Comparative Parser Performance Analysis across Grammar Frameworks through Automatic Tree Conversion using Synchronous Grammars

Takuya Matsuzaki [1] **Jun'ichi Tsujii** [1,2,3]
1. Department of Computer Science, University of Tokyo, Japan
2. School of Computer Science, University of Manchester, UK
3. National Center for Text Mining, UK
{matuzaki, tsujii}@is.s.u-tokyo.ac.jp

Abstract

This paper presents a methodology for the comparative performance analysis of the parsers developed for different grammar frameworks. For such a comparison, we need a common representation format of the parsing results since the representation of the parsing results depends on the grammar frameworks; hence they are not directly comparable to each other. We first convert the parsing result to a shallow CFG analysis by using an automatic tree converter based on synchronous grammars. The use of such a shallow representation as a common format has the advantage of reduced noise introduced by the conversion in comparison with the noise produced by the conversion to deeper representations. We compared an HPSG parser with several CFG parsers in our experiment and found that meaningful differences among the parsers' performance can still be observed by such a shallow representation.

1 Introduction

Recently, there have been advancement made in the parsing techniques for large-scale lexicalized grammars (Clark and Curran, 2004; Ninomiya et al., 2005; Ninomiya et al., 2007), and it have presumably been accelerated by the development of the semi-automatic acquisition techniques of large-scale lexicalized grammars from parsed corpora (Hockenmaier and Steedman, 2007; Miyao et al., 2005). In many of the studies on lexicalized grammar parsing, the accuracy of the parsing results is evaluated in terms of the accuracy of the semantic representations output by the parsers. Since the formalisms for the semantic representation are different across the grammar frameworks, it has been difficult to directly compare the performance of the parsers developed for different grammar frameworks.

Several researchers in the field of lexicalized grammar parsing have recently started to seek a common representation of parsing results across different grammar frameworks (Clark and Curran, 2007; Miyao et al., 2007). For example, Clark and Curran (2007) developed a set of mapping rules from the output of a Combinatorial Categorial grammar parser to the Grammatical Relations (GR) (Carroll et al., 1998). They found that the manual development of such mapping rules is not a trivial task; their mapping rules covered only 85% of the GRs in a GR-annotated corpus; i.e., 15% of the GRs in the corpus could not be covered by the mapping from the gold-standard CCG analyses of those sentences.

We propose another method for the cross-framework performance analysis of the parsers wherein the output of parsers are first converted to a CFG tree. Specifically, we use CFG trees of the style used in the Penn Treebank (PTB) (Marcus et al., 1994), in which the non-terminal labels are simple phrasal categories (i.e., we do not use function-tags, empty nodes, and co-indexing). We hereafter name such CFG trees, 'PTB-CFG trees.' We use an automatic tree converter based on a stochastic synchronous grammar in order to make the PTB-CFG trees from the analyses based on a lexicalized grammar.

In such a shallow representation, some infor-

mation given by the lexicalized parsers is lost. For instance, long-distance dependency and control/raising distinction cannot be directly represented in the PTB-CFG tree. From the viewpoint of NLP-application developer, the parser evaluation based on such a shallow representation may be not very informative because performance metrics based on the shallow representation, e.g., labeled bracketing accuracy, do not serve as a direct indicator of the usefulness of the parser in their applications. Nevertheless, we consider the parser performance analysis based on the shallow representation is still very useful from the viewpoint of *parser developers* because the accuracy of the structure of the CFG-trees is, though not an ideal one, a good indicator of the parsers' structural disambiguation performance.

In addition, there are at least two advantages in using the CFG-trees as the common representation for the evaluation. The first advantage is that the conversion from the parser's output to the CFG-trees can be achieved with much higher accuracy than to deeper representations like GRs; we obtained a conversion accuracy of around 98% in our experiments using an HPSG grammar. The accuracy of the conversion is critical in the quantitative comparison of parsers that have similar performances because the difference in the parsers' ability would soon be masked by the errors introduced in the conversion process. The second advantage is that we can compare the converted output directly against the outputs of the well-studied CFG-parsers derived from PTB.

In the experiments, we applied the conversion to an HPSG parser, and compared the results against several CFG parsers. We found that the parsing accuracy of the HPSG parser is a few points lower than state-of-the-art CFG parsers in terms of the labeled bracketing accuracy. By further investigating the parsing results, we have identified a portion of the reason for the discrepancy, which comes from the difference in the architecture of the parsers.

2 Background

In this section, we first give a brief overview of the semi-automatic acquisition framework of lexicalized grammars. Although our methodology is also applicable to manually developed grammars, in this paper, we concentrate on the evaluation of the parsers developed for lexicalized grammars derived from a CFG treebank. Next, we introduce a specific instance of the treebank-derived lexicalized grammars used in our experiment: the Enju English HPSG grammar. Using the Enju grammar as a concrete example, we present the motivations for our tree conversion method based on a stochastic synchronous grammar. We also provide a summary of the basic concepts and terminologies of the stochastic synchronous grammar.

2.1 Semi-automatic Acquisition of Lexicalized Grammars

A lexicalized grammar generally has two components: a small set of grammar rules and a large set of lexical items. The grammar rules represent generic grammatical constraints while the lexical items represent word-specific characteristics. An analysis of a sentence is created by iteratively combining lexical items assigned to to the words in the sentence by applying the grammar rules.

Several researchers have suggested to extract the lexicon; i.e., the set of lexical items, from a treebank such as PTB. Most of the lexicon acquisition methods proceed as follows:

1. Fix the the grammar rules and the basic design of the lexical items.

2. Re-analyse the sentences in terms of the target grammar framework, exploiting the analysis given in the source treebank. A re-analysis is generally represented as a derivation of the sentence; i.e., a history of rule applications.

3. Find a lexical item for each word in the sentences so that it matches the re-analysis of the sentence, and extract it.

We used the pairs of the original trees and the re-analyses of the same sentence as a parallel treebank, from which we extract a synchronous grammar.

2.2 The Enju HPSG Grammar

We used the Enju English HPSG grammar (Miyao et al., 2005) [1] in the experiments. The design of the grammar basically follows the definition in the text by Pollard and Sag (1994). A program called Mayz is distributed with the grammar, which was

[1] Version 2.2., publicly available from http://www-tsujii.is.s.u-tokyo.ac.jp/enju

used to make the HPSG treebank (i.e., a set of re-analyses based on the HPSG grammar) from PTB; the lexicon was extracted from the HPSG treebank. We reproduced the HPSG treebank using the program.

An analysis of a sentence in the HPSG formalism is represented by a phrasal tree, in which each node is assigned a data structure called typed feature structure (TFS). The TFS represents syntactic/semantic structures of the corresponding phrase. To convert an HPSG analysis to a corresponding PTB-CFG trees, we first map the TFSs to atomic symbols like PP, NP, NX, etc. (33 symbols in total). We hereafter name such HPSG trees after the TFS-to-symbol mapping, 'simplified HPSG trees.' Similarly to the PTB-CFG trees, the simplified HPSG trees do not include empty categories, co-indexing, and function-tags. However, we cannot attain a PTB-CFG tree by simply mapping those atomic symbols to the corresponding PTB non-terminal symbols, because the analyses by the PTB-CFG and the HPSG yield different tree structures for the same sentence.

The conversion of the tree structure from HPSG trees to PTB-CFG trees can be regarded as the inverse-mapping of the transformation from PTB trees to HPSG trees implemented in the Mayz program. A most notable transformation is the binarization of the PTB trees; all the branches in the HPSG treebank are unary or binary. The binarization scheme used in Mayz is similar to the head-centered binarization, which is often used for the extraction of 'Markovised' PCFGs from the treebank. Mayz identifies the head daughters by using a modified version of Collins' head finding rules (Collins, 1999). It is also notable that the PTB-to-HPSG transformation by Mayz often makes a bracketing in the HPSG analyses that crosses with the original bracketing in the PTB. Such a transformation is used, for instance, to change the attachment level of an article to a noun phrase with a post-modifier (Figure 1).

The tree transformation by Mayz is achieved by sequentially applying many tree transformation rules to an input PTB tree. Although each of the rules operates on a relatively small region of the tree, the net result can be a very complex transformation. It is thus very difficult, if not impossible, to invert the transformation programmatically.

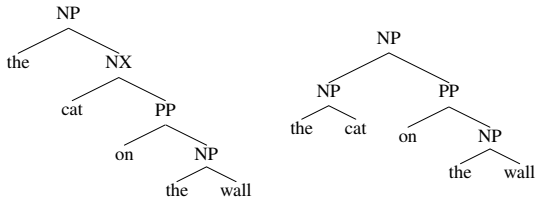

Figure 1: Different attachment level of the articles: HPSG analysis (left) and PTB-CFG analysis (right).

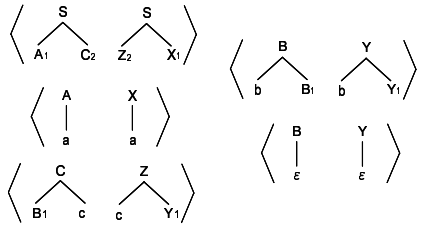

Figure 2: An example of synchronous CFG

2.3 Stochastic Synchronous Tree-Substitution Grammar for Tree Conversion

For the purpose of the inverted transformation of simplified HPSG trees to PTB-CFG trees, we use a statistical approach based on the stochastic synchronous grammars. Stochastic synchronous grammars are a family of probabilistic models that generate a pair of trees by recursively applying *synchronous productions*, starting with a pair of initial symbols. See e.g., Eisner (2003) for a more formal definition. Figure 2 shows an example of synchronous CFG, which generates the pairs of strings of the form $(\mathbf{a}\mathbf{b}^m\mathbf{c}, \mathbf{c}\mathbf{b}^m\mathbf{a})$. Each non-terminal symbol on the yields of the synchronous production is linked to a non-terminal symbol on the other rule's yield. In the figure, the links are represented by subscripts. A linked pair of the non-terminal symbols is simultaneously expanded by another synchronous production.

The probability of a derivation D of a tree pair $\langle S, T \rangle$ is defined as the product of the probability of the pair of initial symbols (i.e., the root nodes of S and T), and the probabilities of the synchronous productions used in the derivation:

$$P(D) = P\left(\left\langle R^1, R^2 \right\rangle\right) \prod_{\left\langle t_i^1, t_i^2 \right\rangle \in D} P\left(\left\langle t_i^1, t_i^2 \right\rangle\right),$$

where $\langle R^1, R^2 \rangle$ is the pair of the symbols of the root nodes of S and T, and $\langle t_i^1, t_i^2 \rangle$ is a synchronous production.

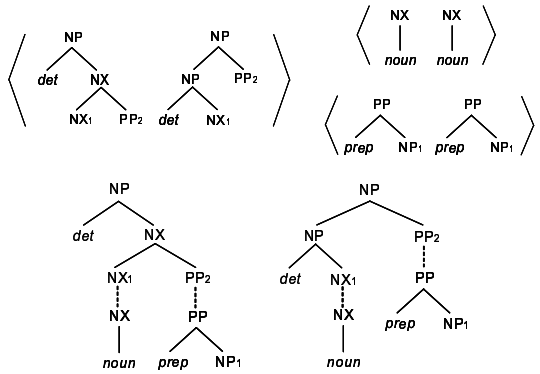

Figure 3: An example of synchronous TSG: synchronous productions (top) and a synchronous derivation (bottom).

Stochastic synchronous grammars have been used in several machine-translation systems to serve as a model of tree-to-tree translation (e.g., (Eisner, 2003; Chiang, 2007)). Our objective of automatic conversion between syntactic analyses is similar to the tree-to-tree machine translation. An important difference is that, for our purpose, the generated tree pair should have the same yields since they are two analyses of the same sentence. We also want the synchronous grammar to be able to generate a pair of trees wherein some constituents in one tree cross with the constituents in the other tree; for example, such a tree-transformation is necessary to change the article-attachment levels.

We show below that, by means of a simple algorithm, we can obtain a synchronous tree-substitution grammar (STSG) that meets the above two requirements. Synchronous productions in STSGs may include a local tree of depth greater than one. Figure 3 shows a tiny STSG that generates two different analyses for PP-modified NPs. Furthermore, a part of the derivation of the two analyses is presented; the dotted-lines indicate applications of the synchronous productions.

3 Tree Transformation Based on a Stochastic Synchronous Grammar

3.1 Extraction of Synchronous Grammars

We created an STSG from a parallel treebank, in which a sentence is assigned with a PTB-CFG tree and a simplified HPSG tree. The synchronous productions were obtained by splitting the tree pairs at several pairs of non-terminal nodes. Specifically, for a tree pair (T_1, T_2) for a sentence $\mathbf{s}$,

1. we select a set of 'common-span node pairs' $\{(N_i^1, N_i^2) | N_i^1 \in T_1, N_i^2 \in T_2, i = 1, \ldots k\}$, which is a set of pairs of non-terminal nodes that dominate the same span of $\mathbf{s}$, and then
2. split (T_1, T_2) at each (N_i^1, N_i^2) for $i = 1, \ldots, k$.

For tree pairs that do not include unary productions, the above procedure uniquely determines a collection of pairs of tree-fragments (t_i^1, t_i^2) by splitting the tree pair at all the common-span node pairs. For tree pairs including unary productions, we chose the split points $\{(N_i^1, N_i^2)\}$ so that they yield a pair of the longest unary chains as a tree-fragment pair when both T_1 and T_2 include one or more unary production dominating a common span. When only T_1 (or T_2) includes one or more unary productions on a common span, we selected the upper-most node of the chain of the unary productions in T_1, and the corresponding node in T_2 as the split points.

A node on the yields of the resulting tree-fragments is either a terminal node or a node in a common-span node pair that is selected as a split node. We make the synchronous productions by placing links between the common-span node pairs on the yields of the tree-fragment pairs. We can show that the grammar obtained as above only generates pairs of identical sentences by noting that the terminal symbols on the yields of two local trees in a synchronous production, if any, are always identical.

By regarding the division of tree-pairs into the tree-fragment pairs as a synchronous derivation, we get the maximum-likelihood estimates of the probabilities of the synchronous productions:

$$P\left(\langle t_1, t_2 \rangle\right) = \frac{\mathrm{Count}(\langle t_1, t_2 \rangle)}{\mathrm{Count}(\langle \mathrm{root\_of}(t_1), \mathrm{root\_of}(t_2) \rangle)},$$

where Count$(\cdot)$ represents the number of the synchronous productions or the pair of their root symbols observed in the derivations.

As is clear from the construction, certain types of conversions can not be handled properly by the STSG. For instance, the apparent conversion rule between the right-branching analysis of the listing: (*rats*, (*cows*, (*tigers*, ... (*and pigs*))...) and left-branching analysis: (...((*rats*, *cows*,) *tigers*,) ... *and pigs*) can not be represented in its full generality by the STSG. We however expect that most of such cases could be handled by combining the STSG-based conversion with a programmatic conversion.

3.2 Tree Conversion Algorithm

We can define a conversion function $f(S)$, which takes a tree S and returns the converted tree T, as:

$$f(S) = \operatorname*{argmax}_T P(T|S) = \operatorname*{argmax}_T P(S,T),$$

where $P(S,T)$ is the marginal probability of the tree pair $\langle S,T\rangle$. The marginal probability is the sum of the probabilities of all the derivations that generate the tree pair $\langle S,T\rangle$. Since the synchronous productions in a STSG may include a local tree of any depth, there are exponentially many derivations that generate the same tree pair. To our knowledge, no polynomial time algorithm is known for the above optimization problem.

We have instead searched the max-probability synchronous derivation of the tree pair of the form $\langle S,\cdot\rangle$ and taken the opponent tree as the conversion result of the input tree S:

$$f(S) = \operatorname*{argmax}_T \operatorname*{argmax}_{D:\text{deriv.of}\langle S,T\rangle} P(D).$$

We used Eisner's decoding algorithm (Eisner, 2003), which is similar to the Viterbi algorithm for HMMs, to obtain the max-probability derivation.

3.3 A Back-off Mechanism for the Conversion

In our preliminary experiment, a certain number of source trees were not covered by the synchronous grammar. For the trees that are not covered by the synchronous grammar, we used a set of "back-off rules," which are synchronous productions consisting of two 1-level local trees:

$$\{\langle X\to\alpha, Y\to\beta\rangle \mid \quad X\in N_1, \alpha\in(N_1\cup\Sigma)^*, \\ Y\in N_2, \beta\in(N_2\cup\Sigma)^*\},$$

where N_1 and N_2 are the sets of non-terminal symbols used in the two treebanks, and Σ is the set of terminal symbols. We assign small scores to the back-off rules as:

$$\text{score}_{\langle X\to\alpha, Y\to\beta\rangle} = \epsilon P(X\to\alpha)\prod P(B_i|A_i),$$

where ϵ is a small constant, $P(X\to\alpha)$ is the maximum-likelihood estimate of the PCFG rule probability of $X\to\alpha$ in the source treebank, and $\prod P(B_i|A_i)$ is the product of 're-labeling probabilities' for the pairs of linked non-terminal symbols in α and β, defined as:

$$P(B|A) = \frac{\text{Count}(\langle A,B\rangle)}{\sum_{B'}\text{Count}(\langle A,B'\rangle)},$$

where Count$(\cdot)$ represents the number of common-span node pairs used as the split points.

When a source tree is not covered by the original synchronous productions, we add the back-off rules to the synchronous productions, and search for the highest-scored derivation. The score of a derivation including the back-off rules is defined as the product of the probabilities of the original synchronous productions, and the scores of the back-off rules. We set the value of ϵ to be sufficiently small so that the highest-scored derivation includes a minimum number of back-off rules.

4 Experiments

4.1 Experiment Setting

We compared the performance of an HPSG parser with several CFG parsers. The HPSG parser is the Enju parser (Ninomiya et al., 2007), which has been developed for parsing with the Enju HPSG grammar. A disambiguation module based on a discriminative maximum-entropy model is used in the Enju parser. We compared the Enju parser with four CFG parsers: Stanford's lexicalized parser (Klein and Manning, 2003), Collins' parser (Collins, 1999), Charniak's parser (Charniak, 2000), and Charniak and Johnson's reranking parser (Charniak and Johnson, 2005). The first three parsers are based on treebank PCFGs derived from PTB. The last parser is a combination of Charniak's parser and a reranking module based on a maximum-entropy model. The Enju parser and Collins' parser require POS-tagged sentences as the input. A POS tagger distributed with the Enju parser was used for the POS-tagging.

We used a standard split of PTB for the training/development/test data: sections 02-21 for the extraction of the synchronous grammar, section 22 for the development, and section 23 for the evaluation of the parsers. Some of the trees in PTB are missing in the HPSG treebank because the Enju grammar does not cover all sentences in PTB. Section 23 of the HPSG treebank, which was used as the gold-standard of the HPSG analyses in the experiments, thus contains fewer sentences (2,278 sentences) than the original PTB section 23 (2,416 sentences). We use a notation, "section 23*," to indicate the portion of PTB section 23 covered by the Enju grammar.

Each parser's output was evaluated in the following three representations:

- PTB-CFG trees: We converted the Enju

parser's output to PTB-CFG trees by using a synchronous grammar. For the CFG parsers, we used their output as is.

- Simplified HPSG trees: We converted the CFG parsers' output to simplified HPSG trees by using another synchronous grammar. The Enju parser's output was mapped to simplified HPSG trees. This was achieved by simply mapping the TFSs assigned to the non-terminal nodes to atomic symbols.

- Unlabeled word-word dependency: We extracted head-modifier dependencies from the PTB-CFG trees by using Collins' head finding rules.

The evaluation in the unlabeled word-word dependency is motivated by the expectation that, by converting PTB-CFG trees to an even simpler representation, we can reduce the effect of the noise introduced in the conversion of the Enju parser's output to the PTB-CFG trees.

4.2 Extraction of Synchronous grammars

The stochastic synchronous grammars used for the tree-to-tree conversion were extracted from a parallel treebank consisting of the PTB-CFG trees in sections 02-21 of PTB and the simplified HPSG trees mapped from the HPSG treebank created by the Mayz. We treated the POS tags, which are common to the PTB-CFG grammar and the Enju HPSG grammar, as the terminal symbols.

Although we can use a single synchronous grammar for the conversion of both directions (i.e., from PTB-CFG trees to simplified HPSG trees, and the opposite), we used two different synchronous grammars to achieve better conversion accuracies. The two synchronous grammars were created by applying different pre-processing to the parallel treebank. Specifically, for the HPSG→PTB-CFG direction, 1) the PTB-CFG trees in the parallel treebank were binarized in such a way that maximized the number of common-span node pairs;[2] 2) commas in the PTB-CFG trees were raised as high as possible, approximating the change of the position of commas by the Mayz program; 3) the POS tags for 'not' are changed from 'RB' to 'RB-not,' because in PTB, 'not' is treated differently from other adverbs; 4) base NPs in PTB-CFG trees are marked as 'NP-B.' The PTB-CFG trees converted from the Enju parser's output were post-processed so that the effect of the pre-processing was removed; i.e., artificially created non-terminal nodes like '$A*$' were removed, 'RB-not' was changed to 'RB', and 'NP-B' was changed to 'NP.'[3]

For the PTB-CFG→HPSG direction, 1) PTB-CFG trees in the parallel treebank are head-centered-binarized by using Collins' head finding rules; 2) the same pre-processing as for the comma-raising, 'not' adverbs, and base NPs was applied to the PTB-CFG trees. The same pre-processing was applied to the CFG parsers' output before the conversion to simplified HPSG trees.

4.3 Accuracy of the Tree Conversion

To evaluate the accuracy of the tree conversion, we converted the trees in section 23* of the HPSG/PTB treebank into the other format (target format) and compared the conversion results against the corresponding trees in the gold-standard treebank of the target format. Table 1 presents the result of the evaluation. With the exception of the last column, the columns list the PARSEVAL metrics of the converted trees. The last column headed 'back-off%' shows the percentages of the trees for which the back-off mechanism described in Section 3 were used. The accuracy of the word-word dependencies extracted from the PTB-CFG trees converted from the HPSG treebank was 98.76%.[4]

On average, a tree in section 23* of the HPSG treebank includes 0.89 brackets that cross with the brackets in the corresponding tree in the PTB-CFG treebank, and a tree in the PTB-CFG treebank includes 0.78 brackets that cross with the brackets in the corresponding tree in the HPSG treebank. The figures in the column headed CBs (average number of crossing brackets) show that most of such crossing brackets are 'corrected' by the conversion.

4.4 Comparative Parser Evaluation using Common Representations

We measured the labeled bracketing accuracy of the parsers' output in the PTB-CFG tree representation (on section 23), and in the simplified HPSG

[2]The non-terminal nodes artificially introduced in the binarization process were labeled as '$A*$', where 'A' is the label of the nearest 'non-artificial' ancestor node.

[3]The positions of commas were left unchanged, because we ignored commas in the evaluation, just as in the standard way of evaluating CFG parsers based on the PARSEVAL metrics.

[4]Dependency relations involving a punctuation mark as the modifier were not counted in the evaluation.

Conversion direction	LP	LR	F_1	CBs	0 CBs	≤ 2 CBs	back-off%
HPSG $\rightarrow$ PTB-CFG	98.41	98.08	98.24	0.01	99.34	100.00	2.59
PTB-CFG $\rightarrow$ HPSG	97.54	97.45	97.49	0.13	93.99	98.33	10.80

Table 1: Accuracy of the tree conversion

Parser	LP	LR	F_1
Charniak and Johnson	91.79	90.88	91.33
Charniak	89.49	88.78	89.13
Collins (model 3)	88.62	88.28	88.45
Collins (model 2)	88.48	88.15	88.31
Collins (model 1)	87.94	87.51	87.72
Stanford lexicalized	86.36	86.47	86.41
Enju + tree conv.	87.18	86.47	86.82

Table 2: PTB-CFG tree evaluation

Parser (+ tree conv.)	LP	LR	F_1
Charniak and Johnson	91.44	91.31	91.37
Charniak	90.07	89.97	90.02
Collins (model 3)	88.79	88.38	88.58
Collins (model 2)	88.74	88.33	88.53
Collins (model 1)	88.62	88.48	88.54
Stanford lexicalized	88.04	88.12	88.08
Enju	90.79	90.30	90.54

Table 3: Simplified HPSG tree Evaluation

Parser	accuracy
Charniak and Johnson	93.66
Charniak	92.50
Collins (model 3)	91.15
Collins (model 2)	91.12
Collins (model 1)	90.66
Stanford lexicalized	90.91
Enju + tree conv.	90.77

Table 4: Unlabeled dependency evaluation

Modifier	Head		Enju	Charinak	p-value
t_m	t_c	t_w	X (Z)	Y (W)	
VB	ROOT	NN	20 (20)	1 (1)	8.57e-5
NN	VB	NN	69 (94)	30 (55)	1.34e-4
NN	VB	ROOT	14 (14)	0 (0)	5.12e-4
CD	CD	IN	0 (0)	11 (11)	2.57e-3
CD	NN	CD	12 (12)	1 (2)	5.55e-3
VB	VB	NN	17 (26)	4 (9)	8.83e-3
DT	NN	VB	10 (15)	1 (4)	1.59e-2
DT	NN	JJ	7 (10)	0 (3)	2.33e-2
VB	VB	MD	7 (9)	0 (1)	2.33e-2
TO	VB	NN	24 (31)	10 (19)	2.58e-2

Table 5: Error types with smallest p-values

tree representation (on section 23*). The results are shown in Table 2 and Table 3. The accuracy of word-word dependencies extracted from the PTB-CFG representation is shown in Table 4. As shown in the previous section, approximately 2% of the brackets are wrong and approximately 1% of the word-word dependencies are wrong after the conversion in both direction even if a parser gives 100% correct output. Taking this into consideration, we might conclude, utilizing the results shown in the three tables, that the performance of the Enju parser is roughly the same level as Collins' parser and Stanford's lexicalized parser. Another notable fact depicted in the tables is that Charniak and Johnson's reranking parser outperforms the Enju parser even in the simplified HPSG tree representation.

4.5 Comparative Error Analysis

To closely examine the difference in the parsers' performances, we conducted a comparative error analysis using the word-word dependency representation of the parsing results. Specifically, we tested the difference in the ability of resolving specific types of syntactic ambiguities between two parsers by using McNemar's test. To make the samples for the McNemar's test, we identified a subset of tokens of type t_m in the test set that have a true head of type t_c and another word of type t_w that is confusable as the head. For example, a subset of tokens where (t_m, t_c, t_w) = (preposition, noun, verb) can be used to test the difference in the parsers' ability to resolve PP-attachment ambiguities, by extracting the pairs of the predicted head words for the prepositions from two parsers' output, and using them as the sample set for the McNemar's test. When comparing two parsers A and B, we approximated such a subset of tokens by collecting the tokens of type t_m, which have true head of type t_c, and for which either A or B predicted a wrong head of type t_w.

We show the results of the comparison between the Enju parser and the Charinak's parser in Table 5. We used section 22 of PTB for this experiment. The table lists the type of the ambiguities (t_m, t_c, t_w) for which the accuracies by the two parsers differ with the smallest p-values. Note that the POS types t_m, t_c, and t_w are determined from the gold-standard POS tags, not the POS tags given by parsers or the POS tagger. In the table, X is the number of the tokens of type (t_m, t_c, t_w) for which only the Enju parser outputs wrong heads; Z is the total number of wrong predictions of that type made by the Enju parser; Y and W are de-

fined similarly for the Charinak's parser.

The results indicate, for example, that there is a significant difference between the two parsers in the ability to identify the root of a sentence (the first and third row). Investigation of the Enju parser's output including the root identification errors revealed that almost all of the errors of this type were caused by POS-tagging error; for example, when the matrix verb of a sentence, like 'costs,' is mistakenly tagged as a noun, the matrix verb is not identified as the root (as in the case with the first row), and the subject is often mistakenly identified as the root (as in the case with the third row). This weak-point of the Enju parser is a consequence of the architecture of the parser, wherein the POS-tagging phase is completely separated from the parsing phase. Although this weak-point has already been pointed out (Yoshida et al., 2007), we expect that we can identify other reasons for the difference in the parsers' performances by investigating the other types of errors with small p-values as well.

5 Conclusion

We have proposed the use of shallow CFG analyses as the common representation for the comparative performance analysis of parsers based on different grammar frameworks. We have presented a method to convert the parsers' output to the shallow CFG analyses that is based on synchronous grammars. The experimental results showed that our method gave a conversion accuracy of around 98% in terms of the labeled bracketing accuracy, which was sufficiently high for extracting a meaningful conclusion from a quantitative comparison of the parsers' performance. Furthermore, we conducted a comparative analysis of the parsing results represented in word-word dependencies extracted from the shallow CFG analyses. As a result, we could identified a weak-point of the HPSG parser that comes from the parser's architecture.

Acknowledgements

This work was partially supported by Grant-in-Aid for Specially Promoted Research (MEXT, Japan).

References

Carroll, J., T. Briscoe, and A. Sanlippo. 1998. Parser evaluation : A survey and a new proposal. In *In Proceedings First Conference on Linguistic Resources*, pages 447–455.

Charniak, E. and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. ACL*, pages 173–180.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proc. NAACL*, pages 132–139.

Chiang, D. 2007. Hierarchical phrase-based translation. *Comput. Linguist.*, 33(2):201–228.

Clark, S. and J. R. Curran. 2004. The importance of supertagging for wide-coverage ccg parsing. In *Proc. COLING*, pages 282–288.

Clark, S. and J. Curran. 2007. Formalism-independent parser evaluation with ccg and depbank. In *Proc. ACL*, pages 248–255.

Collins, M. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Eisner, J. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proc. ACL, Companion Volume*, pages 205–208.

Hockenmaier, J. and M. Steedman. 2007. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.

Klein, D. and C. D. Manning. 2003. A parsing: fast exact viterbi parse selection. In *Proc. NAACL*, pages 40–47.

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz. 1994. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Miyao, Y., T. Ninomiya, and J. Tsujii. 2005. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Natural Language Processing - IJCNLP 2004*, volume 3248 of *LNAI*, pages 684–693. Springer-Verlag.

Miyao, Y., K. Sagae, and J. Tsujii. 2007. Towards framework-independent evaluation of deep linguistic parsers. In *Proc. GEAF*, pages 238–258.

Ninomiya, T., Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic hpsg parsing. In *Proc. IWPT*, pages 103–114.

Ninomiya, T., T. Matsuzaki, Y. Miyao, and J. Tsujii. 2007. A log-linear model with an n-gram reference distribution for accurate hpsg parsing. In *Proc. IWPT*, pages 60–68.

Pollard, C. and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Yoshida, K., Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2007. Ambiguous part-of-speech tagging for improving accuracy and domain portability of syntactic parsers. In *Proc. IJCAI*, pages 1783–1788.