from 2 to 8. In Fig. 4, only the highest accuracy is plotted for each $k$. Since logistic regression only accepts fixed length data, we prepare 19 logistic regression models, one for each length $k$ ($2 \leq k \leq 20$).

[8] We use PRISM [1, 2] to implement a mixture of HMMs and that of PCFG and also to compute prefix probability. For the implementation of logistic regression we use the 'nnet' package of R.

a standard generative model without gramatical information respectively, when prefix is long. Actually all differences at prefix length $k = 20$ in the graph are statistically significant and confirmed by t-test at 0.05 significance level. Also we can observe that the PCFG method rapidly deteriorates when prefix gets shorter though the prefix method keeps fairly good performance comparable to logistic regression and HMM.

## 5 Infix probability computation by parsing

The previous experiment deals with prefixes of action sequences. However if we wish to estimate visitor goals from more incomplete sequences that lack the beginning part in addition to the ending part, we have to compute infix probability.

### 5.1 Nederhof and Satta's algorithm

An *infix* $\mathbf{v}$ in a PCFG $\mathsf{G}$ is a substring of a sentence written as $\mathbf{uvw}$ for some terminal sequences $\mathbf{u}$ and $\mathbf{w}$. The infix probability $P_{\text{in}}^{N^1}(\mathbf{v})$ is defined as

$$P_{\text{in}}^{N^1}(\mathbf{v}) = \sum_{\mathbf{u},\mathbf{w}} P_{\mathsf{G}}(\mathbf{uvw})$$

where $\mathbf{u}$ and $\mathbf{w}$ range over strings such that $\mathbf{uvw}$ is a sentence. According to Nederhof and Satta [8], roughly speaking, $P_{\text{in}}^{N^1}(\mathbf{v})$ is computed by first constructing an intersection PCFG $\mathsf{G}' = \mathsf{G} \cap \mathsf{FA}$ of $\mathsf{G}$ and a finite automaton $\mathsf{FA}$ which accepts every string containing $\mathbf{v}$, and second computing the sum of probabilities of all sentences derived from $\mathsf{G}'$. The second computation is reduced to solving a set of multi-variate polynomial equations (details omitted).

The problem is that although their algorithm is completely general, building the intersection PCFG $\mathsf{G}'$ contains redundancy. Let $A \to BC$ be a CFG rule in $\mathsf{G}$ and $\{s_0, \ldots, s_n\}$ a set of states in $\mathsf{FA}$. To create $\mathsf{G}'$, rules of the form $\langle s_i A s_k \rangle \to \langle s_i B s_j \rangle \langle s_j C s_k \rangle$ are constructed for every possible $i, j, k$ $(0 \le i < j < k \le n)$[9]but some of these rules do not contribute to producing terminal sequences and hence need to be removed.

### 5.2 Infix parsing and cyclic explanation graphs

To avoid blindly combining states and rules and removing them later, we here propose to introduce parsing to their algorithm. More concretely we parse an infix `L` by a PRISM program in Fig. 5. It is a modification of the prefix parser in Fig. 1 that faithfully simulates the parsing action of the intersection PCFG $\mathsf{G}'$.

This program differs from the prefix parser in that an input infix $\mathbf{w} = w_1 \cdots w_n$ is asserted in the memory as a sequence of state transitions: `tr(0,`$w_1$`,1)`,

[9] This is to simulates a state transition of $\mathsf{FA}$ made by a string derived from the nonterminal $A$ using $A \to BC$.

```
values(s,[[s,s],[a],[b]]).
:- set_sw(s,[0.4,0.3,0.3]).

infix_pcfg(L):-                    % L : input infix
   build_FA(L),                    % FA asserted in the memory
   assert_last_state(L,End),       % last_state($(End)) asserted
   start_symbol(C),
   infix_pcfg(0,End,[C]).          % FA transits from state 0 to End

infix_pcfg(S0,S2,[A|R]):-
   ( get_values(A,_) ->            % A : nonterminal
       msw(A,RHS),                 % use A -> RHS to expand A
       infix_pcfg(S0,S1,RHS)
   ; tr(S0,A,S1) ),                % state transition by A from S0 to S1
   ( last_state($(S1)) -> S2 = S1  % pseudo success
   ; infix_pcfg(S1,S2,R) ).
infix_pcfg(S,S,[]).
```

**Fig. 5.** Infix parser $DB_2$

$\ldots$,`tr(`$n-1$`,`$w_n$`,`$n$`)`, together with other transitions constituting the finite automaton FA. In the program, `tr(S0,A,S1)` represents a state transition from `S0` to `S1` by a word `A` in the infix. `infix_pcfg(S0,S2,`$\alpha$`)` reads that $\alpha$, a sequence of terminals and nonterminals, spans a terminal sequence which causes a state transition of FA from `S0` to `S2`. Parsing an infix by the infix parser in Fig. 5 yields an explanation graph which usually is cyclic. For example parsing "`a b`" by running `?- probf(infix_pcfg([$,a,b,$]))`[10]yields an explanation graph shown in Fig. 6 in which `infix_pcfg(0,0,[s,s])` and `infix_pcfg(0,0,[s])` call each other.

An *infix parse tree* for an infix **w** is defined to be a tree constructed by a minimal derivation such that every node except for the leaf nodes spans a leaf subsequence that intersects **w**. We allow here a leaf node to be a nonterminal. The probability of an infix parse tree is defined to be the product of probabilities associated with CFG rules appearing in the tree. Infix parsing by the infix parser in Fig. 5 generates a set of infix parse trees compactly represented as an explanation graph. Although details are omitted due to space limitations, it is possible to extract the most likely infix tree for **w** from **w**'s explanation graph.

Equivalence formulas in the explanation graph are converted to a set of probability equations just like the case of prefix probability computation. From Fig. 6, we obtain, for instance, the probability equations about `X` $= P($`infix_pcfg(0,0,[s,s])`$)$, `Y` $= P($`infix_pcfg(0,0,[b])`$)$ and `Z` $= P($`infix_pcfg(0,0,[s])`$)$ in Fig. 7, which

[10] `probf/1` is a PRISM's built-in predicate and displays an explanation graph.

```
infix_pcfg([$,a,b,$]) <=> infix_pcfg(0,2,[s])
infix_pcfg(0,2,[s])   <=> infix_pcfg(0,2,[s,s]) & msw(s,[s,s])
...
infix_pcfg(0,0,[s,s])
   <=> infix_pcfg(0,0,[b])   & infix_pcfg(0,0,[s]) & msw(s,[b])
     v infix_pcfg(0,0,[s,s]) & infix_pcfg(0,0,[s]) & msw(s,[s,s])
infix_pcfg(0,0,[b]) <=> infix_pcfg(0,0,[])
infix_pcfg(0,0,[s])
   <=> infix_pcfg(0,0,[b])   & infix_pcfg(0,0,[]) & msw(s,[b])
     v infix_pcfg(0,0,[s,s]) & infix_pcfg(0,0,[]) & msw(s,[s,s])
...
infix_pcfg(0,0,[])
```

**Fig. 6.** Part of the explanation graph for infix "a b"

are non-linear (see XZ). We use parameter values $P(\texttt{msw}(\texttt{s},[\texttt{s},\texttt{s}]) = 0.4$ and $P(\texttt{msw}(\texttt{s},[\texttt{b}]) = 0.3$ which are set in the program $DB_2$. Solving the set of equations gives $P(\texttt{infix\_pcfg}(0,0,[\texttt{s},\texttt{s}]) = \texttt{X} = 0.121$ and $P(\texttt{infix\_pcfg}(0,0,[\texttt{s}])$ $= \texttt{Z} = 0.348$. Similarly the infix probability of "a b" is computed as 0.464. As can be seen from this example, unlike prefix probability, probability equations for an infix are usually non-linear and we solve them by Broyden's method, a quasi-Newton method. In this way, we can compute infix probability by way of explanation graphs and estimate a visitor's goal from part of their session logs.

$$P(\texttt{infix\_pcfg}(0,0,[\texttt{s},\texttt{s}]) = \texttt{X} = 0.3 \cdot \texttt{YZ} + 0.4 \cdot \texttt{XZ}$$
$$P(\texttt{infix\_pcfg}(0,0,[\texttt{b}]) = \texttt{Y} = 1$$
$$P(\texttt{infix\_pcfg}(0,0,[\texttt{s}]) = \texttt{Z} = 0.3 \cdot \texttt{Y} + 0.4 \cdot \texttt{X}$$

**Fig. 7.** Probability equations for `infix_pcfg(0,0,[s,s])`, `infix_pcfg(0,0,[b])` and `infix_pcfg(0,0,[s])`

## 6 Conclusion

We have presented the recent progress concerning prefix and infix probability for PCFGs in PRISM. One is an application of prefix probability computation to goal recognition of visitors from their action sequences who visit a website. A comparative experiment using three real datasets is conducted using the prefix

method which we proposed, the PCFG method that always treats action sequences as complete sentence, the HMM method that uses a mixture of HMMs instead of a mixture of PCFGs, and logistic regression. The result demonstrates the superiority of the prefix method for long action sequences. Another is infix probability computation that generalizes prefix probability computation. We proposed to compute it by way of cyclic explanation graphs to delete redundancy in Nederhof and Satta' method [8]. Our approach also gives infix parse trees as side effect. Infix probability computation requires to solve a non-linear set of probability equations unlike prefix probability computation. The implementation of infix probability computation using Broyden's method is underway and will be included in the future release of PRISM.

## References

1. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15** (2001) 391–454
2. 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
3. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In Gopalakrishnan, G., Qadeer, S., eds.: Proceeding of the 23rd International Conference on Computer Aided Verification (CAV'11). Volume 6806 of LNCS., Springer (2011) 585–591
4. Jelinek, F., Lafferty, J.: Computation of the probability of initial substring generation by stochastic context-free grammars. Computational Linguistics **17**(3) (1991) 315–323
5. Stolcke, A.: An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computational Linguistics **21**(2) (1995) 165–201
6. Gorlin, A., Ramakrishnan, C., Smolka, S.: Model checking with probabilistic tabled logic programming. Theory and Practice of Logic Programming (TPLP) **12(4-5)** (2012) 681–700
7. Sato,T. and Meyer,P.: Infinite probability computation by cyclic explanation graphs. Theory and Practice of Logic Programming (TPLP) DOI: `http://dx.doi.org/10.1017/S1471068413000562`, published online, Nov. 04 (2013) 1–29
8. Nederhof, M., Satta, G.: Computation of infix probabilities for probabilistic context-free grammars. In: Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP'11). (2011) 1213–1221
9. Wetherell, C.S.: Probabilistic languages: a review and some open questions. Computing Surveys **12**(4) (1980) 361–379
10. Arlitt, M.F., Williamson, C.L.: Web server workload characterization: The search for invariants. In: ACM SIGMETRICS Performance Evaluation Review. Volume 24. (1996) 126–137
11. ITA: The Internet Traffic Archive. http://ita.ee.lbl.gov/ (2001)
12. Schwarz, G.: Estimating the dimension of a model. Annals of Statistics **6**(2) (1978) 461–464