A New Probabilistic LR Language Model for Statistical Parsing

INUI Kentaro, Virach SORNLERTLAMVANICH, TANAKA Hozumi, and TOKUNAGA Takenobu

TR97-0004 April

DEPARTMENT OF COMPUTER SCIENCE
TOKYO INSTITUTE OF TECHNOLOGY
Ôokayama 2-12-1 Meguro Tokyo 152, Japan
http://www.cs.titech.ac.jp/

©The author(s) of this report reserves all the rights.
Abstract

This paper presents a newly formalized probabilistic LR language model. Our model inherits its essential features from Briscoe and Carroll’s generalized probabilistic LR (PLR) model [3], which obtains context-sensitivity by assigning a probability to each LR parsing action according to its left and right context. However, our model is simpler while maintaining a higher degree of context-sensitivity as compared to Briscoe and Carroll’s model. In this paper, we first formalize our PLR model and enumerate some of its features. We then discuss the differences between Briscoe and Carroll’s model and ours. We also qualitatively compare a model based on canonical LR with one based on lookahead LR.

1 Introduction

The increasing availability of text corpora has been encouraging researchers to explore statistical approaches for various tasks in natural language processing. Statistical parsing is one of these approaches. In statistical parsing, one of the most straightforward methodologies is to generalize context-free grammars by associating probability with each rule in producing probabilistic context-free grammars (PCFGs). However, as many researchers have already pointed out, PCFGs are not quite adequate for statistical parsing due to the lack of context-sensitivity. Probabilistic LR parsing is one existing statistical parsing methodology which is more context-sensitive than PCFG-based parsing.

Several attempts have been made to incorporate probability into generalized LR (GLR) parsing [14]. For example, Wright and Wrigley proposed an algorithm to distribute probabilities originally associated with CFG rules to LR parsing actions, in such a way that the resulting model is equivalent to the original PCFG [16]. However, since their principal concern was in compiling PCFGs into the GLR parsing framework, their language model failed to capture the context-sensitivity of languages.

Su et al. proposed a way of introducing probabilistic distribution into the shift-reduce parsing framework [13]. Unlike Wright and Wrigley’s work, the goal of this research was the construction of a mildly context-sensitive model that is effective for statistical parsing. Their model distributes probabilities
to stack transitions between two shift actions, and associates each parse
derivation with a probability, given by the product of the probability of each
change included in the derivation. However, although they described an
algorithm to handle this model in the GLR parsing framework, gaining parse
efficiency, their probabilistic model itself is not intimately coupled with the
GLR parsing algorithm. As a result, their model needs an additional complex
algorithm for training. In addition, to implement a statistical parser with
their model, significant extensions to the GLR parsing algorithm are required.

On the other hand, Briscoe and Carroll proposed the distribution of
probabilities directly to each action in an LR table to realize mildly context-
sensitive parsing [3]. Their model overcomes the drawback of context-insensitivity
of PCFGs by estimating the probability of each LR parsing action according
to its left and right context. This model distributes probabilities to actions
associated with each LR parse state so that the probabilities of the transi-
tions from any given state sum up to one. The probability of each parse
derivation is computed as the product of the probability assigned to each ac-
tion included in the derivation. Unlike the approach of Su et al., this makes
it easy to implement context-sensitive probabilistic parsing by slightly ex-
tending GLR parsers, and the probabilities can be easily trained simply by
counting the frequency of application of each action in parsing the training
sentences. However, Briscoe and Carroll’s model seems to be rather expe-
riential since sufficiently well-founded formalization is lacking, which may
degrade its performance.

This paper presents a newly formalized probabilistic LR language model
(PLR model) for statistical parsing, and reviews Briscoe and Carroll’s model
in terms of our formalization, suggesting some of its drawbacks. In what
follows, we first formalize our PLR model and enumerate some of its ad-
vantages (section 2). We then qualitatively compare Briscoe and Carroll’s
model with ours in terms of the number of free parameters and the context-
sensitivity (section 3). We finally present a brief example to demonstrate the
advantages of our model (section 4).

2 A Probabilistic LR Language Model

Suppose we have a CFG and its corresponding LR table. Let $V_n$ and $V_t$
be the nonterminal and terminal alphabets, respectively, of the CFG. Further,
let $\mathbf{S}$ and $\mathbf{A}$ be the sets of LR parse states and parsing actions appearing in
the CLR table, respectively. For each state $s \in \mathbf{S}$, the CLR table specifies
a set $La(s) \subseteq V_i$ of possible next input symbols. Further, for each coupling
of a state $s$ and input symbol $l \in La(s)$, the table specifies a set of possible
parsing actions: $Act(s, l) \subseteq \mathbf{A}$. Each action $a \in \mathbf{A}$ is either a shift action
or reduce action. Let $\mathbf{A}_s$ and $\mathbf{A}_r$ be the set of shift and reduce actions,
respectively, such that $\mathbf{A} = \mathbf{A}_s \cup \mathbf{A}_r \cup \{\text{accept}\}$ (accept is a special action
denoting the completion of parsing).

As with most statistical parsing frameworks, given an input sentence, we
rank the parse tree candidates according to the probabilities of the parse
derivations that generate those trees. In LR parsing, each parse derivation
can be regarded as a sequence $T$ of transitions between LR parse stacks, which
we will describe in detail below. Thus, in the following, we use the terms
parse tree, parse derivation, and stack transition sequence interchangeably.

Given an input word sequence $W = \{w_1, \ldots, w_n\}$, we estimate the distri-
bution over the parse tree candidates $T$ as follows:

$$P(T|W) = \alpha_W \cdot P(T) \cdot P(W|T)$$ (1)

The first scaling factor $\alpha_W$ is a constant depending only on $W$, and thus does
not need to be considered in ranking parse trees. The second factor $P(T)$ is
the distribution over all the possible trees that can be derived from a given
grammar. We estimate it using a PLR model. The third factor $P(W|T)$ is
the distribution of lexical derivations from $T$, where each terminal symbol
of $T$ is assumed to be a part of speech symbol. Most statistical parsing
frameworks estimate this distribution by assuming that the probability of the
$i$-th word $w_i \in W$ depends only on its corresponding terminal symbol
(i.e. part of speech) $l_i$. Since $l_i$ is uniquely specified by $T$ for each $i$, we
obtains equation (2).

$$P(W|T) \approx \prod_{w_i \in W} P(w_i|l_i)$$ (2)

One could use a more context-sensitive model to estimate the lexical dis-
tribution $P(W|T)$; for example, one could introduce the statistics of word
collocations. However, this is beyond the scope of this paper. For further
discussion, see [6].

A stack transition sequence $T$ can be described as (3):

$$\sigma_0 \xrightarrow{l_1a_1} \sigma_1 \xrightarrow{l_2a_2} \cdots \xrightarrow{l_{n-1}a_{n-1}} \sigma_{n-1} \xrightarrow{l_na_n} \sigma_n$$ (3)
where \( \sigma_i \) is the \( i \)-th stack, whose stack-top state is denoted by \( \text{top}(\sigma_i) \), and \( l_i \in \text{La}(\text{top}(\sigma_{i-1})) \) and \( a_i \in \text{Act}(\text{top}(\sigma_{i-1}), l_i) \) are, respectively, a input symbol and a parsing action chosen at \( \sigma_{i-1} \). It can be proven from the LR parsing algorithm that, given a derived input symbol \( l_{i+1} \in \text{La}(\text{top}(\sigma_i)) \) and an action \( a_{i+1} \in \text{Act}(\text{top}(\sigma_i), l_{i+1}) \), the next (derived) stack \( \text{next}(\sigma_i, a_{i+1}) (= \sigma_{i+1}) \) can always be uniquely determined as follows:

- If the current action \( a_{i+1} \) is a shift action for an input symbol \( l \), then the parser consumes \( l \), pushing \( l \) onto the stack, and then pushes the next state \( s_{i+1} \), which is uniquely specified by the LR table, onto the stack.

- If the current action \( a_{i+1} \) is a reduction by a rule \( A \rightarrow \beta \), the parser derives the next stack as follows. The parser first pops \(|\beta|\) symbols off the stack, where \(|\beta|\) is the length of \( \beta \), exposing the stack-top state \( \sigma_i \). The parser then pushes \( A \) and \( s_j \), the entry for \( s_i \) and \( A \) specified in the goto part of the LR table, onto the stack. All these operations are executed deterministically.

Hereafter, we consistently refer to an LR parse state as a state and an LR parse stack as a stack. And, unless defined explicitly, \( s_i \) denotes the stack-top state of the \( i \)-th stack \( \sigma_i \), i.e., \( s_i = \text{top}(\sigma_i) \).

A parse derivation completes either if \( l_n = \$$ and \( a_n = \text{accept} \) or if \( \text{Act}(s_{n-1}, l_n) = \$$). In the former case, we say stack transition sequence \( T \) is accepted, and \( \sigma_n = \text{final} \), where \( \text{final} \) is a dummy symbol denoting the stack when parsing is completed. In the latter case, on the other hand, we say stack transition sequence \( T \) is rejected. Let \( \mathcal{T} \) and \( \mathcal{T}_{\text{acc}} \) be the infinite set of all possible complete stack transition sequences and the infinite set of all possible complete and acceptable stack transition sequences. Note that, if one uses canonical LR (CLR) tables as the source of modeling, every transition sequence in \( \mathcal{T} \) is acceptable, whereas, if one uses lookahead LR (LALR) tables, certain transition sequences of \( \mathcal{T} \) may be rejected.\(^1\)

The second factor \( P(T) \) in (1), which we would like to estimate here, is a distribution over complete and acceptable stack transition sequences such that the probabilities of all the transition sequences in \( \mathcal{T}_{\text{acc}} \) sum up to one. However, if one considers an LALR-based model, since there may be rejected

\(^1\)For details of CLR and LALR, see [1], [4], etc.
transition sequences in $\mathcal{T}$, equation (1) should be replaced with (4):

$$P(T|W) = \alpha_W \cdot P(T|T \in \mathcal{T}_{acc}) \cdot P(W|T)$$

(4)

$$= \alpha_W \cdot \left( \sum_{T \in \mathcal{T}_{acc}} P(T) \right)^{-1} \cdot P(T) \cdot P(W|T)$$

(5)

$$= \alpha' \cdot P(T) \cdot P(W|T)$$

(6)

where $\alpha'$ is a constant that is independent of $T$, and $P(T)$ is a distribution over all the possible complete transition sequences such that:

$$\sum_{T \in \mathcal{T}} P(T) = 1$$

(7)

Thus, one can rank the parse tree candidates for any given input sentence according to $P(T)$ and $P(W|T)$, whether one bases the model on CLR or LALR. We will make further discussion of the differences between CLR-based and LALR-based models in Appendix A.

The probability of a complete stack transition sequence $T$ can be decomposed as:

$$P(T) = P(\sigma_0, l_1, a_1, \sigma_1, \ldots, \sigma_{n-1}, l_n, a_n, \sigma_n)$$

(8)

$$= P(\sigma_0) \cdot \prod_{i=1}^{n} P(l_i, a_i, \sigma_i|\sigma_0, l_1, a_1, \sigma_1, \ldots, l_{i-1}, a_{i-1}, \sigma_{i-1})$$

(9)

Here we assume that $\sigma_i$ contains all the information of its preceding parse derivation, namely:

$$P(\sigma_0, l_1, a_1, \sigma_1, \ldots, \sigma_{i-1}, l_i, a_i|\sigma_i) = 1$$

(10)

This assumption simplifies equation (9) to:

$$P(T) = \prod_{i=1}^{n} P(l_i, a_i, \sigma_i|\sigma_{i-1})$$

(11)

Now, we show how we estimate each transition probability $P(l_i, a_i, \sigma_i|\sigma_{i-1})$, which can be decomposed as in (12):
Case 1. \( i = 1 \):

\[
P(l_1|\sigma_0) = P(l_1|s_0)
\]  
(13)

Case 2. The previous action \( a_{i-1} \) is a shift action, i.e. \( a_{i-1} \in A_s \). We assume that only the current stack-top state \( s_{i-1} = \text{top}(\sigma_{i-1}) \) has any effect on the probability of the next input symbol \( l_i \). This means that:

\[
P(l_i|\sigma_{i-1}) \approx P(l_i|s_{i-1})
\]  
(14)

where

\[
\sum_{l \in \text{La}(s)} P(l|s) = 1
\]  
(15)

Case 3. The previous action \( a_{i-1} \) is a reduce action, i.e. \( a_{i-1} \in A_r \). Unlike Case 2, in case of a reduce action, the input symbol does not get consumed, and thus the next input symbol \( l_i \) is always identical to \( l_{i-1} \); namely, \( l_i \) can be deterministically predicted. Therefore,

\[
P(l_i|\sigma_{i-1}) = 1
\]  
(16)

Then, we estimate the second term \( P(a_{i}|\sigma_{i-1}, l_i) \) relying on the analogous assumption that only the current stack-top state \( s_{i-1} \) and input symbol \( l_i \) have any effect on the probability of the next action \( a_i \):

\[
P(a_i|\sigma_{i-1}, l_i) \approx P(a_i|s_{i-1}, l_i)
\]  
(17)

where

\[
\sum_{a \in \text{Act}(s,l)} P(a|s,l) = 1
\]  
(18)

Finally, as mentioned above, given the current stack \( \sigma_{i-1} \) and action \( a_i \), the next stack \( \sigma_i \) can be uniquely determined:

\[
P(\sigma_i|\sigma_{i-1}, l_i, a_i) = 1
\]  
(19)

As shown in equations (14) and (16), the probability \( P(l_i|\sigma_{i-1}) \) should be estimated differently depending on whether the previous action \( a_{i-1} \) is a shift action or a reduce action. Let \( S_i \) be the set of states reached immediately
after applying a shift action, and \( S_r \) be those reached immediately after applying a reduce action:

\[
S_s \overset{\text{def}}{=} \{ s_0 \} \cup \{ s \mid \exists a \in A_s, \sigma : s = \text{top}(\text{next}(\sigma, a)) \}
\]

\[
S_r \overset{\text{def}}{=} \{ s \mid \exists a \in A_r, \sigma : s = \text{top}(\text{next}(\sigma, a)) \}
\]

(20)

(21)

where \( s_0 \) is the initial state. Note that these two sets are mutually exclusive\(^2\):

\[
S = S_s \cup S_r \quad \text{and} \quad S_s \cap S_r = \emptyset
\]

(22)

Equations (12) through (21) can be summarized as:

\[
P(l_i, a_i, \sigma_{i-1}) \approx \begin{cases} 
P(l_i, a_i|s_{i-1}) & \text{(for } s_{i-1} \in S_s) \\ 
P(a_i|s_{i-1}, l_i) & \text{(for } s_{i-1} \in S_r) \end{cases}
\]

(23)

Since \( S_s \) and \( S_r \) are mutually exclusive, we can assign a probability to each action in the action part of an LR table, according to equation (23). To be more specific, for each state \( s \in S_s \), we associate a probability \( p(a) \) with each action \( a \in \text{Act}(s, l) \) (for \( l \in \text{La}(s) \)), where \( p(a) = P(l, a|s) \) such that:

\[
\sum_{l \in \text{La}(s)} \sum_{a \in \text{Act}(s,l)} p(a) = 1 \quad \text{(for } s \in S_s) 
\]

(24)

On the other hand, for each state \( s \in S_r \), we associate a probability \( p(a) \) with each action \( a \in \text{Act}(s, l) \) (for \( l \in \text{La}(s) \)), where \( p(a) = P(a|s, l) \) such that:

\[
\sum_{a \in \text{Act}(s,l)} p(a) = 1 \quad \text{(for } s \in S_r) 
\]

(25)

Through assigning probabilities to actions in an LR table in this way, we can estimate the probability of a stack transition sequence \( T \) as given in (3)

\(^2\)It is obvious from the algorithm for generating an LR(1) goto graph\(^1\) that, for each state \( s \neq s_0 \), if there exist states \( s_i \) and \( s_j \) whose goto transitions on input symbol \( l_i \) and \( l_j \), respectively, both lead to \( s \), then \( l_i = l_j \). Namely, for any given state \( s \), the symbol \( l \) required to reach \( s \) by way of a goto transition is always uniquely specified. On the other hand, if the current state is in \( S_s \), then it should have been reached through a goto transition on a certain terminal symbol \( l \in V_t \), whereas, if the current state is in \( S_r \), then it should have been reached through a goto transition on a certain nonterminal symbol \( l \in V_n \). Given these facts, it is obvious that \( S_s \) and \( S_r \) are mutually exclusive.
by computing the product of the probabilities associated with all the actions included in $T$:

$$P(T) = \prod_{i=1}^{n} p(a_i)$$  \hspace{1cm} (26)

Before closing this section, we describe three features of our PLR model that make it appropriate for statistical parsing. First, since the probability of each parse derivation can be estimated simply as the product of the probabilities associated with all the actions in that derivation, we can easily implement a probabilistic LR parser through a simple extension to the original LR parser. We can also easily train the model, as we need only count the frequency of each action applied to generate correct parse derivations in the training corpus and then normalize according to equation (24) or (25).

Second, as in equation (17), the probabilistic distribution of each parsing action depends on both its left context (i.e. LR parse state) and right context (i.e. input symbol). This makes our model mildly context-sensitive. We will elaborate this using a concrete example later in section 4.

Third, PCFGs give long-term preference over structures but do not sufficiently reflect short-term bigram statistics of terminal symbols, however our PLR model reflects both types of preference. $P(l_i|s_{i-1})$ in equation (14) is a model that predicts the next terminal symbol $l_i$ for the current left context $s_{i-1} \in S_s$. In this case of $s_{i-1} \in S_s$, since $s_{i-1}$ uniquely specifies the previous terminal symbol $l_{i-1}$, $P(l_i|s_{i-1}) = P(l_i|s_{i-1}, l_{i-1})$, which is a slightly more context-sensitive version of the bigram model of terminal symbols $P(l_i|l_{i-1})$. This feature is significant particularly when one attempts to integrate syntactic parsing with morphological analysis in the GLR parsing framework [9].

3 Comparison with Briscoe and Carroll’s Model

In this section, we briefly review Briscoe and Carroll’s model [3] and make a qualitative comparison between their model and ours.

We consider the probabilities of transitions between stacks as given in equation (11), whereas Briscoe and Carroll consider the probabilities of transitions between $LR$ parse states as below:

$$P(T) \approx \prod_{i=1}^{n} P(l_i, a_i, s_i|s_{i-1})$$  \hspace{1cm} (27)
\[ = \prod_{i=1}^{n} P(l_i, a_i|s_{i-1}) \cdot P(s_i|s_{i-1}, l_i, a_i) \]  

(28)

Briscoe and Carroll initially associate a probability \( p(a) \) with each action \( a \in Acc(s, l) \) (for \( l \in La(s) \)) in an LR table, where \( p(a) \) corresponds to \( P(l_i, a_i|s_{i-1}) \), the first term in (28):

\[ p(a) = P(l, a|s) \]  

(29)

such that:

\[ \forall s \in S. \sum_{l \in La(s)} \sum_{a \in Acc(s, l)} p(a) = 1 \]  

(30)

In this model, for any state, the probability associated with each action is normalized in the same manner. However, as discussed in the previous section, the probability assigned to an action should be normalized differently depending on whether the state associated with the action is of class \( S_s \) or \( S_r \) as in equations (24) and (25). Without this difference, probability \( P(l|s_{i-1}) \) in equation (14) could be duplicated for a single terminal symbol. As a consequence, in Briscoe and Carroll’s formulation, the probabilities of all the complete parse derivations may not sum up to one, which would be inconsistent with the definition of \( P(T) \) (see equation (7)).

Briscoe and Carroll are also required to include the second term \( P(s_i|s_{i-1}, l_i, a_i) \) in (28) since it seems that it is not always one. In general, if we have only the information of the current state and apply a reduce action, we cannot always uniquely determine the next state. For this reason, Briscoe and Carroll further subdivide probabilities assigned to reduce actions according to the stack-top states exposed immediately after the stack-pop operations associated with those reduce action are executed. Contrastively, in our model, given the current stack, the next stack after applying any action can be uniquely determined as in (19), and thus we do not need to subdivide the probability for any reduce action.

One may argue that this is only a matter of the degree of context-sensitivity; namely, Briscoe and Carroll's model is less context-sensitive than ours, which causes the nondeterminism in states reached after reduce actions. However, it is also important to compare the number of free parameters in the two model, in order to gauge their relative complexity. In Briscoe and Carroll's model, since probabilities assigned to actions are normalized for
each state, the number $N$ of free parameters is:

$$N = \sum_{s \in \mathcal{S}} \left( \sum_{l \in \text{Lo}(s)} |\text{Act}(s, l)| - 1 \right) = |\mathcal{A}| - |\mathcal{S}| \quad (31)$$

Furthermore, since Briscoe and Carroll’s model subdivides probabilities assigned to reduce actions according to the states reached after the stack-pop operation associated with those reduce actions, the number of free parameters can be more than $N$. On the other hand, in our model, since, for each state in class $\mathcal{S}_r$, probabilities assigned to actions are normalized for each input symbol, the number $N'$ of free parameters is less than $N$, and thus, less than that of Briscoe and Carroll’s model:

$$N' = \sum_{s \in \mathcal{S}_s} \left( \sum_{l \in \text{Lo}(s)} |\text{Act}(s, l)| - 1 \right) + \sum_{s \in \mathcal{S}_r} \sum_{l \in \text{Lo}(s)} (|\text{Act}(s, l)| - 1) < |\mathcal{A}| - |\mathcal{S}| \quad (32)$$

To sum up, one can safely state that our model is simpler, while maintaining a higher degree of context-sensitivity, than Briscoe and Carroll’s model. That is, our model requires less training data, while being expected to achieve higher context-sensitivity.

4 An Example

In this section, we present an example to demonstrate how our model partially captures the context-sensitivity of language, comparing it with a PCFG model and Briscoe and Carroll’s model.

Consider a simple grammar $G_1$ (assuming that the target language is Japanese) as follows:

Grammar $G_1$:

(1) $S \rightarrow \text{conj} \ S$
(2) $S \rightarrow \text{pp} \ S$
(3) $S \rightarrow \text{v}$
(4) $S \rightarrow S \ S$

10
Further, let us assume that we train the PCFG model, Briscoe and Carroll’s model, and our PLR model, respectively, using an artificial training set as shown in Figure 1. Table 1 shows the CLR table for grammar $G$ and the parameters trained using the training set in Figure 1. For each state, the numbers in the middle and bottom rows denote the parameters for our model and Briscoe and Carroll’s model, respectively. In practical applications, when computing parameters, one would need to use some smoothing technique in order to avoid assigning zero to any parameter associated with an action that had never occurred in training. We further discuss Briscoe and Carroll’s model later in this section.

Table 2 shows the distributions given by the above three models to each tree shown in Figure 1, where the trees (a) through (c) are parse trees for a sentence “conj pp v v”, and the trees (d) through (f) are for “pp pp v v”.

Let us first see how our PLR model realizes mildly context-sensitive statistic parsing, as compared to the PCFG model. According to the training set in Figure 1, (a) is most preferable for the parse tree of the sentence “conj
Table 1: CLR table for grammar G, with trained parameters

<table>
<thead>
<tr>
<th>state</th>
<th>conj</th>
<th>pp</th>
<th>action</th>
<th>goto</th>
<th>$</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>re3</td>
<td>re3</td>
<td>re3</td>
<td>re3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td>acc</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>sh1/re1</td>
<td>sh2/re1</td>
<td>sh3/re1</td>
<td>re1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>sh1/re2</td>
<td>sh2/re2</td>
<td>sh3/re2</td>
<td>re2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sh1/re4</td>
<td>sh2/re4</td>
<td>sh3/re4</td>
<td>re4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state</th>
<th>conj</th>
<th>pp</th>
<th>action</th>
<th>goto</th>
<th>$</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>re3</td>
<td>re3</td>
<td>re3</td>
<td>re3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>sh1</td>
<td>sh2</td>
<td>sh3</td>
<td>acc</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>sh1/re1</td>
<td>sh2/re1</td>
<td>sh3/re1</td>
<td>re1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>sh1/re2</td>
<td>sh2/re2</td>
<td>sh3/re2</td>
<td>re2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sh1/re4</td>
<td>sh2/re4</td>
<td>sh3/re4</td>
<td>re4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Table 2: Distributions over the parse trees in Figure 1

<table>
<thead>
<tr>
<th></th>
<th>P(a)</th>
<th>P(b)</th>
<th>P(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFG</td>
<td>$1.0 \times 10^{-3}$</td>
<td>$1.0 \times 10^{-3}$</td>
<td>$1.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>B&amp;C</td>
<td>$3.4 \times 10^{-4}$</td>
<td>$3.6 \times 10^{-4}$</td>
<td>$4.0 \times 10^{-9}$</td>
</tr>
<tr>
<td>PLR</td>
<td>$5.5 \times 10^{-2}$</td>
<td>$2.7 \times 10^{-2}$</td>
<td>$1.1 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P(d)</th>
<th>P(e)</th>
<th>P(f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFG</td>
<td>$2.7 \times 10^{-3}$</td>
<td>$2.7 \times 10^{-3}$</td>
<td>$2.7 \times 10^{-3}$</td>
</tr>
<tr>
<td>B&amp;C</td>
<td>$1.1 \times 10^{-6}$</td>
<td>$6.7 \times 10^{-5}$</td>
<td>$2.8 \times 10^{-10}$</td>
</tr>
<tr>
<td>PLR</td>
<td>$2.6 \times 10^{-3}$</td>
<td>$1.8 \times 10^{-2}$</td>
<td>$2.9 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

$pp \; v \; v'$, and (e) for "conj $pp \; v \; v'$". As shown in Table 2, our PLR model properly learns these preferences, whereas the PCFG model gives an equal probability to all the parse tree candidates for each sentence.

In our model, the preference for (a) over (b) is mostly derived from the distribution for the shift-reduce conflict in state 5 with the next input symbol being $v$. This is illustrated in (a) and (b) in Figure 1, where each circled number denotes the LR parse state reached after parsing has proceeded from the left-most corner to the location of that number. After $pp$ and $S$ are reduced together to $S$, where the current state is state 5, if the shift action is chosen, parse tree (a) is generated, while, if the reduce action is chosen, (b) is generated. According to Table 1, our model prefers the shift action in state 5 with the next input symbol being $v$. This preference leads the parser to prefer parse tree (a) to (b). Contrastively, in parsing sentence "$pp \; pp \; v \; v'$", the parsing process reaches state 6 (instead of 5) after the second $pp$ and $S$ are reduced together to $S$. In state 6 with the next input symbol being $v$, the reduce action is preferred. Thus, the parser prefers parse tree (e) to (d). As shown in this example, each LR parse state gives part of the information of the preceding parsing process (e.g. whether the symbol below the current

$^3$These preferences are based on the general tendencies observed in Japanese texts such that a conjunction (conj) at the beginning of a sentence tends to modify the rest of the sentence at the top-most level, and a postpositional phrase (pp) tends to be subordinated by its left-most verb ($v$).
stack-top symbol S is conj or pp as in the above example), and the PLR model is sensitive to this left context information. Furthermore, although not explicitly demonstrated in the above example, it should be also noted that the PLR model is also sensitive to the next input symbol as shown in (17) in section 2.

Next, let us compare our model with Briscoe and Carroll’s model. As mentioned in section 3, Briscoe and Carroll’s model is different from our model in the following two respects:

(a) In our model, the probability assigned to each action is normalized differently depending on whether the state associated with the action is of class $S_8$ or $S_r$, whereas Briscoe and Carroll’s model does not take this into account. This difference appears, for example, in states 4 and 5 of Table 1, which are both of class $S_r$.

(b) In Briscoe and Carroll’s model, probabilities assigned to reduce actions are subdivided according to the states reached after them. This is shown, for example, in the parameters associated with the reduce action in state 3 with the next input symbol being “$”.

For these reasons, Briscoe and Carroll’s model contains more free parameters than our model does. This is shown in Table 3. Note that, in training Briscoe

<table>
<thead>
<tr>
<th></th>
<th># of free parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFG</td>
<td>3</td>
</tr>
<tr>
<td>B&amp;C</td>
<td>36</td>
</tr>
<tr>
<td>PLR</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 3: Number of free parameters contained in each model

and Carroll’s model in this example, we explicitly subdivided the parameter for a reduce action only when it actually led the parser to different states in our training set. Thus, if we trained the model using a training set containing more diverse trees, the number of free parameters could become even larger.

Let us return to Table 2. The table shows that, unlike our model, Briscoe and Carroll’s model does not correctly prefer parse tree (a) for sentence “conj
pp v v⁴. The reason can be analyzed as follows. Like our model, Briscoe and Carroll’s model correctly prefers the shift action in state 5. However, for the rest of the parsing process, Briscoe and Carroll’s model associate a higher probability with the process from state 4 through 3 and 7 to 4, which derives tree (b), than the process from 3 through 7 and 5 to 4, which derives tree (a), since, in Briscoe and Carroll’s model, the former process is incorrectly supported by the occurrence of tree (e). For example, Briscoe and Carroll’s model subdivides the probability of the reduce action in state 3 with the next input symbol being $ (\text{referred to as (p)}$ and $(q)$ in Figure 2) according to the state reached after the stack-pop operation; thus, since event $(q)$ occurs not only in tree (b) but also in (e), the probability of $(q)$ becomes higher than that of $(p)$ (.25/.17). On the other hand, in our model, the preference for (a) to (b) is not influenced by the occurrence of tree (e). This example may seem to be relatively artificial and forced. However, our preliminary experiments using real corpora have, so far, supported our claim. In fact, such cases frequently occur, and significantly degrade the performance of Briscoe and Carroll’s model (for further detail on our experiments, see [12]).

5 Conclusion

In this paper, we formalized a probabilistic LR language model. The model is mildly context-sensitive, and naturally integrates short-term bigram statistics of terminal symbols and long-term preference over structures of parse trees, which are both expected to be appropriate for statistical parsing. Furthermore, since the model is tightly coupled with GLR parsing, it can be easily implemented and trained. We then claimed that that our model is simpler while maintaining a higher degree of context-sensitivity even higher as compared to Briscoe and Carroll’s model. However, large-scaled experimental comparisons are clearly needed. We are now conducting experiments

⁴Although Briscoe and Carroll proposed to take the geometric mean of peripheral distributions to prevent the system from always preferring parse trees involving fewer grammar rules almost regardless of training data, we did not apply this operation when computing the probabilities in Table 2 to give the reader a sense of the difference between the probabilities given by Briscoe and Carroll’s model and our model. Note that, in our example, since the number of state transitions for each parse tree of any given sentence is always the same, given that the grammar generates only binary trees, taking the geometric mean would not change the preference order.
for parsing of Japanese sentences, some of which are presented in [12].

Other approaches to statistical parsing using context-sensitive language models have also been proposed, such as [2, 7, 10]. We need to make theoretical and experimental comparisons between these models and ours. The significance of introducing lexical sensitivity into language models should also not be underestimated. In fact, several attempts to use lexically sensitive models already exist: e.g., [5, 8, 11]. Our future research will be directed at this area [6].

References


Appendix: Comparison between CLR-based and LALR-based models

In practical applications, it is often preferable to use lookahead LR (LALR) tables instead of CLR tables since the size of an LALR table tends to be much smaller than that of the CLR table for the same grammar, while operating in the same fashion as a CLR parser through use of stored lookahead information. In fact, Briscoe and Carroll use LALR tables as the source of their models, and our modeling is also perfectly applicable to LALR. Although it is a highly empirical issue which performs better, CLR-based models or LALR-based models, it may be still worthwhile to make a qualitative comparison between them.

Let us first briefly review the relationship between CLR and LALR. An LALR table can be generated from the corresponding CLR table as follows\(^5\). Hereafter, we refer to the CLR item set corresponding to a CLR state \(s_i\) as \(\mathcal{I}_i\). Further, we refer to LALR states and LALR item sets with a superscript \(l\), such as \(s'_j\), \(\mathcal{I}'_j\), etc. Given a collection \(\mathcal{I}\) of sets of CLR items, the corresponding collection \(\mathcal{I}'\) of LALR item sets is generated by merging CLR item sets with the same core, as follows:

\[
\mathcal{I}' = \{ \mathcal{I}' \subseteq \mathcal{I} : \mathcal{I}' = LAIS(\mathcal{I}) \} \tag{33}
\]

\(LAIS: \mathcal{I} \mapsto \mathcal{I}'\) is the function that returns the LALR item set corresponding to an original item set in \(\mathcal{I}\):

\[
LAIS(\mathcal{I}) \overset{\text{def}}{=} \bigcup_{\exists i' \in \mathcal{I} : \text{core}(i') = \text{core}(i')} \mathcal{I}' \tag{34}
\]

\(^5\)Although it is known that there are more efficient algorithms for LALR table generation, here we consider translation from CLR to LALR to focus on the relationship between them.
The core of a CLR item set \( \mathbf{i} \) is defined as:

\[
\text{core}(\mathbf{i}) \overset{\text{def}}{=} \{ [A \rightarrow \alpha \cdot \beta] \mid [A \rightarrow \alpha \cdot \beta, \mathbf{l}] \in \mathbf{i} \}
\]  

(35)

The parsing actions for each LALR state \( s'_k \) are constructed from \( s'_i \) in the same manner as in the construction of CLR tables.

Suppose that LR item sets \( \mathbf{i}_i, \mathbf{i}_j \in \mathbf{I} \) are both mapped to the same LALR item set \( s'_k \in \mathbf{I}' \). By the definition of the mapping \( \text{LAIS} \), the following lemmata are obvious:

1. If \([A \rightarrow \alpha \cdot a\beta, u] \in \mathbf{i}_i \) for \( a, u \in \mathbf{V}_i \), then \([A \rightarrow \alpha \cdot a\beta, v] \in \mathbf{i}_j \) for some \( v \in \mathbf{V}_j \). Therefore, as illustrated in Figure 3, if \( \text{Act}(s_i, a) \) contains a shift action for some input symbol \( a \in \mathbf{L}(s_i) \), then \( \text{Act}(s_j, a) \) also contains a shift action, and thus, \( \text{Act}(s'_k, a) \) also contains a shift action.

\[
\begin{array}{c|c|c}
\cdots & a & \cdots \\
\hline
s_i & \text{sh}_x & \Rightarrow \\
\hline
s_j & \text{sh}_y \\
\hline
\end{array}
\quad
\begin{array}{c|c|c}
\cdots & a & \cdots \\
\hline
\Rightarrow & s'_k & \text{sh}_z \\
\end{array}
\]

Figure 3: Merging CLR states associated with shift actions

2. If \([A \rightarrow \alpha, u] \in \mathbf{i}_i \) for \( u \in \mathbf{L}(s_i) \), then \( \text{Act}(s_i, u) \) contains a reduce action (reduce by \( A \rightarrow \alpha \)), however, \( \text{Act}(s_j, u) \) may not contain the same reduce action since the corresponding LR item in \( \mathbf{i}_j \) may be \([A \rightarrow \alpha, v] \) for \( v \in \mathbf{L}(s_j), v \neq u \). This case is illustrated in Figure 4.

Let us compare models based on CLR and LALR, again according to the number of free parameters and the degree of context-sensitivity. The first lemma proves the following: given CLR states \( s_i \) and \( s_j \) that are both to be merged into the same LALR state \( s'_k \), if both \( s_i \) and \( s_j \) are associated only with shift actions as in Figure 3, then \( \mathbf{L}(s_i) = \mathbf{L}(s_j) = \mathbf{L}(s'_k) \) and the number of free parameters associated with \( s'_k \) is exactly one half of that.
Figure 4: Merging CLR states associated with reduce actions

associated with \( s_i \) and \( s_j \). This means that merging \( s_i \) and \( s_j \) into \( s_k \) does not lose any information of the grammatical constraints on the possible next input symbols for \( s_i \) and \( s_j \), although reducing the free parameters makes the probabilistic distributions less context-sensitive.

This seems to suggest that an LALR-based model is a good approximation for a CLR-based model. However, we notice that this is not necessarily the case when we consider cases covered by the second lemma above. In the case illustrated in Figure 4, for example, if \( s_i \) and \( s_j \) are of class \( S_s \), merging them does not reduce the free parameters associated with them. However, we lose the information of the grammatical constraints such that \( s_i \) does not allow any action for input symbol \( v \), and similarly \( s_i \) does not allow any action for \( u \). This means that merging \( s_i \) and \( s_j \) in Figure 4 degrades the model’s context-sensitivity without properly reducing its complexity.

To see this, let us consider an example, which was originally presented by Aho et al. [1]. The grammar is \( G_2 \) as follows:

**Grammar G2**

1. \( S \rightarrow C \ C \)
2. \( C \rightarrow c \ C \)
3. \( C \rightarrow d \)

The CLR and LALR tables for this grammar, indicating an example distribution values, are shown in Table 4. In Table 4, states 3 and 6 in the original CLR table are merged into state 36 in the LALR table. Likewise, 4 and 7 are merged into 47, and 8 and 9 merged into 89. These merging operations influence the overall probabilistic model in the following respects:
Table 4: CLR and LALR tables for Grammar G2

<table>
<thead>
<tr>
<th>state</th>
<th>action</th>
<th>goto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>0</td>
<td>sh3</td>
<td>sh4</td>
</tr>
<tr>
<td>1</td>
<td>.20</td>
<td>.80</td>
</tr>
<tr>
<td>2</td>
<td>sh6</td>
<td>sh7</td>
</tr>
<tr>
<td>3</td>
<td>.70</td>
<td>.30</td>
</tr>
<tr>
<td>4</td>
<td>sh3</td>
<td>sh4</td>
</tr>
<tr>
<td>5</td>
<td>.80</td>
<td>.20</td>
</tr>
<tr>
<td>6</td>
<td>re3</td>
<td>re3</td>
</tr>
<tr>
<td>7</td>
<td>.50</td>
<td>.50</td>
</tr>
<tr>
<td>8</td>
<td>sh6</td>
<td>sh7</td>
</tr>
<tr>
<td>9</td>
<td>.70</td>
<td>.30</td>
</tr>
<tr>
<td>10</td>
<td>re3</td>
<td>re3</td>
</tr>
<tr>
<td>11</td>
<td>.50</td>
<td>.50</td>
</tr>
<tr>
<td>12</td>
<td>sh6</td>
<td>sh7</td>
</tr>
<tr>
<td>13</td>
<td>.70</td>
<td>.30</td>
</tr>
<tr>
<td>14</td>
<td>re3</td>
<td>re3</td>
</tr>
<tr>
<td>15</td>
<td>.50</td>
<td>.50</td>
</tr>
<tr>
<td>16</td>
<td>sh6</td>
<td>sh7</td>
</tr>
<tr>
<td>17</td>
<td>.70</td>
<td>.30</td>
</tr>
<tr>
<td>18</td>
<td>re3</td>
<td>re3</td>
</tr>
<tr>
<td>19</td>
<td>.50</td>
<td>.50</td>
</tr>
<tr>
<td>20</td>
<td>sh6</td>
<td>sh7</td>
</tr>
<tr>
<td>21</td>
<td>.70</td>
<td>.30</td>
</tr>
<tr>
<td>22</td>
<td>re3</td>
<td>re3</td>
</tr>
<tr>
<td>23</td>
<td>.50</td>
<td>.50</td>
</tr>
</tbody>
</table>
• Merging states 8 and 9 into 89 does not make any change in the overall model since they are of class $S_7$, and thus we normalize the probabilities for each next input symbol separately.

• Merging states 3 and 6, which are of class $S_8$, into 36 can be considered as an approximation to the distribution over the next input symbols in those states, since both states are associated with exactly the same sets of actions.

• Merging states 4 and 7, which are of class $S_9$, into 47 may be problematic, since the information that, in state 4, the next input symbol can be only c or d but not $\$ gets lost through this merging operation, despite the number of free parameters not being reduced.

As we mentioned at the beginning of this section, whether it is better to use CLR or LALR is a highly empirical issue. Although replacing CLR with LALR may reduce the model’s context-sensitivity, our preliminary experiments so far have shown that it does not diminish the parsing performance significantly [12].