Left-Corner Parsing for Identifying PTB-Style Nonlocal Dependencies

Yoshihide Kato† and Shigeki Matsubara††

Nonlocal dependencies represent syntactic phenomena such as wh-movement, A-movement in passives, topicalization, raising, control, and right node raising. Nonlocal dependencies play an important role in semantic interpretation. This paper proposes a left-corner parser that identifies nonlocal dependencies. Our parser integrates nonlocal dependency identification into a transition-based system. We adopt a left-corner strategy in order to use the syntactic relation c-command, which plays an important role in nonlocal dependency identification. To utilize the global features captured by nonlocal dependencies, our parser uses a structured perceptron. In experimental evaluations, our parser achieved a good balance between constituent parsing and nonlocal dependency identification.

Key Words: Transition-based Parsing, Structured Perceptron, C-command Relation

1 Introduction

Many constituent parsers based on the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993) are available, but most of them cannot handle nonlocal dependencies, which represent syntactic phenomena such as wh-movement, A-movement in passives, topicalization, raising, control, and right node raising. Nonlocal dependencies play an important role in semantic interpretation. In the Penn Treebank, a nonlocal dependency is represented as a pair comprising an empty element and a filler.

Several methods of nonlocal dependency identification have been proposed to date.¹ These methods can be divided into three approaches: pre-processing approaches (Dienes and Dubey 2003a, 2003b), in-processing approaches (Dienes and Dubey 2003b; Schmid 2006; Kato and Matsubara 2015), and post-processing approaches (Johnson 2002; Levy and Manning 2004; Campbell 2004). In the pre-processing approach, empty elements are detected by a tagger called “trace tagger”, which uses only the surface word information. The in-processing approach integrates nonlocal dependency identification into a parser, which ranks candidate parse trees by a

¹ Information & Communications, Nagoya University
² Graduate School of Information Science, Nagoya University
³ Several methods of empty element detection, a subtask of nonlocal dependency identification, also have been proposed (Cai, Chiang, and Goldberg 2011; Xue and Yang 2013; Xiang, Luo, and Zhou 2013; Takeno, Nagata, and Yamamoto 2015; Hayashi and Nagata 2016).
probabilistic context-free grammar. The post-processing approach recovers the nonlocal dependencies from a parser output lacking these nonlocal dependencies.

The parsing models of previous methods cannot use the global features captured by nonlocal dependencies. Pre- and in-processing approaches adopt a probabilistic context-free grammar, which does not easily incorporate global features. The post-processing approach performs constituent parsing and nonlocal dependency identification separately. Consequently, the constituent parser cannot use the information of nonlocal dependencies.

This paper proposes a parser that integrates nonlocal dependency identification into constituent parsing. Our method adopts an in-processing approach, but without a probabilistic context-free grammar. Our parser is based on a transition system with a 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 \textit{c-command}, which plays an important role in nonlocal dependency identification. In previous transition-based constituent parsers (Sagae and Lavie 2005, 2006; Zhang and Clark 2009; Zhu, Zhang, Chen, Zhang, and Zhu 2013; Wang and Xue 2014; Mi and Huang 2015; Thang, Noji, and Miyao 2015; Watanabe and Sumita 2015; Ballesteros and Carreras 2015; Hayashi, Suzuki, and Nagata 2016; Cross and Huang 2016), the c-command relations were difficult to capture in the parsing process.

Our contributions are summarized below:

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

(2) We extend the c-command relation to deal with nodes in \textit{parse tree stack} in the transition system. Based on this extended relation, we develop heuristic rules for co-indexing the empty elements with their fillers.

(3) We introduce new nonlocal dependency features into our parsing model.

The remainder of this paper is organized as follows: Section 2 explains the representation of nonlocal dependencies in the Penn Treebank, and Section 3 describes our transition-based left-corner parser. Section 4 introduces nonlocal dependency identification into our parser. Section 5 describes the structured perceptron and features. Section 6 reports our experimental results, demonstrating the good balance between constituent parsing and nonlocal dependency identification achieved by our parser. Section 7 concludes the paper.

\footnote{A preliminary version of this paper was presented at the 54th Annual Meeting of the Association for Computational Linguistics (ACL2016) by the same authors (Kato and Matsubara 2016).}
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 a (partial) parse tree with several nonlocal dependencies, extracted from the Penn Treebank. The nodes labeled with -NONE- are empty elements. The terminal symbols such as * and *T* represent different types of nonlocal dependency (an unexpressed subject of a to-infinitive and a trace of wh-movement, respectively). When the terminal symbol of an empty element is indexed, its filler (with the same number) exists in the parse tree. For example, the corresponding filler of *T*-1 is the node WHNP-1. Table 1 briefly describes the empty elements quoted from the annotation guideline (Bies, Ferguson, Katz, and MacIntyre 1995). More details are provided in the guideline.

![Diagram of a parse tree in the Penn Treebank]

**Fig. 1** A parse tree in the Penn Treebank

| type | description | n-posi |
|------|-------------|--------|
| *    | arbitrary PRO, controlled PRO and trace of A-movement (e.g., I want * to have a trip.) | L, R, – |
| *EXP* | expletive (extraposition) (e.g., It *EXP* is difficult to do so.) | R |
| *ICH* | interpret constituent here (discontinuous dependency) (e.g., He started a venture *ICH* with his father, that seemed promising.) | L, R |
| *RNR* | right node raising (e.g., The value is less than *RNR* or equal to *RNR* 100.) | R |
| *T*   | trace of A'-movement (e.g., What does he buy *T*?) | A, L |
| 0     | null complementizer (e.g., I believe 0 she is clever.) | – |
| *U*   | unit (e.g., C$ 5 *U*) | – |
| *?*   | placeholder for ellipsed material (e.g., John is cleverer than they are *?*.) | – |
| *NOT* | anti-placeholder in template gapping | – |
3 Transition-Based Left-Corner Parsing

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

The parsing actions of our parser are similar to those described in Henderson (2003). Here we define an action as a state-to-state mapping. As in previous work (Sagae and Lavie 2005, 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; Maier 2015; Ballesteros and Carreras 2015; Hayashi et al. 2016; Cross and Huang 2016), 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 \((\sigma, i)\), where \(\sigma\) is the stack and \(i\) is the position of the next input word in the buffer. The initial state is \((\langle \rangle, 0)\). The final states are in the form of \((\langle \cdots \rangle_{\text{TOP}}, n)\), where \(\text{TOP}\) is a special symbol denoting the root of the parse tree and \(n\) is the length of the input sentence. Below, the notation \([\alpha]_X\) represents a node with label \(X\) and children \(\alpha\). If the node is non-terminal, \(\alpha\) is a list of nodes. Otherwise, \(\alpha\) is a terminal symbol. Our parser performs the following transition actions:

- **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 simultaneously perform POS tagging and constituent parsing. This is in the same way as Wang and Xue (2014).
- **LeftCorner**\(\{H/\emptyset\}\)(\(X\)): pop up the first node from the stack, attach it with a new parent node labeled with \(X\) 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 to the second 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.

The Attach action is similar to the standard Reduce action used in previous transition-based parsers. However, there is an important difference between Attach and Reduce. The Reduce action cannot handle nodes with more than two children. For this reason, the previous work converts parse trees into binarized ones. This conversion makes it difficult to capture the hierarchical structure of the parse trees. On the other hand, the Attach action can handle more than two children. Therefore, our parser does not require such kind of tree binarization. The transition actions in our parser are summarized in Figure 2.

To guarantee that every non-terminal node has exactly one head child, our parser imposes
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 that derive the parse tree shown in Figure 1. Superscript \(^*\) indicates the head children.

Although most of the transition-based constituent parsers do not handle nonlocal dependencies, there are two exceptions: one developed by Maier (2015), the other by Hayashi and Nagata (2016). Maier 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. As this annotation style cannot deal with one-to-many correspondences between fillers and empty elements, the parser cannot identify any nonlocal dependency in which a filler is co-indexed with several empty elements.\(^3\) Hayashi and Nagata integrated empty element detection into a transition-based spinal parser. However, although their parser detects empty elements, it cannot find their corresponding fillers.

![Fig. 2 Transition actions for left-corner parsing](image)

**Table 2** Example of a transition action sequence

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

\(^3\) Furthermore, the annotation style is unsuitable for nonlocal dependencies of type \(*\). In Evang and Kallmeyer (2011), the PTB-style annotation of types \(*\text{EXP*}\), \(*\text{ICH*}\), \(*\text{RNR*}\), and \(*\text{T*}\) is transformed into an annotation with crossing branches by detaching each filler from a parse tree and reattaching it to the location of the corresponding empty element. If the transformation is applied to nonlocal dependencies of type \(*\), a problem arises. For example, in Figure 1, the information on the subject of “managed” is lost by detaching **NP-SBJ-2**.
4 Nonlocal Dependency Identification

Nonlocal dependency identification consists of two subtasks:
• empty element detection
• empty element resolution, which co-indexes empty elements with their fillers.

For empty element detection, our parser inserts empty elements at arbitrary positions similarly to in-processing approach. Our method co-indexes empty elements with their fillers using simple heuristic rules developed for our transition system.

4.1 Empty Element Detection

We introduce the following action to deal with empty elements:

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

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

4.2 Annotations

For empty element resolution, we augment the Penn Treebank. For the nonlocal dependency types *EXP*, *ICH*, *RNR*, and *T*, we assign the following information to each filler and each empty element:

• The nonlocal dependency type (assigned to filler only).
• The nonlocal dependency category, defined as the category of the parent of the empty element.
• The relative position of the filler, which takes a value from \{A, L, R\}. The relative position is “A” if the filler is an ancestor of the empty element, or “L” (“R”) if the filler occurs to the left (right) of the empty element. Table 1 summarizes the allowed value for each empty element.

This information is utilized for co-indexing empty elements with fillers. Below, we write \(n\text{-}\text{type}(x)\), \(n\text{-}\text{cat}(x)\), and \(n\text{-}\text{posi}(x)\) for the information of a node \(x\), respectively.

If an empty element of type * is indexed, it is annotated in the same way.\(^4\) Furthermore, every empty element co-indexed to a constituent with no function tag SBJ is assigned a tag OBJCTRL.\(^5\)

\(^4\) We omit its nonlocal dependency category, as it is always NP.
\(^5\) In the Penn Treebank, every subject has the tag SBJ.
This assignment enables our parser to distinguish between subject control and object control.

Figure 3 shows the augmented 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 fillers in many cases. For example, all of the empty elements in Figure 1 are c-commanded by their fillers. Our method co-indexes the empty elements with their fillers by simple heuristic rules based on the c-command relation.

4.3.1 C-command Relation

Here, we define the 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 $z$.

Furthermore, we extend the 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:

- Let $S = ((s_m, \ldots, s_0), i)$ be a parser state.
  
  1. Let $y$ be a node $s_j(0 \leq j \leq m)$. If $x$ is a child of some node $s_k(j < k \leq m)$, then $x$ c-commands $y$ in any final state derived from $S$. The other nodes in the stack do not c-command $y$ in any final state derived from $S$.
  
  2. Let $y$ be a proper descendant of $s_j(0 \leq j \leq m)$. If $x$ is a child of some node $s_k(j < k \leq m)$, then $x$ c-commands $y$ in any final state derived from $S$. Except
$s_{j-1}$, none of the other nodes c-command $y$ in any final state derived from $S$. If $s_{j-1}$ c-commands $y$ in some final state $S'$ derived from $S$, then $s_{j-1}$ is attached to $s_j$ immediately after all children of $s_{j-1}$ are instantiated in the transition from $S$ to $S'$.

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

As an example, consider the state shown in Figure 4. The nodes are instantiated in order of their subscripts. According to the above statement, the nodes in the dotted box c-command the shifted node \texttt{-NONE-L14}. In the parse tree shown in Figure 3, which is derived from this state, these nodes c-command \texttt{-NONE-L14} under the original definition.

Previous transition-based constituent parsers cannot easily capture the c-command relations between nodes in a stack. We first consider the case of shift-promote-adjoin constituent parsing (Cross and Huang 2016). The parser combines the top two nodes in a stack using two kinds of adjoin actions. One action attaches the first node to the second node as a child, the other attaches the second node to the first node as a child. This means that the dominance relations (and consequently the c-command relations) between the nodes in the stack cannot be determined until the nodes are actually combined. The same problem arises in transition-based spinal parsing (Ballesteros and Carreras 2015; Hayashi et al. 2016). On the other hand, such problem does not occur in transition-based left-corner parsing, because it is guaranteed that any node $s_j (0 < j \leq m)$ dominates $s_{j-1}$ in any final state derived from $S = (s_m, \ldots, s_0, i)$.

We next consider the case of shift-reduce constituent parsing (Sagae and Lavie 2005, 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). Shift-reduce parsers combine the top two nodes in a stack using the reduce action, which generates a new node and attaches the two nodes as children to the generated node. As these parsers cannot handle nodes with more than two children, the parse trees are converted into binarized trees by introducing \textit{temporary} nodes, which are collapsed and removed when detransforming the binarized trees into the original ones. As an example, we focus

![Fig. 4 Example of resolution of $[e]_{\text{store-L}}$](image)

378
on node $s_1$ in a stack $\langle s_2, s_1, s_0 \rangle$. After two consecutive reduce actions on the stack, we obtain a tree of the form $[s_2[s_1s_0]_X]_Y$. If $X$ is temporary, the original form of the parse tree is $[s_2s_1s_0]_Y$, and $s_1$ c-commands the descendants of $s_2$. If $X$ is not temporary, $s_1$ does not c-command the descendants of $s_2$. Therefore, the c-command relation between nodes depends on whether the generated node is temporary or non-temporary. Furthermore, it is notable that we must recognize $s_0$ to determine whether $s_1$ c-commands the descendants of $s_2$. However, in left-corner parsing, whether or not $s_1$ c-commands the descendants of $s_2$ can be determined without recognizing $s_0$.

The corresponding transitions in left-corner parsing are as follows:

- $((s_2), i) \Rightarrow ([s_2]_Y, i) \Rightarrow^* (([s_2]_Y, s_1), i') \Rightarrow ([s_2s_1]_Y, i')$
- $((s_2), i) \Rightarrow ([s_2]_Y, i) \Rightarrow^* (([s_2]_Y, s_1), i') \Rightarrow ([s_2]_Y, [s_1]_X, i')$

The first transition corresponds to the case of temporary $X$. Clearly, $s_1$ c-commands the descendants of $s_2$. In the second transition, the statement described in the first paragraph of this section guarantees that $s_1$ does not c-command the descendants of $s_2$. In both transitions, $s_0$ is not yet recognized.

### 4.3.2 Resolution Rules

When executing an E-SHIFT or ATTACH, our parser co-indexes an empty element with its filler. The E-SHIFT action co-indexes the shifted empty element $e$ such that $n\text{-posi}(e) = L$ with its filler. The ATTACH action co-indexes the attached filler $s_0$ such that $n\text{-posi}(s_0) = R$ with its corresponding empty element. The 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 the co-indexed element must satisfy. SELECT takes two values, ALL and RIGHTMOST. When several elements satisfy CONSTRAINT, SELECT determines how to select co-indexed elements. ALL means that all of the elements satisfying the CONSTRAINT are co-indexed, and RIGHTMOST selects the rightmost element satisfying the CONSTRAINT.

The most frequent type of nonlocal dependency in the Penn Treebank is $. The resolution rules of this type are presented in Figure 5. Here, $\text{ch}(s)$ designates the set of children of $s$. $\text{sbj}(x)$ means that $x$ has a function tag SBJ. $\text{par}(x)$ and $\text{cat}(x)$ represent the parent and constituent category of $x$, respectively. $\text{des}(s)$ designates the set of proper descendants of $s$, and $\text{free}(x, \sigma)$ denotes that $x$ is not co-indexed with any node in $\sigma$.

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

The second rule \(*\text{-L-OBJCTRL}\) 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 co-indexes \(e\) with the child noun phrase of \(x\) rather than \(x\) itself, to maintain the PTB-style annotation.

The third rule \(*\text{-R}\) is for null subject of a participial clause. Figure 6 demonstrates the application of rule \(*\text{-R}\) to a state. This rule is applied when the transition action is \text{ATTACH} and

---

**Rule: \(*\text{-L}\)**

**Precondition**

- \(\text{ACTION} = \text{E-SHIFT}(-\text{NONE-L}, *)\)
- Constraint for co-indexed element \(x\)
  - \(x \in \bigcup_{j=0}^{m} \text{ch}(s_j)\)
  - \(\# x\) c-commands \(e\)
  - \(\text{sbj}(x)\)
- \(\text{SELECT: RIGHTMOST}\)

**Rule: \(*\text{-L-OBJCTRL}\)**

**Precondition**

- \(\text{ACTION} = \text{E-SHIFT}(-\text{NONE-L-OBJCTRL}, *)\)
- Constraint for co-indexed element \(x\)
  - \(x \in \bigcup_{j=0}^{m} \text{ch}(s_j)\)
  - \(\# x\) c-commands \(e\)
  - \(\text{cat}(x) = \text{NP} \lor \text{cat}(x) = \text{PP}\)
  - \(\text{cat}(\text{par}(x)) = \text{VP}\)
- \(\text{SELECT: RIGHTMOST}\)

---

**Fig. 5**

Resolution rules for type *

---

**Fig. 6**

Example of resolution of \([\ast]\text{-NONE-R}\)
$s_0$ 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 applied to a state when the E-SHIFT action inserts an empty element of type $^{*}T^*$. Here, $\text{match}(x, y)$ checks the consistency between $x$ and $y$; that is, $\text{match}(x, y)$ holds if and only if $\text{n-type}(x) = \text{n-type}(y)$, $\text{n-cat}(x) = \text{n-cat}(y)$, $\text{n-posi}(x) = \text{n-posi}(y)$, $\text{cat}(x) \neq -$NONE- and $\text{cat}(y) = -$NONE-.

$\text{removeCRD}((s_m, \ldots, s_0))$ is a stack which is obtained by removing $s_j (0 \leq j \leq m)$ which is annotated with a tag CRD. The tag CRD denotes that the node is a coordinate structure. In general, each filler of type $^{*}T^*$ is co-indexed with a single empty element. However, a type $^{*}T^*$ filler can be co-indexed with several empty elements if those empty elements are included in a coordinate structure. This is the reason why our parser uses $\text{removeCRD}$. Figures 8 and 9 give examples of resolution for type $^{*}T^*$.

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

**Rule: $^{*}T^*-$L**

**Precondition**

$\text{ACTION= E-SHIFT(-NONE-L, }^{*}T^*)$

**Constraint for co-indexed element $x$**

$x \in \bigcup_{e \in \text{removeCRD}((s_m, \ldots, s_0))} \text{ch}(s)$

$\# x$ c-commands $e$

$\text{match}(x, e)$

$\text{free}(x, \text{removeCRD}((s_m, \ldots, s_0)))$

**Select: Rightmost**

**Fig. 7** Resolution rule for type $^{*}T^*$

**Fig. 8** Example of resolution of $[^{*}T^*]_{-NONE-NP-L}$

---

6 The tag CRD is assigned to any node matching the pattern $[\ldots]X[\ldots](CC|CC|JP|,|:)\ldots[X\ldots]$. 
Fig. 9 Example of resolution of $[^T*]_{none-np-l}$ when the stack has a coordinate structure

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

For the other types of nonlocal dependencies, that is, $^*\text{EXP*}$, $^*\text{ICH*}$, and $^*\text{RNR*}$, we use a similar idea to design the resolution rules. Figure 10 shows the resolution rules.

These heuristic resolution rules are similar to those described in the previous work (Campbell 2004; Kato and Matsubara 2015), which also utilize the c-command relation. However, whereas previous rules were designed for fully-connected parse trees, our heuristic rules apply to stacks of parse trees derived by left-corner parsing. That is, the extended c-command relation plays an important role in our heuristic rules.

5 Parsing with a Structured Perceptron

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:

$$score(S, a) = w \cdot f(S, a),$$
where \( f(S, a) \) is the feature vector for the state–action pair \((S, a)\), and \( w \) is a weight vector. The score of a state \( S' \) obtained by applying action \( a \) to state \( S \) is given by:

\[
score(S') = score(S) + score(S, a).
\]

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

We learn the weight vector \( w \) by the max-violation method (Huang, Fayong, and Guo 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 this difference, we use the IDLE action proposed in Zhu et al. (2013):

\[
\text{IDLE} : ([\cdots]_{\text{TDP}}, n) \Rightarrow ([\cdots]_{\text{TDP}}, n).
\]

Figure 11 shows the details of our beam-search parsing. The algorithm is identical to the previous transition-based parsing with structured perceptron. However, here we must determine the maximum length \( N \) of the action sequence allowed by the parser. As the number of empty elements in a parse tree cannot be known in advance, this parameter must be set to a sufficiently large value.
5.1 Features

A feature is defined as the concatenation of a transition action and a state feature extracted using a feature template. Our baseline feature templates are presented in Table 3. These templates are similar to those of Zhang and Clark (2009), which are standardly used as baseline templates for transition-based constituent parsing. Here, \( b_i \) and \( s_i \) denote the \( i \)-th elements of the buffer and stack, respectively. \( x.c \) represents the augmented label of \( x \). \( x.l, x.r, \) and \( x.h \) represent the leftmost, rightmost, and head children of \( x \), respectively. \( x.t \) and \( x.w \) represent the head POS tag and head word of \( x \), respectively. \( x.i \) indicates whether or not the initial letter of \( x \) is capitalized. When a non-terminal node has no head child, its head-based atomic features are set to a special symbol \( \text{nil} \). To extract the features, we need to identify the head children in the parse trees. For this purpose, we apply the head rules described in Surdeanu, Johansson, Meyers, Márquez, and Nivre (2008).

In addition to these features, we introduce a new feature related to empty element resolution. When a transition action invokes the empty element resolution described in Section 4.3.2, we use as a feature, whether or not the procedure co-indexes an empty element with a filler. Such a feature is not easily captured by probabilistic context-free grammars. This feature enables implicit learning of the resolution rule preferences, while the training process is performed only with oracle action sequences.

We also use the free empty elements and fillers as features. These feature templates are summarized in Table 4. Here, \( x.n_i \) denotes the \( i \)-th rightmost free element in \( x \), and \( \text{rest}_i \) is the stack \( \langle s_m, \ldots, s_i \rangle \).
respectively.

6 Experiment

We experimentally evaluated the performance of our parser on Penn Treebank data. As the training, development and test data, we used sections 02–21, section 22 and section 23, respectively.

In the training phase, we set the beam size \( k \) to 16 to achieve high 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 \( N \) of the action sequences was set to \( 7n \), where \( n \) is the length of the 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, evaluated by the labeled bracketing metric PARSEVAL (Black, Abney, Flickenger, Gdaniec, Grishman, Harrison, Hindle, Ingria, Jelinek, Klavans, Liberman, Marcus, Roukos, Santorini, and Strzalkowski 1991). Here, “CF” is the parser learned from the training data without the nonlocal dependencies. By removing these dependencies, we confirmed that our nonlocal dependency

| Table 3 | Baseline feature templates |
|---------|---------------------------|
| type    | feature templates         |
| unigram | \( s_0.c \circ s_0.t, s_0.c \circ s_0.w, \) |
|         | \( s_0.l.c \circ s_0.l.w, s_0.r.c \circ s_0.r.w, s_0.h.c \circ s_0.h.w, \) |
|         | \( s_1.c \circ s_1.t, s_1.c \circ s_1.w, \) |
|         | \( s_1.l.c \circ s_1.l.w, s_1.r.c \circ s_1.r.w, s_1.h.c \circ s_1.h.w, \) |
|         | \( s_2.c \circ s_2.t, s_2.c \circ s_2.w, s_2.c \circ s_3.t, s_2.c \circ s_3.w, \) |
|         | \( b_0.i, b_0.w, b_1.i, b_1.w, b_2.i, b_2.w, b_3.i, b_3.w \) |
| bigram  | \( s_1.w \circ s_0.w, s_1.c \circ s_0.w, s_1.w \circ s_0.c, s_1.w \circ s_0.w, \) |
|         | \( s_0.c \circ b_0.i, s_0.c \circ b_0.w, s_0.w \circ b_0.i, s_0.w \circ b_0.w, \) |
|         | \( s_1.c \circ b_0.i, s_1.c \circ b_0.w, s_1.w \circ b_0.i, s_1.w \circ b_0.w, \) |
|         | \( b_0.i \circ b_1.i, b_0.w \circ b_1.i, b_0.i \circ b_1.w, b_0.w \circ b_1.w \) |
| trigram | \( s_2.c \circ s_1.c \circ s_0.c, s_2.c \circ s_1.c \circ s_0.w, \) |
|         | \( s_2.c \circ s_1.w \circ s_0.c, s_2.w \circ s_1.c \circ s_0.c, \) |
|         | \( s_1.c \circ s_0.c \circ b_0.i, s_1.w \circ s_0.c \circ b_0.i, \) |
|         | \( s_1.c \circ s_0.w \circ b_0.i, s_1.w \circ s_0.w \circ b_0.i \) |

| Table 4 | Nonlocal dependency feature templates |
|---------|----------------------------------------|
| feature templates |
| \( s_0.n_0.c, s_0.n_1.c, s_1.n_0.c, s_1.n_1.c, \) |
| \( rest_2.n_0.c, rest_2.n_1.c \) |

6 Experiment

We experimentally evaluated the performance of our parser on Penn Treebank data. As the training, development and test data, we used sections 02–21, section 22 and section 23, respectively.

In the training phase, we set the beam size \( k \) to 16 to achieve high 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 \( N \) of the action sequences was set to \( 7n \), where \( n \) is the length of the 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, evaluated by the labeled bracketing metric PARSEVAL (Black, Abney, Flickenger, Gdaniec, Grishman, Harrison, Hindle, Ingria, Jelinek, Klavans, Liberman, Marcus, Roukos, Santorini, and Strzalkowski 1991). Here, “CF” is the parser learned from the training data without the nonlocal dependencies. By removing these dependencies, we confirmed that our nonlocal dependency
Table 5  Constituent parsing performances of our system and previous systems

| 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  |
|        | (Hayashi et al. 2016) (beam 32, DP)                                    | 90.9  |
| NDI    | (Johnson 2002; Campbell 2004)$†$ (post-processing)                     | 89.6  |
|        | (Dienes and Dubey 2003a) (pre-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)* used additional language resources.
Johnson (2002) and Campbell (2004)$†$ used the output of Charniak’s (2000) parser.

Identification does not degrade the constituent parsing. In transition-based constituent parsing, our left-corner parser is outperformed by other perceptron-based shift-reduce parsers. On the other hand, our parser outperforms parsers that identify nonlocal dependencies by pre- and in-processing approaches.

We compared the parsing speeds (number of sentences parsed per second) of our parser and its CF version. Our parser was implemented in CMUCL\textsuperscript{7} with a Xeon 3.47 GHz CPU. The parsing speed was measured on the test data. Our parser (1.22 sent./sec.) was slower than its CF version (1.87 sent./sec.). We suspect that this is due to E-SHIFT action which increases the number of derived states.

The accuracy of nonlocal dependency identification was evaluated by the metric proposed in Johnson (2002). Johnson’s metric represents each nonlocal dependency as a tuple comprising the type, category and position of the empty element, and the category and position of the filler. For example, the type $*$T*$ nonlocal dependency in Figure 1 is represented as ($*$T*$, NP, [4,4], WHNP, [3,4]$). The precision and recall are measured using these tuples. For more

\textsuperscript{7} https://www.cons.org/cmucl/
details, see Johnson (2002).

Table 6 shows the nonlocal dependency identification performances of our method and previous methods. Previous in-processing approaches identified the nonlocal dependencies with state-of-the-art performance, but their constituent parsing accuracies were inferior. In nonlocal dependency identification, our method fairly competed with previous in-processing approaches, while achieving more accurate constituent parsing. Overall, our parser achieved a good balance between constituent parsing and nonlocal dependency identification. Table 7 summarizes the accuracies of different types of nonlocal dependencies identified by our parser.

Although the methods based on post-processing yield higher constituent parsing accuracy than our parser, their nonlocal dependency identifications are less accurate than ours, partly because they do not reflect the nonlocal dependency features. This is an intrinsic drawback of post-processing approaches. Previous pre- and in-processing approaches yield relatively low constituent parsing accuracies, probably because parsing models based on these approaches are probabilistic context-free grammars, which cannot easily capture the global features. Finally, we compared our method with that proposed by Kato and Matsubara (2015), which is similar to ours in many respects. Both methods augment the Penn Treebank with nonlocal dependency information, and both employ a transition-based parser. The main difference between them is that Kato and Matsubara’s parser uses a probabilistic context-free grammar. From the viewpoint of transition-based parsing, it has another drawback, that is, it cannot use look-ahead features. Look-ahead features are useful not only in constituent parsing but also in nonlocal dependency identification. For example, if the look-ahead word is “to,” it is a strong indicator that an empty element * should probably be inserted. Our parser uses such kinds of features as shown in

| Method                                      | rec. | pre. | $F_1$ | Unindexed empty elements are excluded | rec.  | pre.  | $F_1$ |
|---------------------------------------------|------|------|-------|--------------------------------------|-------|-------|-------|
| (Johnson 2002) (post-processing)            | 63   | 73   | 68    |                                      | —     | —     | —     |
| (Dienes and Dubey 2003b) (pre-processing)  | 66.0 | 80.5 | 72.6  |                                      | —     | —     | —     |
| (Dienes and Dubey 2003a) (pre-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  |
| (Kato and Matsubara 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 7  Accuracy of nonlocal dependency identification by our parser

| (F, E, T)               | freq. | pre. | rec. | F1  |
|-------------------------|-------|------|------|-----|
| (NP, NP, ∗)             | 1,146 | 76.7 | 75.4 | 76.1|
| (−, -NONE-, 0)          | 545   | 92.3 | 83.7 | 87.8|
| (WHNP, NP, ∗T+)         | 507   | 88.0 | 84.0 | 86.0|
| (−, NP, ∗)              | 477   | 69.0 | 71.7 | 70.3|
| (−, -NONE-, ∗U+)        | 388   | 98.4 | 93.6 | 95.9|
| (S, S, ∗T+)             | 277   | 83.6 | 80.9 | 82.2|
| (WHADVP, ADVP, ∗T+)     | 164   | 82.1 | 70.1 | 75.7|
| (−, WHNP, 0)            | 107   | 73.3 | 51.4 | 60.4|
| (−, WHADVP, 0)          | 36    | 80.8 | 58.3 | 67.7|
| (PP, PP, ∗ICH+)         | 29    | 20.0 | 3.5  | 5.9 |
| (WHPP, PP, ∗T+)         | 22    | 84.2 | 72.7 | 78.1|
| (SBAR, SBAR, ∗EXP+)     | 16    | 71.4 | 31.3 | 43.5|
| (S, S, ∗ICH+)           | 15    | 36.4 | 26.7 | 30.8|
| (S, S, ∗EXP+)           | 14    | 50.0 | 42.9 | 46.2|
| (SBAR, SBAR, ∗ICH+)     | 12    | 0.0  | 0.0  | 0.0 |
| (−, NP, ∗?∗)            | 11    | 0.0  | 0.0  | 0.0 |
| (−, S, ∗?∗)             | 9     | 100.0| 11.1 | 20.0|
| (−, VP, ∗?∗)            | 8     | 45.5 | 62.5 | 52.6|
| (VP, VP, ∗T+)           | 8     | 40.0 | 25.0 | 30.8|
| (ADVP, ADVP, ∗T+)       | 7     | 80.0 | 57.1 | 66.7|
| (PP, PP, ∗T+)           | 7     | 80.0 | 57.1 | 66.7|
| (−, -NONE-, ??∗)        | 7     | 0.0  | 0.0  | 0.0 |
| (ADJP, ADJP, ∗T+)       | 6     | 66.7 | 33.3 | 44.4|
| (ADVP, ADVP, ∗ICH+)     | 6     | 0.0  | 0.0  | 0.0 |
| (NP, NP, ∗ICH+)         | 6     | 0.0  | 0.0  | 0.0 |
| (VP, VP, ∗ICH+)         | 6     | 0.0  | 0.0  | 0.0 |

Evaluation was performed on all nonlocal dependencies occurring more than 5 times in section 23 of the Penn Treebank. F, E and T denote the filler category, empty element category, and type of nonlocal dependency, respectively.

Section 5.1.

Finally, to separately evaluate the empty element resolution rules proposed in Section 4.3.2, we measured the precision and recall of nonlocal dependency identification when the oracle action sequences were given. The results are listed in Table 8. These results confirm the high accuracy of our resolution rules.

---

8 Note that the oracle action sequences insert empty elements correctly.
Table 8  Accuracy of nonlocal dependency identification by our parser when the oracle action sequences were given.

| (F, E, T)                        | pre. | rec. | F₁   |
|----------------------------------|------|------|------|
| (NP, NP, *)                      | 99.1 | 98.9 | 99.0 |
| (WHNP, NP, *T*)                  | 100.0| 99.8 | 99.9 |
| (S, S, *T*)                      | 99.6 | 95.7 | 97.6 |
| (WHADVP, ADVP, *T*)              | 100.0| 100.0| 100.0|
| (PP, PP, *ICH*)                  | 100.0| 100.0| 100.0|
| (WHPP, PP, *T*)                  | 100.0| 100.0| 100.0|
| (SBAR, SBAR, *EXP*)              | 100.0| 87.5 | 93.3 |
| (S, S, *ICH*)                    | 100.0| 100.0| 100.0|
| (S, S, *EXP*)                    | 100.0| 100.0| 100.0|
| (SBAR, SBAR, *ICH*)              | 100.0| 100.0| 100.0|
| (VP, VP, *T*)                    | 100.0| 100.0| 100.0|
| (ADVP, ADVP, *T*)                | 100.0| 100.0| 100.0|
| (PP, PP, *T*)                    | 100.0| 85.7 | 92.3 |
| (ADJP, ADJP, *T*)                | 100.0| 100.0| 100.0|
| (ADVP, ADVP, *ICH*)              | 100.0| 100.0| 100.0|
| (NP, NP, *ICH*)                  | 100.0| 100.0| 100.0|
| (VP, VP, *ICH*)                  | 100.0| 100.0| 100.0|
| overall                          | 99.5 | 98.7 | 99.1 |

7 Conclusion

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

Another important issue is to compare our work with deep parsers based on different grammar formalisms, such as the CCG-based parser (Clark and Curran 2007), the HPSG-based parser (Miyao and Tsujii 2008) and the LFG-based parser (Cahill, Burke, O’Donovan, Riezler, van Genabith, and Way 2008). Because these parsers are not directly comparable to ours, the evaluation will largely follow that of Miyao and Tsujii (2004), who evaluated deep parsers on PropBank data (Palmer, Gildea, and Kingsbury 2005).
Acknowledgement

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

Reference

Ballesteros, M. and Carreras, X. (2015). “Transition-based Spinal Parsing.” In Proceedings of the 19th Conference on Computational Natural Language Learning, pp. 289–299.

Bies, A., Ferguson, M., Katz, K., and MacIntyre, R. (1995). Bracketing Guidelines for Treebank II Style Penn Treebank Project. University of Pennsylvania.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). “A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars.” In Proceedings of the 4th DARPA Speech and Natural Language Workshop, pp. 306–311.

Cahill, A., Burke, M., O’Donovan, R., Riezler, S., van Genabith, J., and Way, A. (2008). “Wide-coverage Deep Statistical Parsing using Automatic Dependency Structure Annotation.” Computational Linguistics, 34 (1), pp. 81–124.

Cai, S., Chiang, D., and Goldberg, Y. (2011). “Language-Independent Parsing with Empty Elements.” In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp. 212–216.

Campbell, R. (2004). “Using Linguistic Principles to Recover Empty Categories.” In Proceedings of the 42nd Meeting of the Association for Computational Linguistics, Main Volume, pp. 645–652.

Charniak, E. (2000). “A Maximum-Entropy-Inspired Parser.” In Proceedings of the 1st North American Chapter of the Association for Computational Linguistics, pp. 132–139.

Chomsky, N. (1981). Lectures on Government and Binding: The Pisa Lectures. Walter de Gruyter.

Clark, S. and Curran, J. R. (2007). “Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models.” Computational Linguistics, 33 (4), pp. 493–552.

Collins, M. (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, pp. 1–8.
Cross, J. and Huang, L. (2016). “Incremental Parsing with Minimal Features Using Bi-Directional LSTM.” In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 32–37.

Dienes, P. and Dubey, A. (2003a). “Antecedent Recovery: Experiments with a Trace Tagger.” In Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing, pp. 33–40.

Dienes, P. and Dubey, A. (2003b). “Deep Syntactic Processing by Combining Shallow Methods.” In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, pp. 431–438.

Evang, K. and Kallmeyer, L. (2011). “PLCFRS Parsing of English Discontinuous Constituents.” In Proceedings of the 12th International Conference on Parsing Technologies, pp. 104–116.

Hayashi, K. and Nagata, M. (2016). “Empty Element Recovery by Spinal Parser Operations.” In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 95–100.

Hayashi, K., Suzuki, J., and Nagata, M. (2016). “Shift-reduce Spinal TAG Parsing with Dynamic Programming.” Transactions of the Japanese Society for Artificial Intelligence, 31 (2), pp. J-F83.1–8.

Henderson, J. (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, pp. 24–31.

Huang, L., Fayong, S., and Guo, Y. (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, pp. 142–151.

Johnson, M. (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, pp. 136–143.

Kato, Y. and Matsubara, S. (2015). “Identifying Nonlocal Dependencies in Incremental Parsing.” IEICE Transactions on Information and Systems, E98-D (4), pp. 994–998.

Kato, Y. and Matsubara, S. (2016). “Transition-Based Left-Corner Parsing for Identifying PTB-Style Nonlocal Dependencies.” In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 930–940.

Levy, R. and Manning, C. (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, Main Volume, pp. 327–334.
Maier, W. (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)*, pp. 1202–1212.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). “Building a Large Annotated Corpus of English: The Penn Treebank.” *Computational Linguistics, 19* (2), pp. 310–330.

Mi, H. and Huang, L. (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*, pp. 1030–1035.

Miyao, Y. and Tsujii, J. (2004). “Deep Linguistic Analysis for the Accurate Identification of Predicate-Argument Relations.” In *Proceedings of the 20th International Conference on Computational Linguistics*, pp. 1392–1398.

Miyao, Y. and Tsujii, J. (2008). “Feature Forest models for Probabilistic HPSG Parsing.” *Computational Linguistics, 34* (1), pp. 35–80.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). “The Proposition Bank: An Annotated Corpus of Semantic Roles.” *Computational linguistics, 31* (1), pp. 71–106.

Sagae, K. and Lavie, A. (2005). “A Classifier-Based Parser with Linear Run-Time Complexity.” In *Proceedings of the 9th International Workshop on Parsing Technology*, pp. 125–132.

Sagae, K. and Lavie, A. (2006). “A Best-First Probabilistic Shift-Reduce Parser.” In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pp. 691–698.

Schmid, H. (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*, pp. 177–184.

Surdeanu, M., Johansson, R., Meyers, A., Márquez, L., and Nivre, J. (2008). “The CoNLL 2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies.” In *Proceedings of the 12th Conference on Computational Natural Language Learning*, pp. 159–177.

Takeno, S., Nagata, M., and Yamamoto, K. (2015). “Empty Category Detection using Path Features and Distributed Case Frames.” In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1335–1340.

Thang, L. Q., Noji, H., and Miyao, Y. (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)*, pp. 1534–1544.

Wang, Z. and Xue, N. (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), pp. 733–742.

Watanabe, T. and Sumita, E. (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), pp. 1169–1179.

Xiang, B., Luo, X., and Zhou, B. (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), pp. 822–831.

Xue, N. and Yang, Y. (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, pp. 1051–1060.

Zhang, Y. and Clark, S. (2009). “Transition-Based Parsing of the Chinese Treebank using a Global Discriminative Model.” In Proceedings of the 11th International Conference on Parsing Technologies, pp. 162–171.

Zhu, M., Zhang, Y., Chen, W., Zhang, M., and Zhu, J. (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), pp. 434–443.

**Yoshihide Kato:** received the B.E. degree, the M.E. degree, and the Dr. of Engineering degree in information engineering from Nagoya University, in 1997, 1999, and 2003, respectively. He was an Assistant Professor in Graduate School of International Development, Nagoya University, from 2003 to 2009. He was a Researcher at Information Technology Center, Nagoya University, from 2009 to September 2011, and a Designated Assistant Professor, from October 2011 to March 2012. He is currently an Associate Professor at Information & Communications, Nagoya University. His research interests include natural language processing and information retrieval. He is a member of the IPSJ, the IEICE, and the ACL.

**Shigeki Matsubara:** received the B.E. degree in electrical and computer engineering from Nagoya Institute of Technology in 1993, and the M.E. degree, and the Dr. of Engineering from Nagoya University in 1995 and 1998, respectively. After becoming an Assistant Professor at Nagoya University, he became an
Associate Professor at Information Technology Center in 2002, and he is currently an Associate Professor in Graduate School of Information Science. His research interests include natural language processing, information retrieval and digital library. He is a member of the IPSJ, the JSAI, and the IEICE.

(Received September 9, 2016)
(Revised December 2, 2016)
(Accepted January 27, 2017)