# Empty element recovery by spinal parser operations

**Katsuhiko Hayashi** and **Masaaki Nagata**
NTT Communication Science Laboratories, NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0237 Japan
{hayashi.katsuhiko, nagata.masaaki}@lab.ntt.co.jp

## Abstract

This paper presents a spinal parsing algorithm that can jointly detect empty elements. This method achieves state-of-the-art performance on English and Japanese empty element recovery problems.

## 1 Introduction

Empty categories, which are used in Penn Treebank style annotations to represent complex syntactic phenomena like constituent movement and discontinuous constituents, provide important information for understanding the semantic structure of sentences. Previous studies attempt empty element recovery by casting it as linear tagging (Dienes and Dubey, 2003), PCFG parsing (Schmid, 2006; Cai et al., 2011) or post-processing of syntactic parsing (Johnson, 2002; Gabbard et al., 2006). To the best of our knowledge, the results reported by (Cai et al., 2011) are the best yet reported, so we pursue a method that uses syntactic parsing to jointly solve the empty element recovery problem.

Our proposal uses the spinal Tree Adjoining Grammar (TAG) formalism of (Carreras et al., 2008). The spinal TAG has a set of elementary trees, called spines, each consisting of a lexical anchor with a series of unary projections. Figure 1 displays (a) a head-annotated constituent tree and (b) spines extracted from the tree. This paper presents a transition-based algorithm together with several operations to combine spines for constructing full parse trees with empty elements. Compared with the PCFG parsing approaches, one advantage of our method is its flexible feature representations, which allow the incorporation of constituency-, dependency- and spine-based features. Of particular interest, the motivation for our spinal TAG-based approach comes from the intuition that features extracted from spines can be expected to be useful for empty element recovery in the same way as constituency-based vertical higher-order conjunctive features are used in recent post-processing methods (Xiang et al., 2013; Takeno et al., 2015). Experiments on English and Japanese datasets empirically show that our system outperforms existing alternatives.

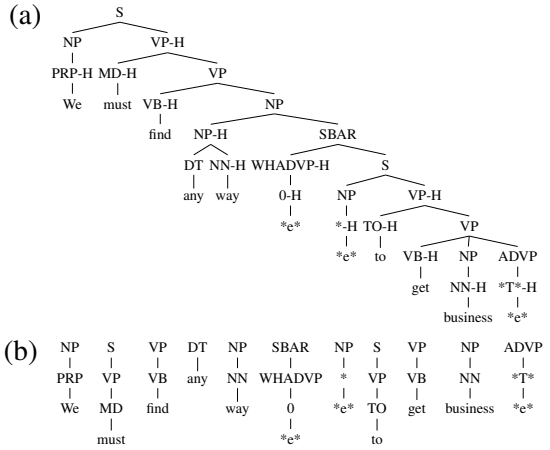


Figure 1: (a) an example of a constituent tree with head annotations denoted by -H; (b) spinal elementary trees extracted from the parse tree (a).

## 2 Spinal Tree Adjoining Grammars

We define here the spinal TAG $G = (N, PT, T, LS)$ where $N$ is a set of nonterminal symbols, $PT$ is a set of pre-terminal symbols (or part-of-speech tags), $T$ is a set of terminal symbols (or words), and $LS$ is a set of **lexical spines**. Each spine, $s$, has the form $n_0 \rightarrow n_1 \rightarrow \cdots \rightarrow n_{k-1} \rightarrow n_k$ ($k \in \mathbb{N}$) which satisfies the conditions:

- $n_0 \in T$ and $n_1 \in PT$,
- $\forall i \in [2,k]$, $n_i \in N$.

The **height** of spine $s$ is $ht(s) = k+1$ and for some position $i \in [0,k]$, the **label** at $i$ is $s(i) = n_i$. Tak-

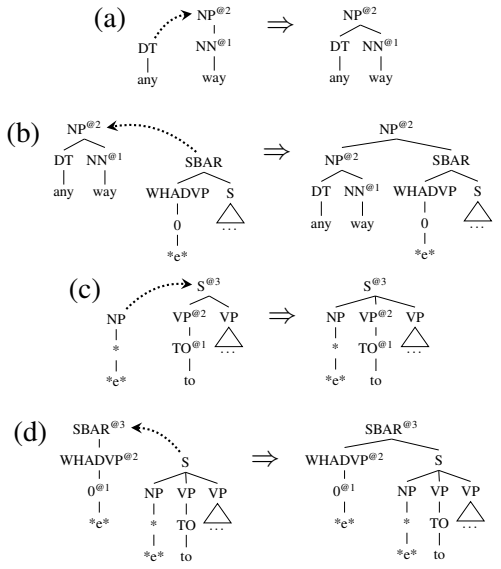


Figure 2: An example of parser operations: (a) sister adjunction left (b) regular adjunction right (c) insert left (d) combine right.

ing the leftmost spine $s$ = We → PRP → NP in Figure 1 (b), $ht(s) = 3$ and $s(1)$ = PRP.

The spinal TAG uses two operations, **sister** and **regular** adjunctions, to combine spines. Both adjunctions also have **left** and **right** types. Figures 2 (a) and (b) show examples of sister adjunction left and regular adjunction right operations. We use @# to illustrate node position on a spine, explicitly. After a regular adjunction, the resulting tree has an additional node level which has a copy of its original node at position $@x$, while a sister adjunction simply inserts a spine into some node of another spine. If adjunction left (or right) inserts spine $s_1$ into some node at $@x$ on spine $s_2$, we call $s_2$ the **head spine** of $s_1$ and $s_1$ the **left** (or **right**) **child spine** of $s_2$[1]. This paper denotes sister adjunction left and right as $s_1 \circledrightarrow^x s_2$, $s_2 \circledleft^x s_1$, regular adjunction left and right as $s_1 \blacktriangleright^x s_2$, $s_2 \blacktriangleleft^x s_1$, respectively.

## 3 Arc-Standard Shift-Reduce Spinal TAG Parsing

There are three algorithms for spinal TAG parsing, (1) Eisner-Satta CKY (Carreras et al., 2008), (2) arc-eager shift-reduce (Ballesteros and Carreras, 2015) and (3) arc-standard shift-reduce (Hayashi et al., 2016) algorithms. This paper uses the arc-standard shift-reduce algorithm since it provides a more simple implementation.

A **transition system** for spinal TAG parsing is the tuple $S = (C, T, I, C_t)$, where $C$ is a set of configurations, $T$ is a set of transitions, which are partial functions $t : C \rightharpoonup C$, $I$ is a total initialization function mapping each input string to a unique configuration, and $C_t \subseteq C$ is a set of terminal configurations. A **configuration** is the tuple $(\alpha, \beta, A)$ where $\alpha$ is a **stack** of stack elements, $\beta$ is a **buffer** of elements from an input, and $A$ is a set of parser operations. A **stack element** s is a pair $(s, j)$ where $s$ is a spine and $j$ is a node index of $s$. We refer to $s$ and $j$ of s as s.$s$ and s.$j$, respectively.

Let $\mathbf{x} = \langle \mathrm{w}_1/\mathrm{t}_1, \ldots, \mathrm{w}_n/\mathrm{t}_n \rangle$ ($\forall i \in [1,n]$, $\mathrm{w}_i \in T$ and $\mathrm{t}_i \in PT$) be a pos-tagged input sentence. The **arc-standard transition system** by Hayashi et al. (2016) can be defined as follows: its initialization function is $I(\mathbf{x}) = ([], [\mathrm{w}_1/\mathrm{t}_1, \ldots, \mathrm{w}_n/\mathrm{t}_n], \emptyset)$, its set of terminal configurations is $C_t = ([], [], A)$, and it has the following transitions:

1. for each $s \in LS$ with $s(0) = \mathrm{w}_i$ and $s(1) = \mathrm{t}_i$, a shift transition of the form $(\alpha, \mathrm{w}_i/\mathrm{t}_i|\beta, A) \vdash (\alpha|\mathrm{s}_1, \beta, A)$ where $\mathrm{s}_1 = (s, 2)$[2];

2-3. for each $j$ with $\mathrm{s}_1.j \leq j < ht(\mathrm{s}_1.s)$, a sister adjunction left transition of the form

$$(\sigma|\mathrm{s}_2|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{\mathrm{s}_2.s \circledrightarrow^j \mathrm{s}_1.s\})$$

and a regular adjunction left transition of the form

$$(\sigma|\mathrm{s}_2|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{\mathrm{s}_2.s \blacktriangleright^j \mathrm{s}_1.s\})$$

where $\mathrm{s}'_1 = (\mathrm{s}_1.s, j)$;

4-5. for each $j$ with $\mathrm{s}_2.j \leq j < ht(\mathrm{s}_2.s)$, a sister adjunction right transition of the form

$$(\sigma|\mathrm{s}_2|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{\mathrm{s}_2.s \circledleft^j \mathrm{s}_1.s\})$$

and a regular adjunction right transition of the form

$$(\sigma|\mathrm{s}_2|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{\mathrm{s}_2.s \blacktriangleleft^j \mathrm{s}_1.s\})$$

where $\mathrm{s}'_1 = (\mathrm{s}_2.s, j)$;

6. a finish transition of the form $([\mathrm{s}], [], A) \vdash ([], [], A)$.

[1] After adjunctions, the result forms a phrase consisting of several spines. If a phrasal spine is also used in adjunction operations as Figure 2 (b), we treat it as a lexical spine by referring to its head spine.

[2] To construct a full parse tree from $A$, our actual implementation attaches index $i$ to spine $s$ after shift transition.

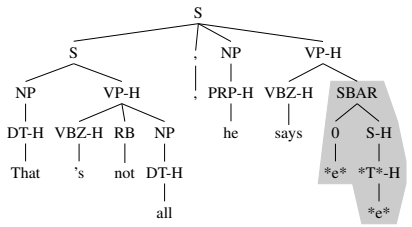

Figure 3: A phrasal empty spine shown on the shaded region.

To reduce search errors, Hayashi et al. (2016) employed beam search with Dynamic Programming of (Huang and Sagae, 2010). For experiments, we also use this technique and discriminative modeling of (Hayashi et al., 2016).

## 4 Empty Element Recovery

### 4.1 Spinal TAG with Empty Elements

In this paper, we redefine the spinal TAG as $G = (N, PT, T, LS, \text{*e*}, ET, ES)$, where *e* is a special word, $ET$ is a set of empty categories, and $ES$ is a set of empty spines. An **empty spine** $s = n_0 \rightarrow n_1 \rightarrow \cdots \rightarrow n_{k-1} \rightarrow n_k$ ($k \in \mathbb{N}$) has the same form as lexical spines, but $n_0 =$ *e* and $n_1 \in ET$. The height and label definitions are also the same as those of lexical spines. For example, the rightmost spine $s =$ *e* $\rightarrow$ *T* $\rightarrow$ ADVP in Figure 1 (b) is an empty spine with $ht(s) = 3$ and $s(1) =$ *T*.

This paper extends empty spines to allow the use of phrasal constituents that consist of only empty elements, as a single spine. A **phrasal empty spine** is a tuple $(t, h)$, where $t$ is a sequence of (phrasal) empty spines specifying some sister adjunctions between these spines and $h$ is a head spine in $t$. The phrasal empty spine in Figure 3 consists of two empty spines *e* $\rightarrow$ 0 and *e* $\rightarrow$ *T* $\rightarrow$ S $\rightarrow$ SBAR, where a sister adjunction left is performed at the SBAR node of the latter spine, which is a head spine in the phrase. To apply parser operations to a phrasal empty spine, we use its head spine rather than itself. This paper defines the height and label of a phrasal empty spine as those of its head spine.

To recover empty elements, this paper introduces two additional operations, **insert** and **combine**, both of which have **left** and **right** types. Figures 2 (c) and (d) show insert left and combine right operations. These operations are similar to sister adjunctions in that the former simply inserts some phrasal empty spine into some node of another spine and the latter also inserts a spine into some node of a phrasal empty spine.

### 4.2 New Transitions

To handle empty spines in parsing process, we add the following five transitions to the arc-standard transition system of (Hayashi et al., 2016):

7-8. for each $s \in ES$ and each $j$ with $\mathrm{s}_1.j \leq j < ht(\mathrm{s}_1.s)$, an insert left transition of the form

$$(\sigma|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{s \triangleright^j \mathrm{s}_1.s\})$$

and an insert right transition of the form

$$(\sigma|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{\mathrm{s}_1.s \triangleleft^j s\})$$

where $\mathrm{s}'_1 = (\mathrm{s}_1.s, j)$;

9-10. for each $s \in ES$ and each $j$ with $2 \leq j < ht(s)$, a combine left transition of the form

$$(\sigma|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{\mathrm{s}_1.s \triangleright^j s\})$$

and a combine right transition of the form

$$(\sigma|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}'_1, \beta, A \cup \{s \triangleleft^j \mathrm{s}_1.s\})$$

where $\mathrm{s}'_1 = (s, j)$;

11. an idle transition of the form $(\sigma|\mathrm{s}_1, \beta, A) \vdash (\sigma|\mathrm{s}_1, \beta, A)$;

Like **unary** and **idle** rules in shift-reduce CFG parsing (Zhu et al., 2013), our current system prohibits $> b$ consecutive actions consisting of only insert, combine and idle operations. Given an input sentence with length $n$, after performing $n$ shift, $n-1$ adjunction, $b \cdot (2n-1)$ {insert, combine or idle} actions, the system triggers the finish action and terminates. For training, we make oracle derivations using the stack-shortest strategy.

## 5 Related Work

To realize empty element recovery, other lexicalized TAG formalisms (Chen and Shanker, 2004; Shen et al., 2008) attach some or all empty elements directly to surface word lexicons. Our framework, however, uses spinal TAG parser operations as they provide more efficient parsing and more compact sets of lexicons. It is remarkable that this paper is the first study to present a shift-reduce spinal TAG parsing algorithm to recover empty elements.

Recent work has shown that empty element recovery can be effectively solved in conjunction

| | Tagger | | Lattice | | Proposed | | |
|---|---|---|---|---|---|---|---|
| | M | O | M | O | M | O | Gold |
| *ICH* | 2 | 5 | 2 | 2 | 31 | 43 | 78 |
| *RNR* | 0 | 3 | 0 | 4 | 4 | 5 | 6 |
| *EXP* | 10 | 12 | 0 | 0 | 19 | 26 | 30 |

Table 2: Result Analysis: M denotes the number of matches of system outputs (O) with the gold.

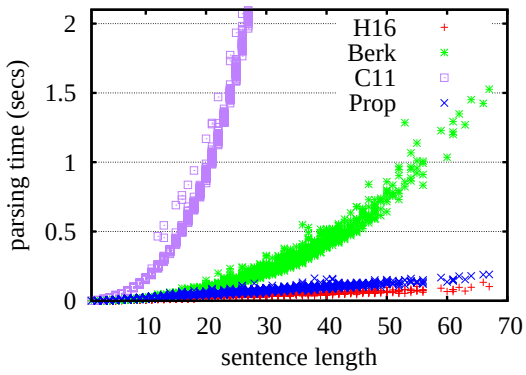


Figure 4: Scatter plot of parsing time against sentence length, comparing with Hayashi16, Berkeley and Cai11 parsers.

with parsing (Schmid, 2006; Cai et al., 2011). Schmid (2006) annotated a constituent tree with slash features to recover a direct path from a filler node to its trace. Cai et al. (2011) successfully integrated empty element recovery into lattice parsing for latent PCFGs. Compared with PCFG parsing, the spinal TAG parser provides a more flexible feature representation.

## 6 Experiments

### 6.1 Experiments on the English Penn Treebank

We used the Wall Street Journal (WSJ) part of the English Penn Treebank: Sections 02–21 were used for training, Section 22 for development, and Section 23 for testing. We annotated trees with heads by **treep** (Chiang and Bikel, 2002)[3] with the application of Collins's head rules. The 78524 lexical and 115 phrasal empty spine types were obtained from the training data[4]. The set of phrasal empty spines covered all phrasal empty spines extracted from the development data.

We used the Stanford part-of-speech tagger to tag development and test data. To train the proposed parsing model, we used the violation–fixing

[3] http://www3.nd.edu/~dchiang/software/treep/treep.html

[4] Excluding words from lexical spines, there were 1080 lexical spine types.

| | Typed-empty (t,*i*,*i*) | | | All Brackets | | |
|---|---|---|---|---|---|---|
| | P | R | F1 | P | R | F1 |
| Rule | 57.4 | 50.5 | 53.7 | – | – | – |
| Takeno15 | 60.4 | 50.6 | 55.1 | – | – | – |
| Tagger | 63.1 | 34.7 | 44.8 | 72.9 | 68.6 | 70.7 |
| Lattice | 64.1 | 52.2 | 57.5 | 73.7 | 70.6 | 72.1 |
| Proposed | **65.3** | **57.6** | **61.2** | **74.3** | **72.8** | **73.6** |

Table 3: Results on the Japanese Keyaki Treebank.

perceptron algorithm (Huang et al., 2012). For training and testing, we set beam size to 16 and max count $b$, introduced in Section 4.2, to 2. For comparison with other systems in our environment, we also implemented two systems:

- **Lattice** is a method by Cai et al. (2011). We also used **blatt**[5], which is an extension of the Berkeley parser, to parse word lattices in which the special word *e* is encoded as described in (Cai et al., 2011).

- **Tagger** decides whether some empty category is inserted at the front of a word or not, with regularized logistic regression. To simplify point-wise linear tagging, we combined empty categories, those that appeared in the same position of a sentence, into a single category: thus the original 10 empty types increased to 63.

Table 1 shows final results on Section 23. To evaluate the accuracy of empty element recovery, we calculated precision, recall and F1 scores for (1) Labeled Empty Bracket (X/t,*i*,*i*), (2) Labeled Empty Element (t,*i*,*i*), and (3) All Brackets, where X $\in NT$, t $\in ET$ and $i$ is a position of the empty element, using **eevalb**[6]. The results clearly show that our proposed method significantly outperforms the other systems. Table 2 shows the main reason for the improvement achieved by our method. The *ICH*, *RNR* and *EXP* empty types are used to show the relation between non-adjacent constituents, caused by syntactic phenomena like Extraposition and Conjunction. Our method captures such complex relations better with the help of the syntactic feature richness.

Table 1 reports the scores for non-empty brackets to examine whether the joint method improves the standard PARSEVAL scores. While the Lattice

[5] http://www.cs.bgu.ac.il/~yoavg/software/blatt/

[6] http://www3.nd.edu/~dchiang/software/eevalb.py

| | Johnson (X/t,*i*,*i*) | | | Typed-empty (t,*i*,*i*) | | | All Brackets | | | Non-empty Brackets | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | P | R | F1 | P | R | F1 | P | R | F1 | P | R | F1 |
| Schmid06 | – | – | – | 87.9 | 83.0 | 85.4 | – | – | – | – | – | – |
| Cai11 | 90.1 | 79.5 | 84.5 | 92.3 | 80.9 | 86.2 | 90.1 | 88.5 | 89.3 | – | – | – |
| Tagger | 89.7 | 69.3 | 78.1 | 90.7 | 70.1 | 79.0 | 87.8 | 85.5 | 86.7 | 87.8 | 86.8 | 87.3 |
| Lattice (Cai11) | 89.8 | 79.2 | 84.2 | 91.4 | 80.6 | 85.7 | 90.2 | 88.7 | 89.5 | 90.2 | 89.5 | 89.8 |
| Proposed | **90.3** | **81.7** | **85.8** | **91.8** | **83.2** | **87.3** | **90.8** | **89.7** | **90.3** | 90.8 | 90.3 | 90.6 |
| Berkeley | – | – | – | – | – | – | – | – | – | 89.9 | 90.3 | 90.1 |
| Hayashi16 | – | – | – | – | – | – | – | – | – | **90.9** | **90.4** | **90.7** |

Table 1: Results on the English Penn Treebank (Section 23): to calculate the scores for Tagger, we obtained a parse tree by supplying the 1-best Tagger output with the Berkeley parser trained on Sections 02-21 including empty elements (using the option "-useGoldPOS").

method was less accurate than the vanilla Berkeley parser, the performance of our method could be maintained with little loss in parsing accuracy. Figure 4 shows the parse time in seconds for each test sentence and that our empty element recovery parser works in reasonable time.

### 6.2 Experiments on the Japanese Keyaki Treebank

Finally, to show that our method works well on other languages, we conduct experiments on the Japanese Keyaki Treebank (Butler et al., 2012). For this data, we modified blatt to keep function labels And, in order to consider segmentation errors, we also modified eevalb to calculate not word but character span in a sentence. We follow the experiments in (Takeno et al., 2015) and show the results in Table 3. Our method significantly outperforms the state-of-the-art post-processing method in Japanese.

## 7 Conclusion and Future Work

Using spinal parsing for the joint recovery of empty elements achieves state-of-the-art performance in standard English and Japanese datasets. We plan to extend our work to recover trace-filler and frame semantic structures using the PropBank data.

## Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This work was supported in part by JSPS KAKENHI Grant Number 26730126.

## References

M. Ballesteros and X. Carreras. 2015. Transition-based spinal parsing. In *Proc. of CoNLL*.

A. Butler, Z. Hong, T. Hotta, R. Otomo, K. Yoshimoto, and Z. Zhou. 2012. Keyaki treebank: phrase structure with functional information for japanese. In *Proc. of Text Annotation Workshop*.

S. Cai, D. Chiang, and Y. Goldberg. 2011. Language-independent parsing with empty elements. In *Proc. of ACL-HLT*, pages 212–216.

X. Carreras, M. Collins, and T. Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proc. of CoNLL*, pages 9–16.

J. Chen and V. K. Shanker. 2004. Automated extraction of tags from the penn treebank. In *New developments in parsing technology*, pages 73–89. Springer.

D. Chiang and D. M. Bikel. 2002. Recovering latent information in treebanks. In *Proc. of COLING*, pages 1–7.

P. Dienes and A. Dubey. 2003. Deep syntactic processing by combining shallow methods. In *Proc. of ACL*, pages 431–438.

R. Gabbard, M. Marcus, and S. Kulick. 2006. Fully parsing the penn treebank. In *Proc. of NAACL-HLT*, pages 184–191.

K. Hayashi, J. Suzuki, and M. Nagata. 2016. Shift-reduce spinal tag parsing with dynamic programming. *Transactions of the Japanese Society for Artificial Intelligence*, 31(2).

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proc. of ACL*, pages 1077–1086.

L. Huang, S. Fayong, and Y. Guo. 2012. Structured perceptron with inexact search. In *Proc. of NAACL*, pages 142–151.

M. Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proc. of ACL*, pages 136–143.

H. Schmid. 2006. Trace prediction and recovery with unlexicalized pcfgs and slash features. In *Proc. of COLING-ACL*, pages 177–184.

L. Shen, L. Champollion, and A. K. Joshi. 2008. Ltag-spinal and the treebank. *Language Resources and Evaluation*, 42(1):1–19.

S. Takeno, M. Nagata, and K. Yamamoto. 2015. Empty category detection using path features and distributed case frames. In *Proc. of EMNLP*, pages 1335–1340.

B. Xiang, X. Luo, and B. Zhou. 2013. Enlisting the ghost: modeling empty categories for machine translation. In *Proc. of ACL*, pages 822–831.

M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. of ACL*, pages 434–443.