Transition-Based Left-Corner Parsing for Identifying PTB-Style Nonlocal Dependencies

Yoshihide Kato[1] and **Shigeki Matsubara**[2]
[1]Information & Communications, Nagoya University
[2]Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601 Japan
yoshihide@icts.nagoya-u.ac.jp

Abstract

This paper proposes a left-corner parser which can identify nonlocal dependencies. Our parser integrates nonlocal dependency identification into a transition-based system. We use a structured perceptron which enables our parser to utilize global features captured by nonlocal dependencies. An experimental result demonstrates that our parser achieves a good balance between constituent parsing and nonlocal dependency identification.

1 Introduction

Many constituent parsers based on the Penn Treebank (Marcus et al., 1993) are available, but most of them do not deal with nonlocal dependencies. Nonlocal dependencies represent syntactic phenomenon such as wh-movement, A-movement in passives, topicalization, raising, control, right node raising and so on. Nonlocal dependencies play an important role on semantic interpretation. In the Penn Treebank, a nonlocal dependency is represented as a pair of an empty element and a filler.

Several methods of identifying nonlocal dependencies have been proposed so far. These methods can be divided into three approaches: pre-processing approach (Dienes and Dubey, 2003b), in-processing approach (Dienes and Dubey, 2003a; Schmid, 2006; Cai et al., 2011; Kato and Matsubara, 2015) and post-processing approach (Johnson, 2002; Levy and Manning, 2004; Campbell, 2004; Xue and Yang, 2013; Xiang et al., 2013; Takeno et al., 2015).[1] In pre-processing approach, a tagger called "trace tagger" detects empty elements. The trace tagger uses only surface word information. In-processing approach integrates nonlocal dependency identification into a parser. The parser uses a probabilistic context-free grammar to rank candidate parse trees. Post-processing approach recovers nonlocal dependencies from a parser output which does not include nonlocal dependencies.

The parsing models of the previous methods cannot use global features captured by nonlocal dependencies. Pre- or in-processing approach uses a probabilistic context-free grammar, which makes it difficult to use global features. Post-processing approach performs constituent parsing and nonlocal dependency identification separately. This means that the constituent parser cannot use any kind of information about nonlocal dependencies.

This paper proposes a parser which integrates nonlocal dependency identification into constituent parsing. Our method adopts an in-processing approach, but does not use a probabilistic context-free grammar. Our parser is based on a transition system with structured perceptron (Collins, 2002), which can easily introduce global features to its parsing model. We adopt a left-corner strategy in order to use the syntactic relation *c-command*, which plays an important role on nonlocal dependency identification. Previous work on transition-based constituent parsing adopts a shift-reduce strategy with a tree binarization (Sagae and Lavie, 2005; Sagae and Lavie, 2006; Zhang and Clark, 2009; Zhu et al., 2013; Wang and Xue, 2014; Mi and Huang, 2015; Thang et al., 2015; Watanabe and Sumita, 2015), or convert constituent trees to "spinal trees", which are similar to dependency trees (Ballesteros and Carreras, 2015). These conversions make it difficult for their parsers to capture c-command relations in the parsing process. On the other hand, our parser does not require such kind of conversion.

[1]The methods of (Cai et al., 2011; Xue and Yang, 2013; Xiang et al., 2013; Takeno et al., 2015) only detect empty elements.

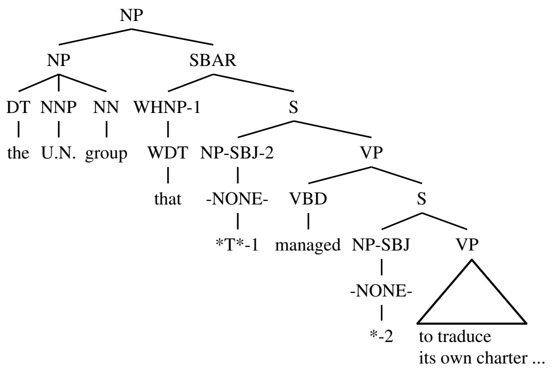

Figure 1: A parse tree in the Penn Treebank.

Our contribution can be summarized as follows:

1. We introduce empty element detection into transition-based left-corner constituent parsing.

2. We extend c-command relation to deal with nodes in *parse tree stack* in the transition system, and develop heuristic rules which coindex empty elements with their fillers on the basis of the extended version of c-command.

3. We introduce new features about nonlocal dependency to our parsing model.

This paper is organized as follows: Section 2 explains how to represent nonlocal dependencies in the Penn Treebank. Section 3 describes our transition-based left-corner parser. Section 4 introduces nonlocal dependency identification into our parser. Section 5 describes structured perceptron and features. Section 6 reports an experimental result, which demonstrated that our parser achieved a good balance between constituent parsing and nonlocal dependency identification. Section 7 concludes this paper.

2 Nonlocal Dependency

This section describes nonlocal dependencies in the Penn Treebank (Marcus et al., 1993). A nonlocal dependency is represented as a pair of an *empty element* and a *filler*. Figure 1 shows an example of (partial) parse tree in the Penn Treebank. The parse tree includes several nonlocal dependencies. The nodes labeled with `-NONE-` are empty elements. The terminal symbols such as $*$ and $*$T$*$ represent the type of nonlocal dependency: $*$ represents an unexpressed subject of to-infinitive. $*$T$*$ represents a trace of wh-movement. When a terminal symbol of empty element is indexed, its filler exists in the parse tree. The filler has the same number. For example, `*T*-1` means that the node `WHNP-1` is the corresponding filler. Table 1 gives a brief description of empty elements quoted from the annotation guideline (Bies et al., 1995). For more details, see the guideline.

3 Transition-Based Left-Corner Parsing

This section describes our transition-based left-corner parser.

As with previous work (Sagae and Lavie, 2005; Sagae and Lavie, 2006; Zhang and Clark, 2009; Zhu et al., 2013; Wang and Xue, 2014; Mi and Huang, 2015; Thang et al., 2015; Watanabe and Sumita, 2015), our transition-based parsing system consists of a set of parser states and a finite set of transition actions, each of which maps a state into a new one. A parser state consists of a stack of parse tree nodes and a buffer of input words. A state is represented as a tuple (σ, i), where σ is the stack and i is the next input word position in the buffer. The initial state is $(\langle\rangle, 0)$. The final states are in the form of $(\langle[\cdots]_{\texttt{TOP}}\rangle, n)$, where `TOP` is a special symbol for the root of the parse tree and n is the length of the input sentence. The transition actions for our parser are as follows:

- SHIFT(X): pop up the first word from the buffer, assign a POS tag X to the word and push it onto the stack.

The SHIFT action assigns a POS tag to the shifted word to perform POS tagging and constituent parsing simultaneously. This is in the same way as Wang and Xue (2014).

- LEFTCORNER-$\{$H/$\emptyset\}(X)$: pop up the first node from the stack, attach a new node labeled with X to the node as the parent and push it back onto the stack. H and $\emptyset$ indicate whether or not the popped node is the head child of the new node.

- ATTACH-$\{$H/$\emptyset\}$: pop up the top two nodes from the stack, attach the first one to the second one as the rightmost child and push it back onto the stack. H and $\emptyset$ indicate whether or not the first node is the head child of the second one.

We introduce new actions LEFTCORNER and ATTACH. ATTACH action is similar to REDUCE action standardly used in the previous transition-based parsers. However, there is an important

type	description	n-posi
$*$	arbitrary PRO, controlled PRO and trace of A-movement	L, R, $-$
$*$EXP$*$	expletive (extraposition)	R
$*$ICH$*$	interpret constituent here (discontinuous dependency)	L, R
$*$RNR$*$	right node raising	R
$*$T$*$	trace of A$'$-movement	A, L
0	null complementizer	$-$
$*$U$*$	unit	$-$
$*$?$*$	placeholder for ellipsed material	$-$
$*$NOT$*$	anti-placeholder in template gapping	$-$

Table 1: Empty elements in the Penn Treebank.

SHIFT(X)	$(\langle s_m, \ldots, s_0 \rangle, i) \Rightarrow (\langle s_m, \ldots, s_0, [w_i]_X \rangle, i+1)$
LEFTCORNER-{H/$\emptyset$}(X)	$(\langle s_m, \ldots, s_1, s_0 \rangle, i) \Rightarrow (\langle s_m, \ldots, s_1, [s_0]_X \rangle, i)$
ATTACH-{H/$\emptyset$}	$(\langle s_m, \ldots, s_2, [\sigma_1]_X, s_0 \rangle, i) \Rightarrow (\langle s_m, \ldots, s_2, [\sigma_1 s_0]_X \rangle, i)$

Figure 2: Transition actions for left-corner parsing.

difference between ATTACH and REDUCE. The REDUCE action cannot deal with any node with more than two children. For this reason, the previous work converts parse trees into binarized ones. The conversion makes it difficult to capture the hierarchical structure of the parse trees. On the other hand, ATTACH action can handle more than two children. Therefore, our parser does not require such kind of tree binarization. These transition actions are similar to the ones described in (Henderson, 2003), although his parser uses right-binarized trees and does not identify head-children. Figure 2 summarizes the transition actions for our parser.

To guarantee that every non-terminal node has exactly one head child, our parser uses the following constraints:

- LEFTCORNER and ATTACH are not allowed when s_0 has no head child.
- ATTACH-H is not allowed when s_1 has a head child.

Table 2 shows the first several transition actions which derive the parse tree shown in Figure 1. Head children are indicated by the superscript *.

Previous transition-based constituent parsing does not handle nonlocal dependencies. One exception is the work of Maier (2015), who proposes shift-reduce constituent parsing with swap action. The parser can handle nonlocal dependencies represented as discontinuous constituents. In this framework, discontinuities are directly annotated by allowing crossing branches. Since the annotation style is quite different from the PTB annotation, the parser is not suitable for identifying the PTB style nonlocal dependencies.[2]

4 Nonlocal Dependency Identification

Nonlocal dependency identification consists of two subtasks:

- empty element detection.
- empty element resolution, which coindexes empty elements with their fillers.

Our parser can insert empty elements at an arbitrary position to realize empty element detection. This is in a similar manner as the in-processing approach. Our method coindexes empty elements with their fillers using simple heuristic rules, which are developed for our transition system.

4.1 Empty Element Detection

We introduce the following action to deal with empty elements:

$$\text{E-SHIFT}(E, t): \quad (\langle s_m, \ldots, s_0 \rangle, i) \Rightarrow (\langle s_m, \ldots, s_0, [t]_E \rangle, i)$$

This action simply inserts an empty element at an arbitrary position and pops up no element from the buffer (see the transition from #11 to #12 shown in Table 2 as an example).

4.2 Annotations

For empty element resolution, we augment the Penn Treebank. For nonlocal dependency types

[2]In (Evang and Kallmeyer, 2011), the PTB-style annotation of types $*$EXP, $*$ICH$*$, $*$RNR$*$ and $*$T$*$ is transformed into an annotation with crossing branches.

action	#	state
(initial state)	1	$(\langle\rangle, 0)$
SHIFT(DT)	2	$(\langle[\text{the}]_{\texttt{DT}}\rangle, 1)$
LEFTCORNER-∅(NP)	3	$(\langle[[\text{the}]_{\texttt{DT}}]_{\texttt{NP}}\rangle, 1)$
SHIFT(NNP)	4	$(\langle[[\text{the}]_{\texttt{DT}}]_{\texttt{NP}}, [\text{U.N.}]_{\texttt{NNP}}\rangle, 2)$
ATTACH-∅	5	$(\langle[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}]_{\texttt{NP}}\rangle, 2)$
SHIFT(NN)	6	$(\langle[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}]_{\texttt{NP}}, [\text{group}]_{\texttt{NN}}\rangle, 3)$
ATTACH-H	7	$(\langle[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}[\text{group}]_{\texttt{NN*}}]_{\texttt{NP}}\rangle, 3)$
LEFTCORNER-H(NP)	8	$(\langle[[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}[\text{group}]_{\texttt{NN*}}]_{\texttt{NP*}}]_{\texttt{NP}}\rangle, 3)$
SHIFT(WDT)	9	$(\langle[[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}[\text{group}]_{\texttt{NN*}}]_{\texttt{NP*}}]_{\texttt{NP}}, [\text{that}]_{\texttt{WDT}}\rangle, 4)$
LEFTCORNER-H(WHNP-∗T∗-NP-L)	10	$(\langle[[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}[\text{group}]_{\texttt{NN*}}]_{\texttt{NP*}}]_{\texttt{NP}}, [[\text{that}]_{\texttt{WDT*}}]_{\texttt{WHNP-*T*-NP-L}}\rangle, 4)$
LEFTCORNER-H(SBAR)	11	$(\langle[[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}[\text{group}]_{\texttt{NN*}}]_{\texttt{NP*}}]_{\texttt{NP}}, [[[\text{that}]_{\texttt{WDT*}}]_{\texttt{WHNP-*T*-NP-L*}}]_{\texttt{SBAR}}\rangle, 4)$
E-SHIFT(-NONE-NP-L, ∗T∗)	12	$(\langle[[[\text{the}]_{\texttt{DT}}[\text{U.N.}]_{\texttt{NNP}}[\text{group}]_{\texttt{NN*}}]_{\texttt{NP*}}]_{\texttt{NP}}, [[[\text{that}]_{\texttt{WDT*}}]_{\texttt{WHNP-*T*-NP-L*}}]_{\texttt{SBAR}}, [*\text{T}*]_{\texttt{-NONE-NP-L}}\rangle, 4)$

Table 2: An example of transition action sequence.

∗EXP∗, ∗ICH∗, ∗RNR∗ and ∗T∗, we assign the following information to each filler and each empty element:

- The nonlocal dependency type (only for filler).
- The nonlocal dependency category, which is defined as the category of the parent of the empty element.
- The relative position of the filler, which take a value from {A, L, R}. "A" means that the filler is an ancestor of the empty element. "L" ("R") means that the filler occurs to the left (right) of the empty element. Table 1 summarizes which value each empty element can take.

The information is utilized for coindexing empty elements with fillers. Below, we write n-type(x), n-cat(x) and n-posi(x) for the information of a node x, respectively.

If an empty element of type ∗ is indexed, we annotate the empty element in the same way.[3] Furthermore, we assign a tag OBJCTRL to every empty element if its coindexed constituent does not have the function tag SBJ.[4] This enables our parser to distinguish between subject control and object control.

Figure 3 shows the augmented version of the parse tree of Figure 1.

4.3 Empty Element Resolution

Nonlocal dependency annotation in the Penn Treebank is based on Chomsky's GB-theory (Chomsky, 1981). This means that there exist c-command relations between empty elements and

[3]We omit its nonlocal dependency category, since it is always NP.

[4]In the Penn Treebank, every subject has the tag SBJ.

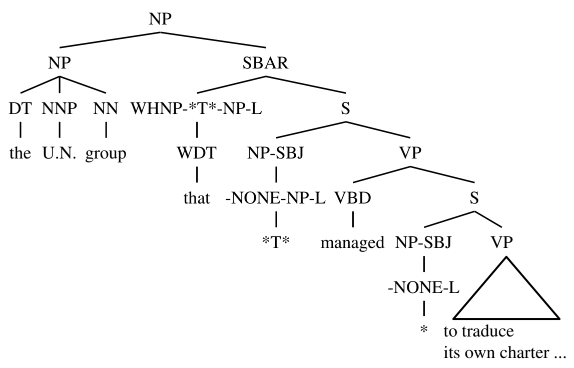

Figure 3: An augmented parse tree.

fillers in many cases. For example, all the empty elements in Figure 1 are c-commanded by their fillers. Our method coindexes empty elements with their fillers by simple heuristic rules based on the c-command relation.

4.3.1 C-command Relation

Here, we define c-command relation in a parse tree as follows:

- A node x *c-commands* a node y if and only if there exists some node z such that z is a sibling of $x (x \neq z)$ and y is a descendant of z.

It is difficult for previous transition-based shift-reduce constituent parsers to recognize c-command relations between nodes, since parse trees are binarized. On the other hand, our left-corner parser needs not to binarize parse trees and can easily recognize c-command relations. Furthermore, we extend c-command relation to handle nodes in a stack of our transition system. For two nodes x and y in a stack, the following statement necessarily holds:

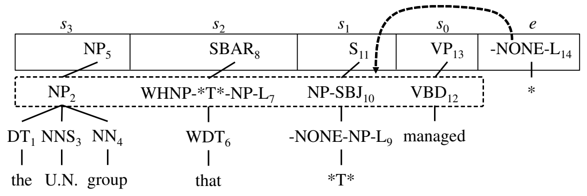

Figure 4: An example of resolution of $[*]$-NONE-L.

- Let $S = (\langle s_m, \ldots, s_0 \rangle, i)$ be a parser state. Let y be a descendant of s_j and x be a child of some node $s_k (j < k \leq m)$, respectively. Then, x c-commands y in any final state derived from the state S.

Below, we say that x c-commands y, even when the nodes x and y satisfy the above statement.

As an example, let us consider the state shown in Figure 4. The subscripts of nodes indicate the order in which the nodes are instantiated. The nodes in dotted box c-command the shifted node -NONE-L$_{14}$ in terms of the above statement. In the parse tree shown in Figure 3, which is derived from this state, these nodes c-commands -NONE-L$_{14}$ by the original definition.

4.3.2 Resolution Rules

Our parser coindexes an empty element with its filler, when E-SHIFT or ATTACH is executed. E-SHIFT action coindexes the shifted empty element e such that n-posi(e) = L with its filler. ATTACH action coindexes the attached filler s_0 such that n-posi(s_0) = R with its corresponding empty element. Resolution rules consist of three parts: PRECONDITION, CONSTRAINT and SELECT. Empty element resolution rule is applied to a state when the state satisfies PRECONDITION of the rule. CONSTRAINT represents the conditions which coindexed element must satisfy. SELECT can take two values ALL and RIGHTMOST. When there exist several elements satisfying the CONSTRAINT, SELECT determines how to select coindexed elements. ALL means that all the elements satisfying the CONSTRAINT are coindexed. RIGHTMOST selects the rightmost element satisfying the CONSTRAINT.

The most frequent type of nonlocal dependency in the Penn Treebank is $*$. Figure 5 shows the resolution rules for type $*$. Here, ch(s) designates the set of the children of s. sbj(x) means that x has a function tag SBJ. par(x) designates the parent of x. cat(x) represents the constituent category of x. des(s) designates the set of the proper descendants of s. free(x, σ) means that x is not coindexed with a node included in σ.

```
Rule: $*$-L
  PRECONDITION
    ACTION=E-SHIFT(-NONE-L, $*$)
  CONSTRAINT for coindexed element $x$
    $x \in \bigcup_{j=0}^{m}$ ch$(s_j)$        # $x$ c-commands $e$
    sbj$(x)$
  SELECT: RIGHTMOST

Rule: $*$-L-OBJCTRL
  PRECONDITION
    ACTION=E-SHIFT(-NONE-L-OBJCTRL, $*$)
  CONSTRAINT for coindexed element $x$
    $x \in \bigcup_{j=0}^{m}$ ch$(s_j)$        # $x$ c-commands $e$
    cat$(x)$ = NP $\vee$ cat$(x)$ = PP
    cat(par$(x)$) = VP
  SELECT: RIGHTMOST

Rule: $*$-R
  PRECONDITION
    ACTION=ATTACH
    sbj$(s_0)$
  CONSTRAINT for coindexed element $x$
    $x \in$ des$(s_1)$       # $s_0$ c-commands $x$
    $x = [*]$-NONE-R
    free$(x, \langle s_m, \ldots, s_0 \rangle)$
  SELECT: ALL
```

Figure 5: Resolution rules for type $*$.

The first rule $*$-L is applied to a state when E-SHIFT action inserts an empty element $e = [*]$-NONE-L. This rule seeks a subject which c-commands the shifted empty element. The first constraint means that the node x c-commands the empty element e, since the resulting state of E-SHIFT action is $(\langle s_m, \ldots, s_0, e \rangle, i)$, and x and e satisfy the statement in section 4.3.1. For example, the node NP-SBJ$_{10}$ shown in Figure 4 satisfies these constraints (the dotted box represents the first constraint). Therefore, our parser coindexes NP-SBJ$_{10}$ with -NONE-L$_{14}$.

The second rule $*$-L-OBJCTRL seeks an object instead of a subject. The second and third constraints identify whether or not x is an argument. If x is a prepositional phrase, our parser coindexes e with x's child noun phrase instead of x itself, in order to keep the PTB-style annotation.

The third rule $*$-R is for null subject of participial clause. Figure 6 shows an example of applying the rule $*$-R to a state. This rule is applied to a state when the transition action is ATTACH and s_0 is a subject. By definition, the first constraint means that s_0 c-commands x.

The second most frequent type is $*$T$*$. Figure 7 shows the rule for $*$T$*$. This rule is ap-

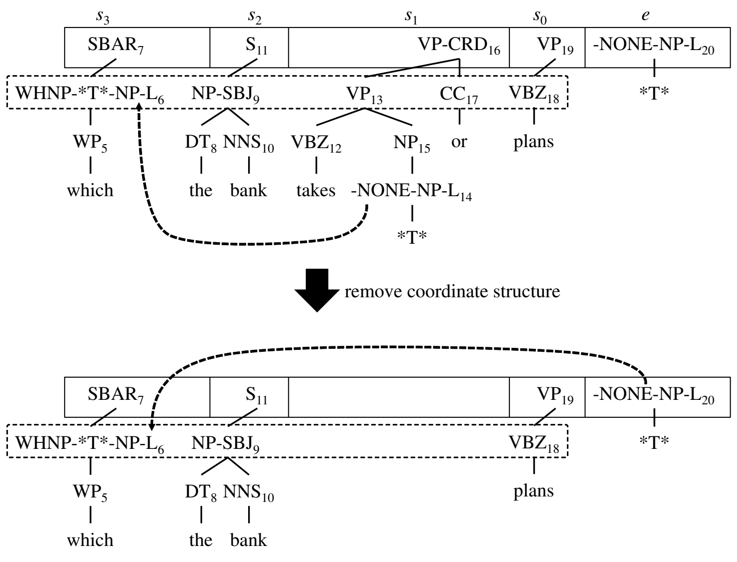

Figure 9: An example of resolution of $[*\texttt{T}*]_{\texttt{-NONE-NP-L}}$ in the case where the stack has coordinate structure.

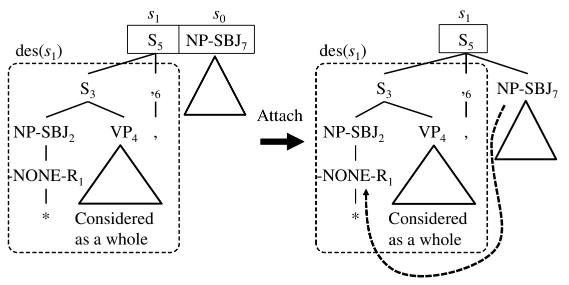

Figure 6: An example of resolution of $[*]_{\texttt{-NONE-R}}$.

```
Rule: $*\texttt{T}*\texttt{-L}$
  PRECONDITION
    ACTION=E-SHIFT(-NONE-L, $*\texttt{T}*$)
  CONSTRAINT for coindexed element $x$
    # $x$ c-commands $e$
    $x \in \bigcup_{s \in \texttt{removeCRD}(\langle s_m,\ldots,s_0\rangle)} \texttt{ch}(s)$
    $\texttt{match}(x, e)$
    $\texttt{free}(x, \texttt{removeCRD}(\langle s_m, \ldots, s_0\rangle))$
  SELECT: RIGHTMOST
```

Figure 7: Resolution rule for type $*\texttt{T}*$.

plied to a state when E-SHIFT action inserts an empty element of type $*\texttt{T}*$. Here, $\texttt{match}(x, y)$ checks the consistency between x and y, that is, $\texttt{match}(x, y)$ holds if and only if $\texttt{n-type}(x) = \texttt{n-type}(y)$, $\texttt{n-cat}(x) = \texttt{n-cat}(y)$, $\texttt{n-posi}(x) = \texttt{n-posi}(y)$, $\texttt{cat}(x) \neq$ -NONE- and $\texttt{cat}(y) =$ -NONE-. $\texttt{removeCRD}(\langle s_m, \ldots, s_0\rangle)$ is a stack which is obtained by removing $s_j (0 \leq j \leq m)$ which is annotated with a tag CRD.[5] The tag CRD

[5]We assign a tag CRD to a node, when it matches the pattern $[\cdots[\cdots]_X \cdots [\cdots]_{(\texttt{CC}|\texttt{CONJP}|,|:)} \cdots [\cdots]_X \cdots]_X$.

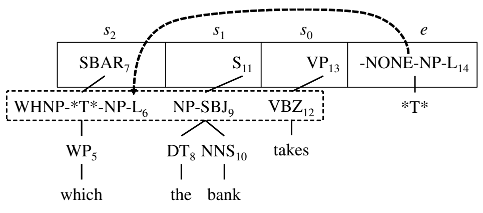

Figure 8: An example of resolution of $[*\texttt{T}*]_{\texttt{-NONE-NP-L}}$.

means that the node is coordinate structure. In general, each filler of type $*\texttt{T}*$ is coindexed with only one empty element. However, a filler of type $*\texttt{T}*$ can be coindexed with several empty elements if the empty elements are included in coordinate structure. This is the reason why our parser uses removeCRD. Figure 8 and 9 give examples of resolution for type $*\texttt{T}*$.

The empty elements $[*\texttt{T}*]_{\texttt{-NONE-A}}$ are handled by an exceptional process. When ATTACH action is applied to a state $(\langle s_m, \ldots, s_0\rangle, i)$ such that $\texttt{cat}(s_0) = \texttt{PRN}$, the parser coindexes the empty element $x = [*\texttt{T}*]_{\texttt{-NONE-A}}$ included in s_0 with s_1. More precisely, the coindexation is executed if the following conditions hold:

- $x \in \texttt{des}(s_0)$
- $\texttt{match}(s_1, x)$
- $\texttt{free}(x, \langle s_m, \ldots, s_0\rangle)$

For the other types of nonlocal dependencies, that is, $*\texttt{EXP}*$, $*\texttt{ICH}*$ and $*\texttt{RNR}*$, we use a simi-

```
Rule: *EXP*-R
  PRECONDITION
    ACTION=ATTACH
    $s_0$ is a filler of type *EXP*
  CONSTRAINT for coindexed element $x$
    $x \in \bigcup_{j=1}^{m} \texttt{ch}(s_j)$      # $x$ c-commands $s_0$
    $x = [[\texttt{it}]_{\texttt{PRP}}]_{\texttt{NP}}$
  SELECT: RIGHTMOST

Rule: *ICH*-L
  PRECONDITION
    ACTION=E-SHIFT(-NONE-L, *ICH*)
  CONSTRAINT for coindexed element $x$
    $\texttt{match}(x, e)$
    $\texttt{free}(x, \langle s_m, \ldots, s_0 \rangle)$
  SELECT: RIGHTMOST

Rule: *ICH*-R
  PRECONDITION
    ACTION=ATTACH
    $s_0$ is a filler of type *ICH*
  CONSTRAINT for coindexed element $x$
    $x \in \bigcup_{j=1}^{m} \texttt{des}(s_j)$
    $\texttt{match}(s_0, x)$
    $\texttt{free}(x, \langle s_m, \ldots, s_0 \rangle)$
  SELECT: RIGHTMOST

Rule: *RNR*-R
  PRECONDITION
    ACTION=ATTACH
    $s_0$ is a filler of type *RNR*
  CONSTRAINT for coindexed element $x$
    $x \in \texttt{des}(s_1)$      # $s_0$ c-commands $x$
    $\texttt{match}(s_0, x)$
    $\texttt{free}(x, \langle s_m, \ldots, s_0 \rangle)$
  SELECT: ALL
```

Figure 10: Resolution rule for *EXP*, *ICH* and *RNR*.

lar idea to design the resolution rules. Figure 10 shows the resolution rules.

These heuristic resolution rules are similar to the previous work (Campbell, 2004; Kato and Matsubara, 2015), which also utilizes c-command relation. An important difference is that we design heuristic rules not for fully-connected parse tree but for stack of parse trees derived by left-corner parsing. That is, the extend version of c-command relation plays an important role on our heuristic rules.

5 Parsing Strategy

We use a beam-search decoding with the structured perceptron (Collins, 2002). A transition action a for a state S has a score defined as follows:

$$score(S, a) = \mathbf{w} \cdot \mathbf{f}(S, a)$$

where $\mathbf{f}(S, a)$ is the feature vector for the state-action pair (S, a), and $\mathbf{w}$ is a weight vector. The

```
input: sentence $w_1 \cdots w_n$, beam size $k$
$H \leftarrow \{S_0\}$   # $S_0$ is the initial state for $w_1 \cdots w_0$
repeat $N$ times do
  $C \leftarrow \{\}$
  for each $S \in H$ do
    for each possible action $a$ do
      $S' \leftarrow$ apply $a$ to $S$
      push $S'$ to $C$
  $H \leftarrow k$ best states of $C$
return best final state in $C$
```

Figure 11: Beam-search parsing.

score of a state S' which is obtained by applying an action a to a state S is defined as follows:

$$score(S') = score(S) + score(S, a)$$

For the initial state S_0, $score(S_0) = 0$.

We learn the weight vector $\mathbf{w}$ by max-violation method (Huang et al., 2012) and average the weight vector to avoid overfitting the training data (Collins, 2002).

In our method, action sequences for the same sentence have different number of actions because of E-SHIFT action. To absorb the difference, we use an IDLE action, which are proposed in (Zhu et al., 2013):

$$\text{IDLE} : (\langle [\cdots]_{\texttt{TOP}} \rangle, n) \Rightarrow (\langle [\cdots]_{\texttt{TOP}} \rangle, n)$$

Figure 11 shows details of our beam-search parsing. The algorithm is the same as the previous transition-based parsing with structured perceptron. One point to be noted here is how to determine the maximum length of action sequence ($= N$) which the parser allows. Since it is impossible to know in advance how many empty elements a parse tree has, we need to set this parameter as a sufficiently larger value.

5.1 Features

A feature is defined as the concatenation of a transition action and a state feature which is extracted using a feature template. Table 3 shows our baseline feature templates. The feature templates are similar to the ones of (Zhang and Clark, 2009), which are standardly used as baseline templates for transition-based constituent parsing. Here, b_i and s_i stand for the i-th element of buffer and stack, respectively. $x.\texttt{c}$ represents x's augmented label. $x.\texttt{l}$, $x.\texttt{r}$ and $x.\texttt{h}$ represent x's leftmost, rightmost and head children. $x.\texttt{t}$ and $x.\texttt{w}$ represents x's head POS tag and head word, respectively. $x.\texttt{i}$ indicates whether or not the initial letter of x is capitalized. When a non-terminal node

type	feature templates
unigram	$s_0.\texttt{c} \circ s_0.\texttt{t}$, $s_0.\texttt{c} \circ s_0.\texttt{w}$,
	$s_0.\texttt{l.c} \circ s_0.\texttt{l.w}$, $s_0.\texttt{r.c} \circ s_0.\texttt{r.w}$, $s_0.\texttt{h.c} \circ s_0.\texttt{h.w}$,
	$s_1.\texttt{c} \circ s_1.\texttt{t}$, $s_1.\texttt{c} \circ s_1.\texttt{w}$,
	$s_1.\texttt{l.c} \circ s_1.\texttt{l.w}$, $s_1.\texttt{r.c} \circ s_1.\texttt{r.w}$, $s_1.\texttt{h.c} \circ s_1.\texttt{h.w}$,
	$s_2.\texttt{c} \circ s_2.\texttt{t}$, $s_2.\texttt{c} \circ s_2.\texttt{w}$, $s_3.\texttt{c} \circ s_3.\texttt{t}$, $s_3.\texttt{c} \circ s_3.\texttt{w}$,
	$b_0.\texttt{i}$, $b_0.\texttt{w}$, $b_1.\texttt{i}$, $b_1.\texttt{w}$, $b_2.\texttt{i}$, $b_2.\texttt{w}$, $b_3.\texttt{i}$, $b_3.\texttt{w}$
bigram	$s_1.\texttt{w} \circ s_0.\texttt{w}$, $s_1.\texttt{c} \circ s_0.\texttt{w}$, $s_1.\texttt{w} \circ s_0.\texttt{c}$, $s_1.\texttt{w} \circ s_0.\texttt{w}$,
	$s_0.\texttt{c} \circ b_0.\texttt{i}$, $s_0.\texttt{c} \circ b_0.\texttt{w}$, $s_0.\texttt{w} \circ b_0.\texttt{i}$, $s_0.\texttt{w} \circ b_0.\texttt{w}$,
	$s_1.\texttt{c} \circ b_0.\texttt{i}$, $s_1.\texttt{c} \circ b_0.\texttt{w}$, $s_1.\texttt{w} \circ b_0.\texttt{i}$, $s_1.\texttt{w} \circ b_0.\texttt{w}$,
	$b_0.\texttt{i} \circ b_1.\texttt{i}$, $b_0.\texttt{w} \circ b_1.\texttt{i}$, $b_0.\texttt{i} \circ b_1.\texttt{w}$, $b_0.\texttt{w} \circ b_1.\texttt{w}$
trigram	$s_2.\texttt{c} \circ s_1.\texttt{c} \circ s_0.\texttt{c}$, $s_2.\texttt{c} \circ s_1.\texttt{c} \circ s_0.\texttt{w}$,
	$s_2.\texttt{c} \circ s_1.\texttt{w} \circ s_0.\texttt{c}$, $s_2.\texttt{w} \circ s_1.\texttt{c} \circ s_0.\texttt{c}$,
	$s_1.\texttt{c} \circ s_0.\texttt{c} \circ b_0.\texttt{i}$, $s_1.\texttt{w} \circ s_0.\texttt{c} \circ b_0.\texttt{i}$,
	$s_1.\texttt{c} \circ s_0.\texttt{w} \circ b_0.\texttt{i}$, $s_1.\texttt{w} \circ s_0.\texttt{w} \circ b_0.\texttt{i}$

Table 3: Baseline feature templates.

feature templates
$s_0.\texttt{n}_0.\texttt{c}$, $s_0.\texttt{n}_1.\texttt{c}$, $s_1.\texttt{n}_0.\texttt{c}$, $s_1.\texttt{n}_1.\texttt{c}$,
$rest_2.\texttt{n}_0.\texttt{c}$, $rest_2.\texttt{n}_1.\texttt{c}$

Table 4: Nonlocal dependency feature templates.

type	system	F_1
TS	Zhu et al. (2013) (beam 16)	90.4
	Zhu et al. (2013)* (beam 16)	91.3
	Mi and Huang (2015) (beam 32)	90.3
	Mi and Huang (2015) (beam 32,DP)	90.8
	Thang et al. (2015) (A*)	91.1
	Ballesteros and Carreras (2015) (beam 64)	89.0
NDI	Charniak (2000)† (post-processing)	89.6
	Dienes and Dubey (2003a) (in-processing)	86.4
	Schmid (2006) (in-processing)	86.6
	Kato and Matsubara (2015) (in-processing)	87.7
ours	CF (beam 64)	88.9
	baseline features (beam 64)	89.0
	baseline + ND features (beam 64)	88.9

TS: transition-based parsers with structured perceptron.
NDI: parsers with nonlocal dependency identification.
DP: Dynamic Programming.
Zhu et al. (2013)* uses additional language resources.
†Johnson (2002) and Campbell (2004) used the output of Charniak's parser.

Table 5: Comparison for constituent parsing.

does not yet have a head child, the head-based atomic features are set to a special symbol nil. To extract the features, we need to identify head children in parse trees. We use the head rules described in (Surdeanu et al., 2008).

In addition to these features, we introduce a new feature which is related to empty element resolution. When a transition action invokes empty element resolution described in section 4.3.2, we use as a feature, whether or not the procedure coindexes empty elements with a filler. Such a feature is difficult for a PCFG to capture. This feature enables our parsing model to learn the resolution rule preferences implicitly, while the training process is performed only with oracle action sequences.

In addition, we use features about free empty elements and fillers. Table 4 summarizes such feature templates. Here, $x.\texttt{n}_i$ stands for the i-th rightmost free element included in x, and $rest_i$ stands for the stack $\langle s_m, \ldots, s_i \rangle$.

6 Experiment

We conducted an experiment to evaluate the performance of our parser using the Penn Treebank. We used a standard setting, that is, section 02-21 is for the training data, section 22 is for the development data and section 23 is for the test data.

In training, we set the beam size k to 16 to achieve a good efficiency. We determined the optimal iteration number of perceptron training, and the beam size (k was set to 16, 32 and 64) for decoding on the development data. The maximum length of action sequences ($= N$) was set to $7n$, where n is the length of input sentence. This maximum length was determined to deal with the sentences in the training data.

Table 5 presents the constituent parsing performances of our system and previous systems. We used the labeled bracketing metric PARSEVAL (Black et al., 1991). Here, "CF" is the parser which was learned from the training data where nonlocal dependencies are removed. This result demonstrates that our nonlocal dependency identification does not have a bad influence on constituent parsing. From the viewpoint of transition-based constituent parsing, our left-corner parser is somewhat inferior to other perceptron-based shift-reduce parsers. On the other hand, our parser outperforms the parsers which identify nonlocal dependency based on in-processing approach.

We use the metric proposed by Johnson (2002) to evaluate the accuracy of nonlocal dependency identification. Johnson's metric represents a nonlocal dependency as a tuple which consists of the type of the empty element, the category of the empty element, the position of the empty element, the category of the filler and the position of the filler. For example, the nonlocal dependency of the type $*\texttt{T}*$ in Figure 1 is represented as $(*\texttt{T}*, \texttt{NP}, [4, 4], \texttt{WHNP}, [3, 4])$. The precision and the recall are measured using these tuples. For more details, see (Johnson, 2002).

Table 6 shows the nonlocal dependency identification performances of our method and previous methods. Previous in-processing approach

	rec.	pre.	F_1	Unindexed empty elements are excluded rec.	pre.	F_1
Johnson (2002) (post-processing)	63	73	68	–	–	–
D & D (2003a) (pre-processing)	66.0	80.5	72.6	–	–	–
D & D (2003b) (in-processing)	68.7	81.5	74.6	–	–	–
Campbell (2004) (post-processing)	75.1	78.3	76.7	–	–	–
Schmid (2006) (in-processing)	–	–	–	73.5	81.7	77.4
K&M (2015) (in-processing)	75.6	80.6	78.0	73.6	80.3	76.8
baseline features	70.4	79.7	74.8	65.4	81.1	72.4
+ ND features	75.5	81.4	78.4	73.8	79.8	76.7

Table 6: Comparison for nonlocal dependency identification.

achieved the state-of-the-art performance of nonlocal dependency identification, while it was inferior in terms of constituent parsing accuracy. Our nonlocal dependency identification is competitive with previous in-processing approach, and its accuracy of constituent parsing is higher than previous in-processing approach. As a whole, our parser achieves a good balance between constituent parsing and nonlocal dependency identification. Table 7 summarizes the accuracy of nonlocal dependency identification for each type of nonlocal dependency.

7 Conclusion

This paper proposed a transition-based parser which identifies nonlocal dependencies. Our parser achieves a good balance between constituent parsing and nonlocal dependency identification. In the experiment reported in this paper, we used simple features which are captured by nonlocal dependencies. In future work, we will develop lexical features which are captured by nonlocal dependencies.

Acknowledgements

This research was partially supported by the Grant-in-Aid for Scientific Research (B) (No.26280082) of JSPS.

(F, E, T)	freq.	pre.	rec.	F_1
$(\texttt{NP}, \texttt{NP}, *)$	1146	76.7	75.4	76.1
$(-, \texttt{-NONE-}, \texttt{0})$	545	92.3	83.7	87.8
$(\texttt{WHNP}, \texttt{NP}, *\texttt{T}*)$	507	88.0	84.0	86.0
$(-, \texttt{NP}, *)$	477	69.0	71.7	70.3
$(-, \texttt{-NONE-}, *\texttt{U}*)$	388	98.4	93.6	95.9
$(\texttt{S}, \texttt{S}, *\texttt{T}*)$	277	83.6	80.9	82.2
$(\texttt{WHADVP}, \texttt{ADVP}, *\texttt{T}*)$	164	82.1	70.1	75.7
$(-, \texttt{WHNP}, \texttt{0})$	107	73.3	51.4	60.4
$(-, \texttt{WHADVP}, \texttt{0})$	36	80.8	58.3	67.7
$(\texttt{PP}, \texttt{PP}, *\texttt{ICH}*)$	29	20.0	3.5	5.9
$(\texttt{WHPP}, \texttt{PP}, *\texttt{T}*)$	22	84.2	72.7	78.1
$(\texttt{SBAR}, \texttt{SBAR}, *\texttt{EXP}*)$	16	71.4	31.3	43.5
$(\texttt{S}, \texttt{S}, *\texttt{ICH}*)$	15	36.4	26.7	30.8
$(\texttt{S}, \texttt{S}, *\texttt{EXP}*)$	14	50.0	42.9	46.2
$(\texttt{SBAR}, \texttt{SBAR}, *\texttt{ICH}*)$	12	0.0	0.0	0.0
$(-, \texttt{NP}, *?*)$	11	0.0	0.0	0.0
$(-, \texttt{S}, *?*)$	9	100.0	11.1	20.0
$(-, \texttt{VP}, *?*)$	8	45.5	62.5	52.6
$(\texttt{VP}, \texttt{VP}, *\texttt{T}*)$	8	40.0	25.0	30.8
$(\texttt{ADVP}, \texttt{ADVP}, *\texttt{T}*)$	7	80.0	57.1	66.7
$(\texttt{PP}, \texttt{PP}, *\texttt{T}*)$	7	80.0	57.1	66.7
$(-, \texttt{-NONE-}, *?*)$	7	0.0	0.0	0.0
$(\texttt{ADJP}, \texttt{ADJP}, *\texttt{T}*)$	6	66.7	33.3	44.4
$(\texttt{ADVP}, \texttt{ADVP}, *\texttt{ICH}*)$	6	0.0	0.0	0.0
$(\texttt{NP}, \texttt{NP}, *\texttt{ICH}*)$	6	0.0	0.0	0.0
$(\texttt{VP}, \texttt{VP}, *\texttt{ICH}*)$	6	0.0	0.0	0.0

Table 7: Accuracy of nonlocal dependency identification for all types of nonlocal dependency that occurred more than 5 times in section 23 of the Penn Treebank. F, E and T give the category of its filler, the category of its empty element and the type of nonlocal dependency, respectively.

References

Miguel Ballesteros and Xavier Carreras. 2015. Transition-based spinal parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 289–299, Beijing, China, July.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. *Bracketing Guidelines for Treebank II Style Penn Treebank Project*. University of Pennsylvania.

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the 4th DARPA Speech and Natural Language Workshop*, pages 306–311.

Shu Cai, David Chiang, and Yoav Goldberg. 2011. Language-independent parsing with empty elements. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 212–

216, Portland, Oregon, USA, June.

Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 645–652, Barcelona, Spain, July.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics*, pages 132–139, April.

Noam Chomsky. 1981. *Lectures on government and binding: The Pisa lectures*. Walter de Gruyter.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, July.

Péter Dienes and Amit Dubey. 2003a. Antecedent recovery: Experiments with a trace tagger. In Michael Collins and Mark Steedman, editors, *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 33–40, July.

Péter Dienes and Amit Dubey. 2003b. Deep syntactic processing by combining shallow methods. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 431–438, Sapporo, Japan, July.

Kilian Evang and Laura Kallmeyer. 2011. Plcfrs parsing of english discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland, October.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 24–31, Edmonton, Canada.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada, June.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 136–143, Philadelphia, Pennsylvania, USA, July.

Yoshihide Kato and Shigeki Matsubara. 2015. Identifying nonlocal dependencies in incremental parsing. *IEICE Transactions on Information and Systems*, E98-D(4):994–998.

Roger Levy and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 327–334, Barcelona, Spain, July.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China, July.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):310–330.

Haitao Mi and Liang Huang. 2015. Shift-reduce constituency parsing with dynamic programming and pos tag lattice. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1030–1035, Denver, Colorado, May–June.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, British Columbia, October.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 691–698, Sydney, Australia, July.

Helmut Schmid. 2006. Trace prediction and recovery with unlexicalized PCFGs and slash features. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 177–184, Sydney, Australia, July.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The conll 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England, August.

Shunsuke Takeno, Masaaki Nagata, and Kazuhide Yamamoto. 2015. Empty category detection using path features and distributed case frames. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1335–1340, Lisbon, Portugal, September.

Le Quang Thang, Hiroshi Noji, and Yusuke Miyao. 2015. Optimal shift-reduce constituent parsing with structured perceptron. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint*

Conference on Natural Language Processing (Volume 1: Long Papers), pages 1534–1544, Beijing, China, July.

Zhiguo Wang and Nianwen Xue. 2014. Joint POS tagging and transition-based constituent parsing in chinese with non-local features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 733–742, Baltimore, Maryland, June.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1169–1179, Beijing, China, July.

Bing Xiang, Xiaoqiang Luo, and Bowen Zhou. 2013. Enlisting the ghost: Modeling empty categories for machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 822–831, Sofia, Bulgaria, August.

Nianwen Xue and Yaqin Yang. 2013. Dependency-based empty category detection via phrase structure trees. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1051–1060, Atlanta, Georgia, June.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, France, October.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria, August.