IEICE TRANS. INF. & SYST., VOL.E98–D, NO.4 APRIL 2015

LETTER

Identifying Nonlocal Dependencies in Incremental Parsing

Yoshihide KATO[†a)], ***Member and*** **Shigeki MATSUBARA**[††], ***Senior Member***

SUMMARY This paper describes a method of identifying nonlocal dependencies in incremental parsing. Our incremental parser inserts empty elements at arbitrary positions to generate partial parse trees including empty elements. To identify the correspondence between empty elements and their fillers, our method adapts a hybrid approach: slash feature annotation and heuristic rules. This decreases local ambiguity in incremental parsing and improves the accuracy of our parser.

key words: *incremental parsing, nonlocal dependency, empty element, slash feature, c-command*

1. Introduction

Nonlocal dependency represents several kinds of syntactic phenomena such as wh-movement, A-movement in passives, raising, control and so on. Information about nonlocal dependency plays an important role in semantic interpretation. Chung and Gildea [1] have reported an experimental result where empty element information improved machine translation results.

This paper investigates a method of identifying nonlocal dependencies in incremental parsing. Incremental parsers analyze an input sentence from left to right and generate partial parse trees, each of which spans an initial fragment. They are useful to realize real-time spoken language processing systems, such as simultaneous machine interpretation systems, automatic subtitle generating systems and so on [2], [3]. Moreover, incremental parsers can analyze unfinished sentences such as "He was appointed CEO, succeeding ..." Our incremental parser inserts empty elements at arbitrary positions to generate partial parse trees including empty elements. Our method identifies the correspondence between empty elements and their fillers on the basis of the slash feature annotation described in [4]–[6]. However, our method does not assign slash features for some kind of nonlocal dependency since it increases local ambiguity in incremental parsing. To identify nonlocal dependencies for which slash features are not assigned, our method uses heuristic rules. This hybrid approach improves the accuracy of our incremental parser.

Manuscript received September 18, 2014.
Manuscript revised December 5, 2014.
Manuscript publicized January 13, 2015.

[†]The author is with Information & Communications, Nagoya University, Nagoya-shi, 464–8601 Japan.
[††]The author is with the Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8603 Japan.
a) E-mail: yoshihide@icts.nagoya-u.ac.jp

DOI: 10.1587/transinf.2014EDL8186

2. Nonlocal Dependencies and Their Identification

This section gives an overview of previous work on nonlocal dependency identification.

2.1 Nonlocal Dependency

We explain nonlocal dependency on the basis of Penn Treebank annotation [7]. Each nonlocal dependency is represented as a pair of an *empty element* and its *filler*. Figure 1 shows an example of Penn Treebank style tree. The label $-\texttt{NONE}-$ means that the node is an empty element. The labels such as $*\texttt{T}*$ and $*$ represent the type of the empty element: $*\texttt{T}*$ is used as a trace of wh-movement. $*$ is used as a trace of A-movement and an unexpressed subject. When an empty element is indexed, there exists its filler in the parse tree. The filler has the same number. For example, $*-1$ means that the node $\texttt{NP}-1$ is the corresponding filler. The dotted arrows in the figure represent the correspondences between the empty elements and the fillers.

2.2 Nonlocal Dependency Identification

Many syntactic parsers based on the Penn Treebank are available, but they generate parse trees which do not include nonlocal dependencies. We call this kind of parse tree a *CF tree*. Several methods of identifying nonlocal dependencies have been proposed so far. These methods can be divided

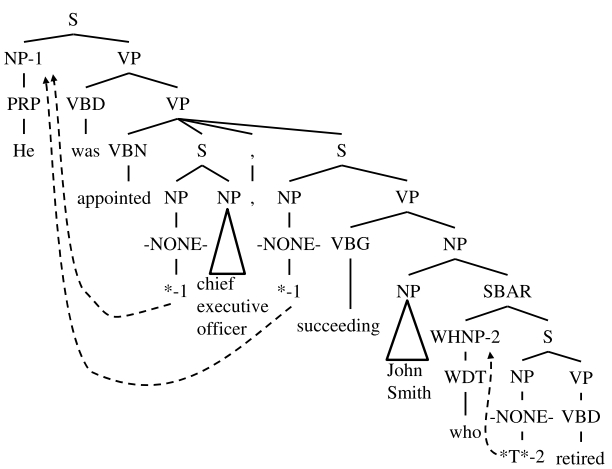

Fig. 1 A Penn Treebank style tree.

Copyright © 2015 The Institute of Electronics, Information and Communication Engineers

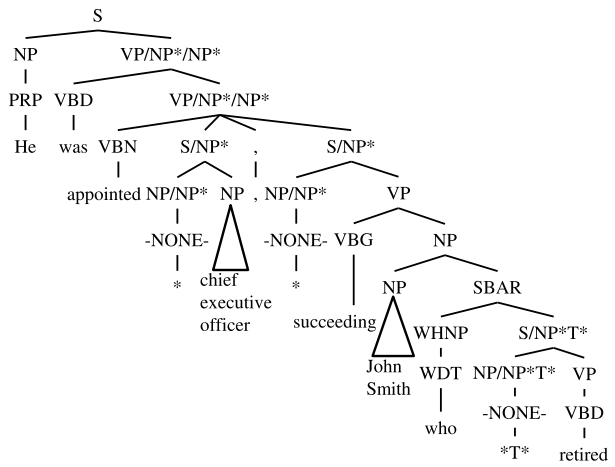

Fig. 2 A tree with slash feature annotation.

into the following:

- Recovering nonlocal dependencies from a CF tree which is generated by a parser.
- Integrating nonlocal dependency identification into a parser.

The former approach receives a CF tree as input and returns the parse tree including nonlocal dependencies. This is called a post-processing approach. Johnson [8] has proposed a method of recovering nonlocal dependencies from a CF tree based on pattern matching. In this method, pattern trees each of which recovers a nonlocal dependency are extracted from the Penn Treebank. Levy and Manning [9] use several classifiers to recover nonlocal dependencies. This method applies the classifiers to each node in a CF tree and identifies nonlocal dependencies. Campbell [10] has developed linguistically-motivated rules for recovering nonlocal dependencies. The method applies the rules in top-down fashion to recover nonlocal dependencies.

In the latter approach, a parser identifies nonlocal dependencies directly. This approach is called in-processing. Dienes and Dubey [4], Dienes and Dubey [5], and Schmid [6] have proposed methods which assign slash features to the nodes on paths from empty elements to their fillers. Figure 2 shows an example of slash feature annotation[†]. Their parsers insert empty elements at arbitrary positions between words and generate a parse tree annotated with slash features. They recover nonlocal dependencies by traversing the nodes with slash features.

3. Nonlocal Dependency Identification in Incremental Parsing

This section proposes a method of identifying nonlocal dependencies in incremental parsing. First, we describe an incremental parsing method. Next, we extend the parser to generate partial parse trees including nonlocal dependency information.

[†]In this example, a node η has a slash feature if η dominates an empty element E and η is not an ancestor of the filler corresponding to E. A slash feature consists of the category of E and the type of E.

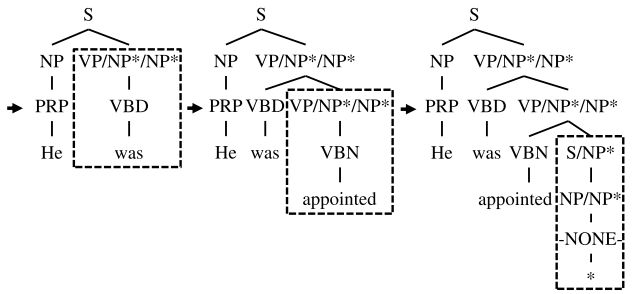

Fig. 3 A process of incremental parsing.

```
1  input: w_1 ... w_n
2  H_0 ← {S}
3  for i ← 1 to n do
4     H_i ← {}
5     while H_{i-1} ≠ {} do
6        pop σ from H_{i-1}
7        push all elements in Combine(σ, w_i) to H_i
8        push all elements in Combine(σ, ε) to H_{i-1}
9  return H_n[0]      # best scoring parse tree
```

Fig. 4 Incremental parsing with inserting empty elements.

3.1 Incremental Parser

Incremental parsers analyze a sentence from left to right, and generate partial parse trees for each initial fragment. The partial parse trees connect all words in each initial fragment of the sentence. Collins and Roark [11] and Kato and Matsubara [12] have proposed incremental parsing methods in which the parsing process proceeds on a word-by-word basis by using *allowable chains*. An allowable chain is a sequence of nonterminal symbols followed by a terminal symbol. Each allowable chain corresponds to a label sequence on a path from a node to its leftmost descendant leaf. Figure 3 illustrates an incremental parsing process. The allowable chains are specified by the dotted boxes. This example illustrates that the incremental parsing method can generate partial parse trees which span each initial fragment.

3.2 Nonlocal Dependency Identification

To identify nonlocal dependencies in an incremental parsing process, we adapt in-processing approach. Post-processing approach is not suitable for incremental processing.

3.2.1 Inserting Empty Elements

Our incremental parser is similar to the one used for previous work in the literatures [11]–[13]. We extend the parser to deal with empty elements. Our approach is the same as the in-processing approach. That is, our incremental parser inserts empty elements at arbitrary positions between words. Figure 4 shows our incremental parsing algorithm with inserting empty elements. H_i is a priority queue, which keeps

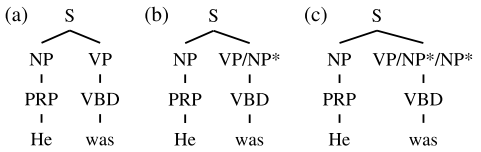

Fig. 5 Local ambiguity caused by slash feature annotation.

partial parse trees for $w_1 \ldots w_i$. Each partial parse tree has a probability score and each priority queue is sorted by the score. To improve the efficiency of the parsing, we adapt a beam search strategy[†]. *Combine*(σ, t) is a set of the results of combining a partial parse tree σ and a terminal symbol t using some allowable chains. ε is an arbitrary empty element terminal symbol.

The parser simply attaches the allowable chains for empty elements with partial parse trees and pushes the results into the priority queue. All we need is to add the line 8 in incremental parsing procedure. This enables the parser to generate partial parse trees including empty elements.

3.2.2 Slash Feature Annotation and Local Ambiguity

Our proposed method uses slash feature annotation in the similar manner to the one used for the previous work. However, our method does not assign slash features for nonlocal dependencies of type $*$, because this increases local ambiguity in incremental parsing. As an example, let us consider the incremental parsing process shown in Fig. 3 again. The partial parse tree for "He was" has two slash features. This represents that at least two empty elements of type $*$ exist in the right context as shown in Fig. 2. On the other hand, to parse sentences such as "He was a chief executive officer." and "He$_i$ was appointed $*_i$ chief executive officer.", the parser needs to generate partial parse trees (a) and (b) shown in Fig. 5, respectively. The ambiguity caused by slash feature annotation of type $*$ cannot be resolved at this point since the decision of a correct partial parse tree depends on the following input. If we remove slash features of type $*$, the ambiguity does not arise.

Removing slash features of type $*$ may have a negative effect since slash features help the parser detect empty elements. To avoid this problem, we introduce another kind of annotation which does not cause local ambiguity. We call this annotation *PRO feature*. For each empty NP of type $*$ and its parent whose category is PP or S, we assign a tag PRO to the node. See Fig. 6.

3.2.3 Identifying Nonlocal Dependencies

Our incremental parser identifies nonlocal dependencies when an empty element E or a filler candidate F is instantiated in a partial parse tree σ. If the type of the instantiated element is not $*$, the parser searches σ for the corresponding element by using slash features. The procedure is the same as the one in the literature [4]. When the parser finds no corresponding element, it assumes that a corresponding element exists in the right context which is not constructed at this point.

For nonlocal dependencies of type $*$, our method uses heuristic rules since slash features are removed. When an empty element E of type $*$ is instantiated in a partial parse tree σ, the parser selects the nearest node F which satisfies the following conditions:

- F c-commands E.
- F is a subject, or the object of an object-control verb.

When a complement noun phrase F is instantiated in a partial parse tree σ, the parser searches σ for the empty element E satisfying the following conditions:

- The type of E is $*$.
- E is c-commanded by F.
- E does not have the corresponding filler.

All the nodes satisfying these conditions are co-indexed with F.

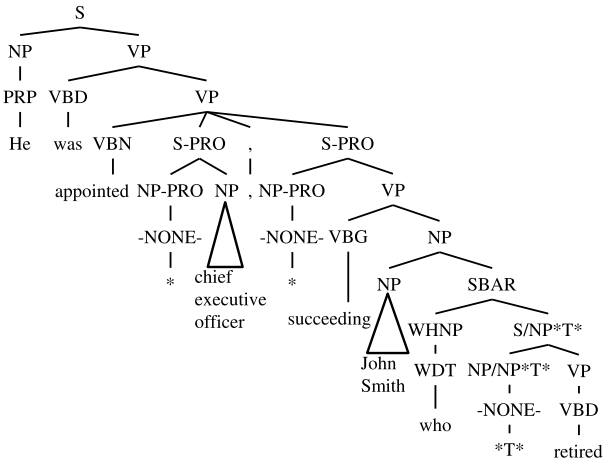

Fig. 6 PRO feature annotation.

4. An Experiment

To evaluate our proposed method, we conducted a parsing experiment. We used the metric proposed by Johnson [8] to evaluate the performance of identifying nonlocal dependencies. 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. The metric measures the precision and the recall using these tuples. For more details, see the literature [8].

We implemented a probabilistic incremental parser described in Sect. 3. The probabilistic model and the grammar were learned from the parse trees in section 02-21 of the Wall Street Journal in Penn Treebank [7]. We marked heads

[†]We use the same strategy proposed by the literature [13], that is, our parser exits the while-loop when $P(H_{i-1}[0]) < 10^{-11} P(H_i[0])|H_i|^3$ holds. Here, $P(\cdot)$ represents a probability. See the literature [13] for more details.

Table 1 Parsing results.

features			Labeled bracketing			Nonlocal dependency identification		
remove slash feature of type $*$	PRO	OC	recall(%)	precision(%)	f-score(%)	recall(%)	precision(%)	f-score(%)
			87.1 (87.2)	87.7 (87.8)	87.4 (87.5)	73.4	78.8	76.0
		√	87.2 (87.2)	87.8 (87.7)	87.5 (87.4)	73.2	78.6	75.8
	√		87.4 (87.2)	87.9 (87.8)	87.6 (87.5)	74.8	80.1	77.4
	√	√	87.4 (87.2)	87.9 (87.7)	87.7 (87.5)	74.6	80.2	77.3
√			87.4 (87.5)	87.9 (88.0)	87.7 (87.7)	71.9	76.9	74.3
√		√	87.3 (87.5)	87.8 (88.0)	87.6 (87.8)	72.0	77.1	74.4
√	√		87.5 (87.5)	88.0 (88.0)	87.8 (87.8)	75.3	80.1	77.6
√	√	√	87.5 (87.6)	88.0 (88.1)	87.7 (87.9)	75.6	80.6	78.0

Table 2 Comparison with previous work.

				Unindexed empty elements are excluded.		
	rec.(%)	prec.(%)	f(%)	rec.(%)	prec.(%)	f(%)
Johnson [8]	63	73	68	–	–	–
D & D [4]	66.0	80.5	72.6	–	–	–
D & D [5]	68.7	81.5	74.6	–	–	–
Campbell [10]	75.1	78.3	76.7	–	–	–
Schmid [6]	–	–	–	73.5	81.7	77.4
our method	75.6	80.6	78.0	73.6	80.3	76.8

and complements using heuristic rules similar to the ones in the literature [14]. All function tags remain. Therefore, the parser can identify subjects using the function tag `SBJ`. To identify object-control verbs, we assigned a tag `OC` to the verbs whose object is a filler of type $*$. Each verb and preposition subcategorized for its complements.

By using section 23, we evaluated the accuracy of nonlocal dependency identification and labeled bracketing task. The labeled bracketing task was evaluated by the PARSEVAL metric [15]. Table 1 shows the results. The accuracies of the parser without empty element insertion are shown in brackets[†]. This result demonstrates that the empty element insertion has little negative effect. The last line shows the results of our proposed method. Our method achieved the highest recall and precision in nonlocal dependency identification. The labeled bracketing f-score is also higher. It is worth noting that worst results of nonlocal dependency identification were observed when slash features were simply removed. On the other hand, the accuracy increased considerably by using PRO features. This means that PRO feature annotation avoids a disadvantage of removing slash features. Table 2 summarizes the accuracies of previous (nonincremental) methods. This result shows that our method compares favorably with the previous ones.

5. Conclusion

This paper proposed a method of identifying nonlocal dependencies in incremental parsing. Our incremental parser can identify nonlocal dependencies at the point when an empty element and the filler are instantiated in a partial parse tree. This paper investigates only nonlocal dependency of type $*$, and the previous approach is simply adapted to the other types of nonlocal dependencies. In future work, we will explore how to deal with the other types of nonlocal dependencies in incremental parsing.

[†]All empty elements were removed from the training data after annotating features.

Acknowledgements

This research was partially supported by the Grant-in-Aid for Scientific Research (B) (No. 22300051, 26280082) of JSPS.

References

[1] T. Chung and D. Gildea, "Effects of empty categories on machine translation," Proc. 2010 Conference on Empirical Methods in Natural Language Processing, pp.636–645, Oct. 2010.
[2] J. Allen, G. Ferguson, and A. Stent, "An architecture for more realistic conversational systems," Proc. International Conference of Intelligent User Interfaces, pp.1–8, 2001.
[3] G. Aist, J. Allen, E. Campana, C.G. Gallo, S. Stoness, M. Swift, and M.K. Tanenhaus, "Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods," Proc. 11th Workshop on the Semantics and Pragmatics of Dialogue, ed. R. Artstein and L. View, pp.149–154, June 2007.
[4] P. Dienes and A. Dubey, "Antecedent recovery: Experiments with a trace tagger," Proc. 2003 Conference on Empirical Methods in Natural Language Processing, pp.33–40, July 2003.
[5] P. Dienes and A. Dubey, "Deep syntactic processing by combining shallow methods," Proc. 41st Annual Meeting of the Association for Computational Linguistics, pp.431–438, July 2003.
[6] H. Schmid, "Trace prediction and recovery with unlexicalized PCFGs and slash features," Proc. 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, pp.177–184, July 2006.
[7] M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," Computational Linguistics, vol.19, no.2, pp.310–330, 1993.
[8] M. Johnson, "A simple pattern-matching algorithm for recovering empty nodes and their antecedents," Proc. 40th Annual Meeting of the Association for Computational Linguistics, pp.136–143, July 2002.
[9] R. Levy and C. Manning, "Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation," Proc. 42nd Meeting of the Association for Computational Linguistics, Main Volume, pp.327–334, July 2004.
[10] R. Campbell, "Using linguistic principles to recover empty categories," Proc. 42nd Meeting of the Association for Computational Linguistics, Main Volume, pp.645–652, July 2004.
[11] M. Collins and B. Roark, "Incremental parsing with the perceptron algorithm," Proceedings of the 42nd Meeting of the Association for Computational Linguistics, Main Volume, pp.111–118, July 2004.
[12] Y. Kato and S. Matsubara, "Incremental parsing with adjoining operation," IEICE Trans. Inf. & Syst., vol.E92-D, no.12, pp.2306–2312,

Dec. 2009.
[13] B. Roark, "Robust garden path parsing," Natural language engineering, vol.10, no.1, pp.1–24, 2004.
[14] D.M. Bikel, "Intricacies of Collins' parsing model," Computational Linguistics, vol.30, no.4, pp.478–511, Dec. 2004.
[15] 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, "A procedure for quantitatively comparing the syntactic coverage of English grammars," Proc. 4th DARPA Speech and Natural Language Workshop, pp.306–311, 1991.