Context-free Approximation of LTAG towards CFG Filtering

Kenta Oouchida[†] **Naoki Yoshinaga**[†] **Jun'ichi Tsujii**[†*]
[†]University of Tokyo [*] CREST, JST (Japan Science and Technology Agency)
{oouchida, yoshinag, tsujii}@is.s.u-tokyo.ac.jp

Abstract

We present a method to approximate a LTAG grammar by a CFG. A key process in the approximation method is finite enumeration of partial parse results that can be generated during parsing. We applied our method to the XTAG English grammar and LTAG grammars which are extracted from the Penn Treebank, and investigated characteristics of the obtained CFGs. We perform CFG filtering for LTAG by the obtained CFG. In the experiments, we describe that the obtained CFG is useful for CFG filtering for LTAG parser.

1 Introduction

Recently, lexicalized grammars such as Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988) and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) have attracted much attention in practical application context (Deep Thought Project, 2003; Kototoi Project, 2001; Kay et al., 1994; Carroll et al., 1998). However, inefficiency of parsing with those grammars have prevented us from adopting them for practical usage. Especially in the LTAG framework, although many studies proposed parsers that are theoretically efficient (Vijay-Shanker and Joshi, 1985; Schabes and Joshi, 1988; van Noord, 1994; Nederhof, 1998), we do not attain any practical LTAG parser that runs efficiently with large-scale hand-crafted grammars such as the XTAG English grammar (XTAG Research Group, 2001).

Yoshinaga et al. (Yoshinaga et al., 2003) demonstrated that a drastic speed-up of LTAG parsing can be achieved when a LTAG grammar is compiled into a HPSG (Yoshinaga and Miyao, 2002) and a CFG filtering technique for HPSG-Style grammar (Kiefer and Krieger, 2000; Torisawa et al., 2000) is applied to the obtained HPSG. In experiments with the XTAG English grammar, they found that an HPSG parser with CFG filtering (Torisawa et al., 2000) outperformed a theoretically efficient LTAG parser (Sarkar, 2000) in terms of empirical time complexity. Although their approach does not guarantee the theoretical bound of parsing complexity, $O(n^6)$ for a sentence of length n, the empirical results of their CFG filtering are still satisfactory.

In this paper, we propose a novel context-free approximation method for LTAG by reinterpreting the method by Yoshinaga et al. in the context of LTAG parsing. A fundamental idea is to enumerate partial parse results that can be generated during parsing. We assign CFG nonterminal labels to the partial parse results, and then regard their possible combinations as CFG rules.

In order to investigate the characteristics of CFGs produced by our method, we applied our method to two kinds of LTAG grammars. One is the XTAG English grammar, which is a large-scale hand-crafted LTAG, and the other is LTAG grammars extracted from Penn Treebank Wall Street Journal by the grammar extraction method described in (Miyao et al., 2003). Then, we compare parsing speed of a CKY parser using the obtained CFG with parsing speed of an existing LTAG parser.

The remainder of the paper is organized as follows. Section 2 introduces background of our research. Section 3 describes our approximation method. Section 4 reports experimental results with the two kinds of LTAG grammars.

2 Background

2.1 Lexicalized Tree-Adjoining Grammar (LTAG)

An LTAG consists of a set of tree structures, which are assigned to words, called *elementary trees*. A parse tree is derived by combining elementary trees using two grammar rules called *substitution* and *adjunction*. Figure 1 shows elementary trees for *"I"*, *"run"* and *"can"*, and depicts how they are combined by substitution and adjunction.

Substitution replaces a leaf node of an elementary tree by another elementary tree whose root node has the same label as the leaf node. In Figure 1, the leaf node labeled

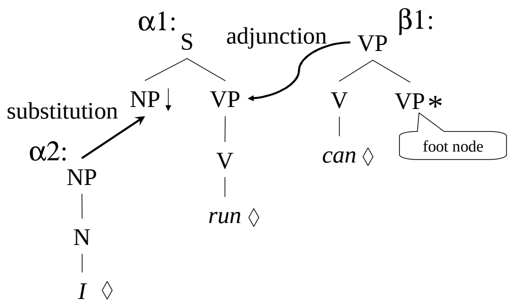

Figure 1: LTAG: elementary trees, substitution and adjunction

"NP" of α_1 is replaced by α_2, which has a root node labeled "NP."

Adjunction replaces an internal node of an elementary tree by another elementary tree whose root node and one leaf node called a *foot node* have the same label as the internal node. In Figure 1, the internal node labeled "VP" of α_1 is replaced by β_2, which has a root node and a foot node labeled "VP."

2.2 CFG filtering

CFG filtering (Harbusch, 1990; Maxwell III and Kaplan, 1993; Torisawa and Tsujii, 1996) is a parsing scheme that filters out impossible parse trees using a CFG extracted from a given grammar prior to parsing. In CFG filtering, we first perform an off-line extraction of a CFG from a given grammar, (*Context-free (CF) approximation*). By using the obtained CFG we can compute efficiently the necessary condition for parse trees the original grammar could generate. Parsing using the obtained CFG as a filter comprises two phases (Figure 2). In the first phase, we parse a sentence by the obtained CFG. In this phase, the necessary condition represented by the CFG acts as a filter of parse trees. In the second phase, using the whole constraints in the original grammar, we examine the generated parse trees, and eliminate overgenerated parse trees.

The performance of parsers with CFG filtering depends on the degree of the CF approximation (Yoshinaga et al., 2003). If CF approximation is good, the number of overgenerated parse trees is small. Thus, the key to achieve efficiency in LTAG parsing is to maintain grammatical restrictions in CFG as efficiently as possible. The more of the grammatical constraints in the given grammar the obtained CFG captures, the more effectively we can restrict the search space.

There are existing CFG filtering techniques for LTAG (Harbusch, 1990; Poller and Becker, 1998). These techniques extract CFG rules by simply dividing elementary trees into branching structures as shown in Figure 3. Since the obtained CFG can capture only local constraints

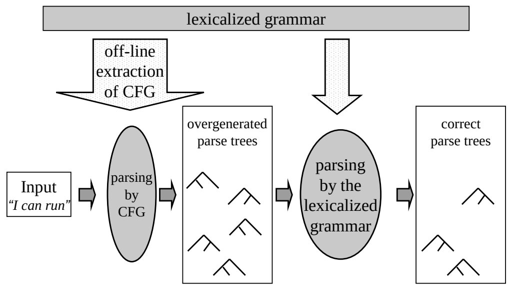
lexicalized grammar
off-line extraction of CFG
overgenerated parse trees
parsing by the lexicalized grammar
correct parse trees
Input
"I can run"
parsing by CFG

Figure 2: CFG filtering

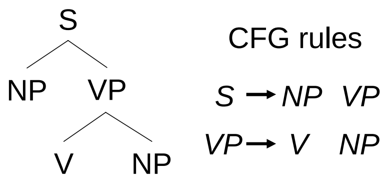

Figure 3: The existing CF approximation for LTAG

given in the elementary trees, we must examine many global constraints in the second phase.

CFG filtering techniques have also been developed for HPSG (Torisawa and Tsujii, 1996; Torisawa et al., 2000; Kiefer and Krieger, 2000). CFG rules are extracted by applying grammar rules to lexical entries and by enumerating partial parse results (*sign*) that can be generated during parsing (in Figure 4). The obtained CFG can capture global constraints given in the lexical entries, because the generated partial parse results preserve the whole constraints given in the lexical entries.

As Yoshinaga et al. demonstrated using HPSG-style grammar converted from LTAG, finite enumeration of partial parse results produces a better CFG filter than the existing CF approximation for LTAG because of its ability to capture the global constraints. In the paper, we re-interpret CF approximation of HPSG by Yoshinaga's method (Yoshinaga et al., 2003).

3 CF Approximation algorithm for LTAG

In this section, we describe an algorithm of our CF approximation of LTAG. In the following, we first describe an approximation of LTAG which consists only of single-anchored elementary trees. We then describe an approximation of general LTAG which includes multi-anchored elementary trees.

In Section 3.1, we introduce a basic idea in our method. In Section 3.2, we explain our method in detail. In Section 3.3, we explain the way of applying our method to LTAG which comprises multi-anchored elementary trees.

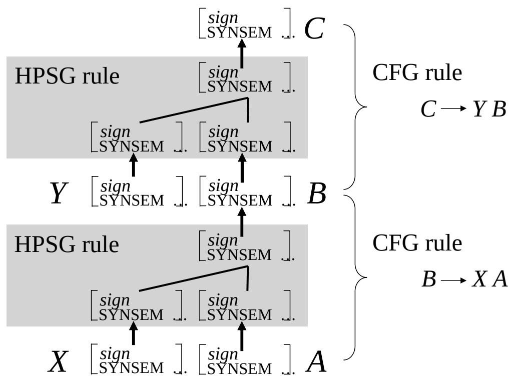

Figure 4: The existing CF approximation for HPSG

3.1 Basic Idea

The fundamental idea of our approximation method is to enumerate partial parse results that can be generated during parsing. We obtain a CFG by assigning CFG non-terminal labels to the partial parse results, and regarding bottom-up derivation relationships between the partial parse results as CFG rules.

By recursively applying substitution and adjunction to elementary trees, we enumerate partial parse results derivable by LTAG. We adopt one of the existing mode, *head-corner traversal* (van Noord, 1994) (Figure 5), to recursively apply grammar rules.

In the first step of head-corner traversal, an elementary tree is taken as input and a directed path from an anchor node called *head-corner* to the root node is defined in a certain manner. The path traverses along all the nodes in the elementary tree. Then, grammar rules are incrementally applied to each node along the path.

We assign a non-terminal label of CFG to a subtree. A labeled subtree must include all information for enumeration. We determine this subtree as follows. A tree is divided into two parts, at the node to which we are applying a grammar rule (Figure 6). The "lower" part of the tree is a subtree below the node to which we are applying a grammar rule. The "upper" part consists of the nodes to which we will apply grammar rules in the rest of enumeration. We need only the "upper" part of the tree that includes all information necessary in the rest of the enumeration process. In this paper, we call the node to which we are applying a grammar rule a **processing node**, and we call the upper part of a tree an **active partial tree**. CFG non-terminal labels are assigned to each active partial tree.

By assigning CFG non-terminals to generated active partial trees, we obtain CFG rules as bottom-up derivation relationships between them. In Figure 7, the following CFG rules are obtained: $G \rightarrow A\,F$, $F \rightarrow E$, $E \rightarrow D\,E$, $D \rightarrow B$ and $E \rightarrow C$.

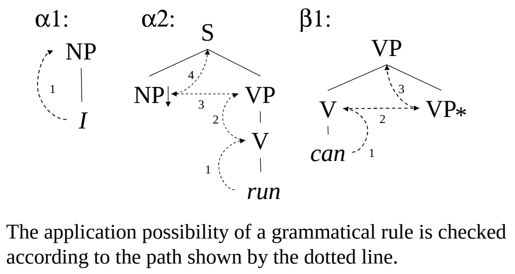
α1:
α2:
β1:
NP
I
S
NP↓
VP
V
run
VP
V
VP*
can
1
2
3
4
The application possibility of a grammatical rule is checked according to the path shown by the dotted line.

Figure 5: head-corner traversal

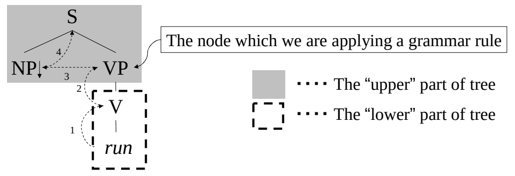

Figure 6: Division of a tree into two parts

3.2 Algorithm

Table 1 shows pseudo-code of our approximation algorithm. The algorithm takes LTAG G as input and outputs a CFG G'.

We start by explaining the top-level function `''extract_cfg_from_ltag.''` The function iteratively picks up two active partial parse trees from the set of active partial trees generated so far, and applies possible grammar rules. Whenever a new active partial parse tree is generated, we assign a new CFG non-terminal label and add it to the set. In case that new partial parse results have not been added during one iteration, we exit with G', which is the resulting CFG.

The function `''apply_rules''` applies the grammar rules to two active partial trees, and change the processing node to the next node. We apply unary rule in line 5, substitution in line 8, and adjunction in line 12.

Let us consider the extraction of CFG from LTAG defined in Figure 1. Figure 7 shows the extraction process. The initial active partial trees A, B and C originate from α_1, α_2, and β_1. In the first iteration in the function `''extract_cfg_from_ltag''`, two partial active trees, D and E, and two CFG rules, $D \rightarrow B$ and $E \rightarrow C$ are extracted. In the second iteration, one partial active tree, F and two CFG rules, $E \rightarrow D\,E$ and $F \rightarrow E$ are extracted. In the third iteration, one partial active tree, G, and one CFG rule, $G \rightarrow A\,F$ are extracted.

When substitution is applied to an active partial tree, the size of the parent's active partial tree is smaller than child's active partial trees. Thus, the number of generated active partial trees is finite, and the number of non-terminal labels in the obtained CFG is finite as well. In other words, if the CFG rules comprise only substitutions,

Table 1: The pseudo code of our algorithm

```
INPUT: $G$ /* LTAG */
OUTPUT: $G'$ /* CFG */

$T_n$:  /* a set of all active partial trees for $n$th iteration generated so far */
$NT_n$:  /* a set of new active partial trees for $n$th iteration */

Initialize:
$T_n$ := $\phi$
$NT_1$ := etree($G$)
$G'$ := $\phi$
$n$ := 1

1:  procedure extract_cfg_from_ltag($G$)
2:  begin
3:    while ( $NT_n \neq \phi$ )
4:      $T_n$ := $T_{n-1} \cup NT_n$
5:      foreach $nt_n$ in ( $NT_n$ )
6:        foreach $t_n$ in ( $T_n$ )
7:          $NT$ := apply_rules( $t_n$, $nt_n$ )
8:          $NT_{n+1}$ = $NT \cup NT_{n+1}$
9:        end foreach
10:     end foreach
11:     $n{+}{+}$
12:   end while
13:   return $G'$
14: end extract_cfg_from_ltag

1:  procedure apply_rules($t_1$, $t_2$)
2:  begin
3:    $NT$ := $\phi$
4:    if ( sibling( c_node( $t_1$ ) == nil ) ) /* if we cannot apply grammar rules */
5:      $NT$ = unary( $t_1$ )
6:      $G'$ = make_rule( $t_1$, $NT$ ) $\cup$ $G'$
7:    else if ( sibling( c_node( $t_1$ ) ) == ''subst'' ) /* if we can apply substitution */
8:      $NT$ = substitute( $t_2$, $t_1$ )
9:      $G'$ = make_rule( $t_2$, $t_1$ ) $\cup$ $G'$
10:   else if ( sibling( c_node( $t_1$ ) ) == ''foot'' ) /* if we can apply adjunction */
11:     if ( depth_foot( $t_1$ ) == 1 || count_adjoing( $t_1$ ) <= LIMIT )
12:       $NT$ = adjoin( $t_1$, $t_2$ )
13:       $G'$ = make_rule( $t_2$, $t_1$ ) $\cup$ $G'$
14:     if ( depth_foot( $t_1$ ) >= 2 && count_adjoing( $t_1$ ) > LIMIT )
15:       $NT$ = $*$
16:       $G'$ = make_rule( $t_2$, $t_1$ ) $\cup$ $G'$
17:     end if
18:   end if
19:   return $NT$
20: end apply_rules
```

etree:	To return the elementary trees with head-corner paths.
c_node:	To return the processing node of the argument.
unary:	To return an active partial tree with the node which we will apply the grammar rules after.
make_rule:	To return the CFG rule of arguments
substitute:	To apply the rule and to return new active partial tree, if we can apply the grammar rule of substitution to the arguments,
adjoin:	To apply the rule and to return new active partial tree, if we can apply the grammar rule of adjunction to the arguments.

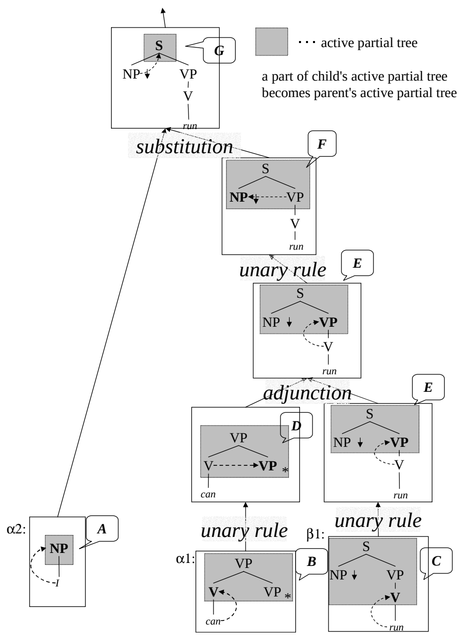

Figure 7: Approximation of LTAG to CFG

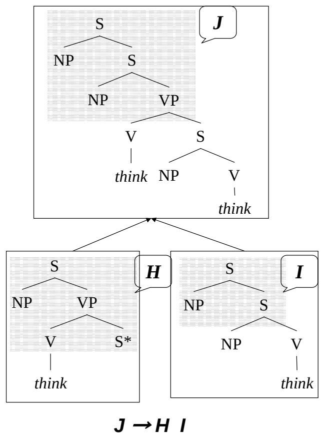

Figure 8: adjunction:the combination of two active partical trees

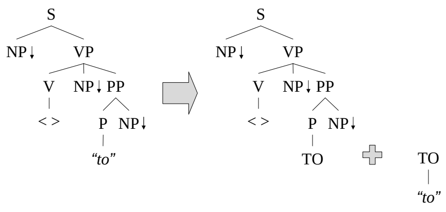

Figure 9: compiling XTAG English grammar

LTAG can be converted to CFG in finite size.

We must be careful about the depth of a foot node of an auxiliary tree when adjunction is applied to an active partial tree. If the depth of a foot node is one, active partial tree becomes the same as one of the two active partial trees to which adjunction are applied. If the depth of a foot node is two or more, parent's active partial tree takes a form of a combination of two active partial trees (Figure 8). This means that the number of active partial trees increases infinitely, if there are some auxiliary trees with a foot node at depth two or more.

In order to prevent the infinite increase of active partial trees, we count the number of the applications of adjunction which generates new active partial trees, and assign a special non-terminal "$*$" to active partial trees when the number of the applications reach a certain threshold. We then add CFG rules, $* \rightarrow X *$ and $* \rightarrow * X$ for all non-terminal labels "X" in order to guarantee that the resulted CFG can generate all parse trees that LTAG can generate. By using these rules, resulted CFG always generate parse trees which are derivable by the given grammar. Thus the obtained CFG can be used as a filter.

3.3 Extention to LTAG including multi-anchored trees

Our method can be applied to LTAG with elementary trees which contain only one anchor. It is the reason that the path from a head node to a root node becomes settled uniquely. The above approximation algorithm a handle general LTAG with multi-anchored trees by simply converting those trees into single anchored trees. When a grammar includes multi-anchored elementary trees, we simply replace an anchor node of them by a node to which can be applied a grammar rule of substitution (e.g. *TO* in Figure 9), and add a new elementary tree (e.g. *TO-"to"* in Figure 9).

4 Experiments

In order to observe the characteristics of CFG obtained by our method, we performed three experiments. In Section 4.1, we apply our method to the XTAG English grammar. In Section 4.2, we apply our method to LTAG grammars of various size extracted from a corpus, and investigate the relation between the size of LTAG grammars and the specification of the obtained CFG. In Section 4.3, we examine the characteristics of the obtained CFG in terms of the parsing speed, and compare the parsing speed of a CKY parser using the obtained CFG with the parsing speed of an existing LTAG parser.

4.1 Experiment on threshold value of adjunction

We applied our algorithm to the XTAG English grammar. In Table 2, we show the obtained CFG approximation of the XTAG English grammar. In this experiment, we varied threshold value of times of adjunction, which generates a new active partial tree, 0.

Table 2: approximating the XTAG English grammar by CFG

# of elementary trees of XTAG	924
# of terminal labels	924
# of non-terminal labels	2,779
# of rules	31,503

Even if the threshold is small, an obtained CFG is useful for parsing, because we hardly perform an adjunction using an auxiliary tree with a foot node with the depth of two or more in LTAG parsing. The maximum number of the possible rules is $2,779 \times (924+2,779) + 2,779 \times (924+2,779) \times (924+2,779) = 370338,116,519,448$. Our method produced 31,503 rules (about 0.00008% of all possible ones). Thus our method is efficient with respect to avoiding meaningless increase of the number of CFG rules. The small percentages means that obtained CFG is excellent as a filter.

4.2 Experiment on Various Size of LTAG

We extracted LTAG grammars from Section 02-06, 02-11, 02-16, and 02-21 of Penn Treebank Wall Street Journal (Table 3), and applied our method to the LTAG grammars.

We investigated the relation between the size of terminal labels, the size of non-terminal labels, and the size of CFG rules (Table 4). The increase of the number of non-terminal labels is slower than the increase of the number of terminal labels. Thus, our method is applicable for a larger LTAG than the LTAG which we used for this experiment.

4.3 Comparing our method to an existing LTAG parser

We investigated parsing performance of the obtained CFG.

We selected sentences which consists of 15 or less words from Section 23 of Penn Treebank Wall Street Journal, and experimented on parsing the sentences by two ways. One way is an existing LTAG parser with the XTAG English grammar. The other way is a CKY parser with the obtained CFG from the XTAG English grammar with threshold 1. The reason why we used the sentences which consists of 15 or less words is that existing LTAG parser is too slow to parse longer sentences.

Figure 10 and Figure 11 show the performances of an existing LTAG parser (Sarkar, 2000) and a CKY parser with the obtained CFG respectively. The CKY parsing is fast enough to employ a CFG filter. For longer sentences, it is expectable that our algorithm is effective.

5 Concluding Remarks and Future direction

In this paper, we showed an approximating method of LTAG by CFG. A specification of a CFG obtained from the XTAG English grammar shows that our method is efficient in the number of CFG rules. In addition, we compared parsing performance between the existing LTAG parser and the CKY parser with CFG which is obtained from automatically extracted LTAG grammars. The comparison showed that the obtained CFG is useful for CFG filtering for a LTAG parser. We will implement CFG filtering for a LTAG parser, and verify the efficiency of CFG filtering with our approximated CFG.

References

John Carroll, Nicolas Nicolov, Olga Shaumyan, Martine Smets, and David Weir. 1998. The LEXSYS project. In *Proc. of the fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 29–33.

K. Harbusch. 1990. An efficient parsing algorithm for Tree Adjoining Grammars. In *Proc. of the 28th ACL*, pages 284–291.

M. Kay, J. Gawron, and P. Norvig. 1994. *Verbmobil: A Translation System for Face-to-Face Dialog*. CSLI Publications.

B. Kiefer and H.-U. Krieger. 2000. A context-free approximation of Head-Driven Phrase Structure Grammar. In *Proc. of the sixth IWPT*, pages 135–146.

J. T. Maxwell III and R. M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.

Deep Thought Project. 2003. http://www.project-deepthought.net/.

Kototoi Project. 2001. http://www-tsujii.is.s.u-tokyo.ac.jp/kototoi/.

XTAG Research Group. 2001. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Y. Miyao, T. Ninomiya, and J. Tsujii. 2003. Lexicalized grammar acquisition. In *Proc. of the 10th EACL companion volume*, pages 127–130.

M.-J. Nederhof. 1998. An alternative LR algorithm for TAGs. In *Proc. of COLING–ACL*, pages 946–952.

C. Pollard and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. CSLI Publications.

P. Poller and T. Becker. 1998. Two-step TAG parsing revisited. In *Proc. of TAG+4*, pages 143–146.

A. Sarkar. 2000. Practical experiments in parsing using Tree Adjoining Grammars. In *Proc. of TAG+5*, pages 193–198.

Table 3: LTAG grammars extracted from Penn Treebank Wall Street Journal

corpus	02-06	02-11	02-16	02-21
# of words	2285	3662	5074	6056
# of non-terminals	190	207	211	216
# of elementary tree	1061	1484	1854	2111

Table 4: Obtained CFG from LTAG grammars

corpus	02-06	02-11	02-16	02-21
# of terminal labels	1061	1484	1854	2111
# of non-terminal labels	1768	2780	3458	3911
# of rules	45173	92908	129542	154407

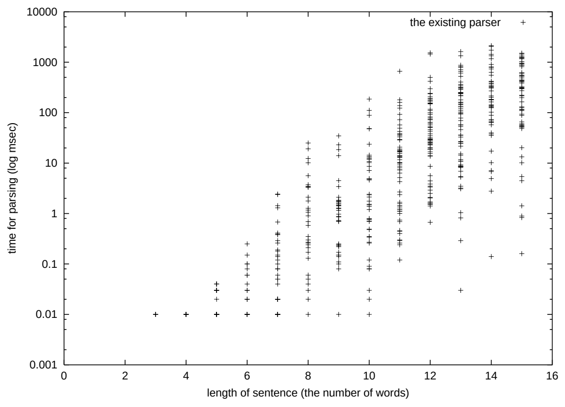

Figure 10: Speed test for an existing LTAG parser

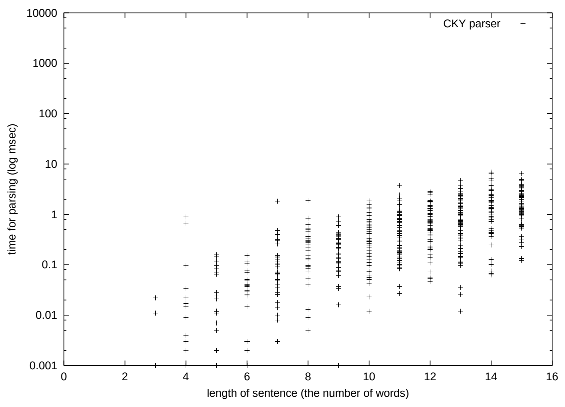

Figure 11: Speed test for a CKY parser with the obtained CFG

Yves Schabes and Aravind Joshi. 1988. An earley-type parsing algorithm for Tree Adjoining Grammars. In *Proc. of the 26th Annual Meeting of the Association for Computational Linguistics (ACL 1988)*, pages 258–269.

Y. Schabes, A. Abeillé, and A. K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In *Proc. of the 12th COLING*, pages 578–583.

K. Torisawa and J. Tsujii. 1996. Computing phrasal-signs in HPSG prior to parsing. In *Proc. of the 16th COLING*, pages 949–955.

K. Torisawa, K. Nishida, Y. Miyao, and J. Tsujii. 2000. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6(1):63–80.

G. van Noord. 1994. Head corner parsing for TAG. *Computational Intelligence*, 10(4):525–534.

K. Vijay-Shanker and A. K. Joshi. 1985. Some computational properties of Tree Adjoining Grammars. In *Proc. of the 23rd ACL*, pages 82–93.

N. Yoshinaga and Y. Miyao. 2002. Grammar conversion from LTAG to HPSG. *The European Student Journal on Language and Speech*. available at http://hltheses.elsnet.org/eventpapers/essliproceed.htm.

N. Yoshinaga, T. Kentaro, and J. Tsujii. 2003. Comparison between CFG filtering techniques for LTAG and HPSG. In *Proc. of the 41st ACL companion volume*, pages 185–188.