A New Representation for Span-based CCG Parsing

Yoshihide Kato and **Shigeki Matsubara**
Information & Communications, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601 Japan
yoshihide@icts.nagoya-u.ac.jp

Abstract

This paper proposes a new representation for CCG derivations. CCG derivations are represented as trees whose nodes are labeled with categories strictly restricted by CCG rule schemata. This characteristic is not suitable for span-based parsing models because they predict node labels independently. In other words, span-based models may generate invalid CCG derivations that violate the rule schemata. Our proposed representation decomposes CCG derivations into several independent pieces and prevents the span-based parsing models from violating the schemata. Our experimental result shows that an off-the-shelf span-based parser with our representation is comparable with previous CCG parsers.

1 Introduction

Combinatory Categorial Grammar (CCG) (Steedman, 2000) is a mildly context-sensitive grammar formalism. Several neural CCG parsing methods have been proposed so far (Lewis and Steedman, 2014; Xu et al., 2015; Lewis et al., 2016; Vaswani et al., 2016; Lee et al., 2016; Xu, 2016; Yoshikawa et al., 2017; Stanojević and Steedman, 2019, 2020; Bhargava and Penn, 2020; Tian et al., 2020; Prange et al., 2021; Liu et al., 2021). Currently, neural span-based models (Cross and Huang, 2016; Stern et al., 2017; Gaddy et al., 2018; Kitaev and Klein, 2018) have been successful in the field of constituency parsing. However, we cannot directly apply this technique to CCG parsing. Span-based models assume that each node label in parse trees can be predicted independently, while, in CCG, each node label (category) is strictly restricted by CCG rule schemata. The independence assumption of span-based models implies that the models are not guaranteed to generate valid CCG derivations.

To solve this problem, we propose a method of representing CCG derivations in a way suitable for span-based parsing models. Our proposed representation decomposes CCG derivations into several independent pieces and can prevent the span-based parsing models from violating the CCG rule schemata. Furthermore, as a by-product of our representation, the parsing models can assign out-of-vocabulary (OOV) categories, which have not appeared in training data. This characteristic has been attracting attention in CCG parsing research (Bhargava and Penn, 2020; Prange et al., 2021; Liu et al., 2021). Our experimental result shows that an off-the-shelf span-based parser with our representation is comparable with previous CCG parsers and can generate correct OOV categories.

$$\begin{array}{llcll} X/Y & Y|_1 Z_1 \cdots |_d Z_d & \Rightarrow & X|_1 Z_1 \cdots |_d Z_d & (>^d) \\ Y|_1 Z_1 \cdots |_d Z_d & X\backslash Y & \Rightarrow & X|_1 Z_1 \cdots |_d Z_d & (<^d) \end{array}$$

Figure 1: CCG rule schemata.

2 CCG and Span-based Parsing

This section gives an overview of Combinatory Categorial Grammar (CCG) (Steedman, 2000) and explains why we cannot directly apply the span-based approach to CCG parsing.

2.1 Combinatory Categorial Grammar

CCG represents syntactic information by basic categories (e.g., S, NP) and complex categories. Complex categories are in the form of X/Y or $X\backslash Y$, where X and Y are categories. Intuitively, each category X/Y means that it receives a category Y from its right and returns a category X. In the case of $X\backslash Y$, the direction is from its left. Formally, categories are combined using CCG rule schemata. Figure 1 shows CCG rule schemata. Here, X, Y and $Z_i (1 \leq i \leq d)$ are categories, and $|_i \in \{/, \backslash\}$. $|_1 Z_1 \cdots |_d Z_d$ is called an *argument stack* (Kuhlmann and Satta, 2014), and we use a Greek letter to represent an argument stack. For example, we use the following notation for the first

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 10579–10584
November 7–11, 2021.

rule schema:

$$X/Y \quad Y\alpha \Rightarrow X\alpha. \tag{1}$$

We define $|\alpha| = d$ and the arity of a category $Y = X\alpha$ where X is a basic category is defined as follows:

$$\texttt{arity}(Y) = |\alpha| \tag{2}$$

2.2 Span-based Parsing

A span-based parsing model (Stern et al., 2017; Gaddy et al., 2018; Kitaev and Klein, 2018) has a single scoring function $s(i, j, l)$ that scores each label l for each span (i, j). The score of a tree T is defined as follows:

$$s(T) = \sum_{(i,j,l)\in T} s(i, j, l). \tag{3}$$

The parsing problem is formulated as finding the tree T^* with the highest score:

$$T^* = \arg\max_T s(T) \tag{4}$$

and can be solved using an efficient CKY-like parsing algorithm because of the following characteristic:[1]

- The model can determine each label l for a span (i, j) independently of the other spans.

Unfortunately, CCG parsing cannot take this approach because each label (category) is strictly restricted by the CCG rule schemata. If we apply the span-based approach to CCG parsing forcibly, the following problem occurs:

- The parsing model may generate invalid CCG derivations that violate the CCG rule schemata.

3 Span-based representation

To overcome the problem described in the previous section, we propose a new representation for CCG derivations. We call the new ones *span-based representations* (*SBRs* for short), which decomposes CCG derivations into several independent pieces to prevent the span-based parsing model from violating the CCG rule schemata. Figure 2 shows an example of CCG derivation and its SBR version.

We realize span-based CCG parsing as follows:

1. Convert CCG derivations into SBRs (Section 3.2).
2. Train a span-based parsing model using SBRs and parse sentences to generate SBRs.
3. Convert the output SBRs into CCG derivations. (Section 3.3).

The basic idea behind our method is that each node label in an SBR represents a constraint on the categories of nodes in a CCG derivation. Our method recovers a CCG derivation from its SBR version by satisfying such constraints. Because constraints encoded in SBR's labels are independent, a span-based model using SBRs does not suffer from violating CCG rule schemata.

3.1 SBR's label

An SBR's label consists of the following information:

- a CCG rule schema
- a mapping from variables that occur only in the left-hand side of the rule to categories

For each node n (except leaf nodes) in a CCG derivation, its SBR version has a corresponding node. The SBR's label means that the category of n is created by the specified rule schema, and the categories of n's children satisfy the constraint represented by the mapping. For example, the label $(>^0, \texttt{Y} := \texttt{NP})$ means that the left and right children's categories are in the form of $X/\texttt{NP}$ and $\texttt{NP}$ and X is inherited from its parent's category.

3.1.1 Additional information

SBR's label cannot encode root categories of CCG derivations and unary rules. To encode this information, we introduce three types of additional information:

- $\texttt{RT}:X$ means that the category of the node n is X, if n is the root node.
- $\texttt{UL}:X$ means that the left child l is unary branching and the category of l's child is X.
- $\texttt{UR}:X$ means that the right child r is unary branching and the category of r's child is X.

We call these information *tags*.

[1] In the standard CKY algorithm, each score is kept for each pair of a span (i, j) and a label l. On the other hand, in span-based parsing, for each span (i, j), only the label with the highest score is kept. For more detail, see (Gaddy et al., 2018).

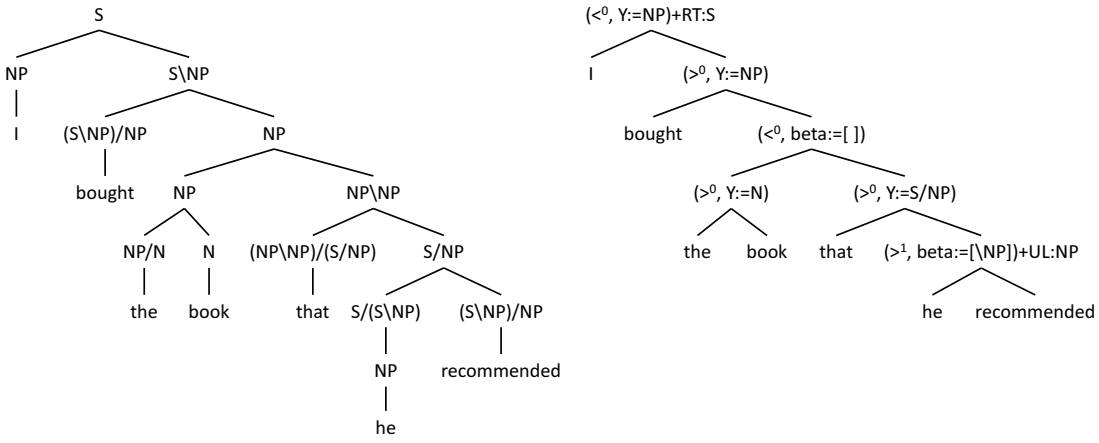

Figure 2: A CCG derivation (left) and its SBR version (right).

CCG derivation's category			SBR's label
left child L	right child R	parent P	S
X/Y	$Y\alpha$	$X\alpha$	$(>^{\lvert\alpha\rvert}, \texttt{Y} := Y)$
$Y\alpha$	$X\backslash Y$	$X\alpha$	$(<^{\lvert\alpha\rvert}, \texttt{Y} := Y)$
$X/(X\beta)$	$(X\beta)\alpha$	$X\alpha$	$(>^{\lvert\alpha\rvert}, \texttt{beta} := \beta)$
$(X\beta)\alpha$	$X\backslash(X\beta)$	$X\alpha$	$(<^{\lvert\alpha\rvert}, \texttt{beta} := \beta)$

Table 1: Conversion from CCG categories to SBR's labels.

3.2 Converting CCG derivations into SBRs

Algorithm 1 obtains an SBR from a CCG derivation. Table 1 summarizes the conversion from categories into SBR's labels. Algorithm 1 uses this table in the function `SBRlabel` that returns a SBR's label. Here, we introduce two additional patterns for adjuncts or type-raised categories (shown in the last two rows).[2] Introducing these patterns reduces the number of SBR's labels.

3.3 Converting SBRs into CCG derivations

Algorithm 2 recovers a CCG derivation from an SBR. The recovery process proceeds in a top-down fashion. First, the root label is recovered from an additional tag `RT`.[3] That is, we call $\texttt{recover}(n, \texttt{RT}(\texttt{label}(n)))$ for an SBR n. Then, the categories of the children are recovered using Table 1 in reverse (the function $\texttt{recoverCAT}(S, P)$ returns the categories). This process is repeated recursively until the leaf nodes are reached. When the SBR's label is in the form of $(>^d, \cdots)$ or $(<^d, \cdots)$ and $\texttt{arity}(P) < d$, L and R cannot be defined. In this case, $\texttt{recoverCAT}(S, P)$ replaces d with $\texttt{arity}(P)$.

Algorithm 1 $\texttt{convert}(n)$

n is a CCG derivation node.
$\texttt{label}(n)$ is the label of n.
$\texttt{par}(n)$ is the parent of n.
$\texttt{chi}_\texttt{L}(n)$, $\texttt{chi}_\texttt{R}(n)$ and $\texttt{chi}_\texttt{U}(n)$ are the left, right and unary child of n.
$\texttt{node}(l, C)$ makes a node with a label l and children C.

if n is a preterminal node **then**
  $n' \leftarrow \texttt{chi}_\texttt{u}(n)$
else if n is binary branching **then**
  $l, r \leftarrow \texttt{chi}_\texttt{L}(n), \texttt{chi}_\texttt{R}(n)$
  $L, R, P \leftarrow \texttt{label}(l), \texttt{label}(r), \texttt{label}(n)$
  $S \leftarrow \texttt{SBRlabel}(L, R, P)$
  if n is a root node **then**
    add `RT` :P to S
  end if
  if l is unary branching **then**
    $l \leftarrow \texttt{chi}_\texttt{u}(l)$
    add a tag `UL` :$\texttt{label}(l)$ to S
  end if
  if r is unary branching **then**
    $r \leftarrow \texttt{chi}_\texttt{u}(r)$
    add a tag `UR` :$\texttt{label}(r)$ to S
  end if
  $n' \leftarrow \texttt{node}(S, \langle \texttt{convert}(l), \texttt{convert}(r) \rangle)$
end if
return n'

4 Generating OOV categories

In our proposed representation, lexical categories are not directly assigned to words. Lexical categories are decomposed into several node labels. This means that lexical categories are not defined by a finite set and that the span-based parsing model learned from SBRs may generate OOV lexical categories that do not appear in the training data.

[2] These are special cases of the CCG rule schemata shown in Figure 1.

[3] If the parsing model fails to assign `RT` tag, we use $\texttt{RT} : \texttt{S}_\texttt{dcl}$ as a default.

Algorithm 2 $\texttt{recover}(n, P)$

n is a node in an SBR.
n' is a node in a CCG derivation.
P, L and R are categories.
$\texttt{UL}(S)$ is a $\texttt{UL}$ tag if exists.
$\texttt{UR}(S)$ is a $\texttt{UR}$ tag if exists.

if n is a terminal (word) node **then**
  $n' = \texttt{node}(P, \langle n \rangle)$
else
  $S \leftarrow \texttt{label}(n)$
  $L, R \leftarrow \texttt{recoverCat}(S, P)$
  if $\texttt{UL}(S) \neq \texttt{null}$ **then**
    $l \leftarrow \texttt{node}(L, \langle \texttt{recover}(\texttt{chi}_\texttt{L}(n), \texttt{UL}(S)) \rangle)$
  else
    $l \leftarrow \texttt{recover}(\texttt{chi}_\texttt{L}(n), L)$
  end if
  if $\texttt{UR}(S) \neq \texttt{null}$ **then**
    $r \leftarrow \texttt{node}(R, \langle \texttt{recover}(\texttt{chi}_\texttt{R}(n), \texttt{UR}(S)) \rangle)$
  else
    $r \leftarrow \texttt{recover}(\texttt{chi}_\texttt{R}(n), R)$
  end if
  $n' \leftarrow \texttt{node}(P, \langle l, r \rangle)$
end if
return n'

5 Experiment

We conducted an experiment using the CCGBank (Hockenmaier and Steedman, 2007)[4] to evaluate the performance of our method.[5] We used the Berkeley Neural Parser (Kitaev and Klein, 2018) with BERT (Devlin et al., 2019) as a span-based parser. We converted the training (sections 02–21) and the development (section 00) data into SBRs and learned the model from the data. The number of SBR's labels in the training data was 486.[6] The hyperparameters for training were identical to those of Kitaev et al. (2019). We evaluated the parsing performance by labeled F_1 on the test data (section 23). We obtained labeled dependencies using the C&C parser's `generate` program (Clark and Curran, 2007). As a baseline model, we trained a model directly using the CCG derivations.

Table 2 shows parsing performances on the test data. Our proposed and the baseline methods have high precision (92.8% and 94.0%) but low recall (82.2% and 76.3%). One of the reasons for the low recall was that the C&C parser's `generate` program failed to obtain dependencies from the output CCG derivations. Our proposed and the baseline methods failed to obtain dependencies from 206 and 371 sentences of 2407 test data sentences, respectively. The `generate` program cannot work when the CCG derivation is invalid or has a lexical category that is not listed in its `markedup` file. To mitigate this problem, we added such lexical categories to the `markedup` file.[7] Adding lexical categories increased the recall (87.6%) of our method significantly. On the other hand, the recall of the baseline method was still low (76.8%) due to the invalid CCG derivations. This result shows that a span-based parsing using CCG derivations does not work well and that our proposed method improves the parsing performance. The final result of our method was comparable with previous CCG parsers.

Method	Pre.	Rec.	F_1
Lewis and Steedman (2014)	–	–	86.1
Xu et al. (2015)	87.7	86.4	87.0
Lewis et al. (2016)	88.6	87.5	88.1
Vaswani et al. (2016)	–	–	88.3
Lee et al. (2016)	–	–	88.7
Xu (2016)	89.8	85.8	87.8
Yoshikawa et al. (2017)	–	–	88.8
Stanojević and Steedman (2019)	–	–	90.5
Bhargava and Penn (2020)	–	–	90.9
Tian et al. (2020)	–	–	90.7
Prange et al. (2021)	–	–	90.8
Liu et al. (2021)	-	–	90.9
Baseline	94.0	76.3	84.2
Baseline + `markedup`	93.9	76.8	84.5
Ours	92.8	82.2	87.2
Ours + `markedup`	91.7	87.6	89.6

Table 2: Labeled F_1 on the test data.

Method	Rec. (%)
Bhargava and Penn (2020)	22
Prange et al. (2021)	3
ours	18

Table 3: Recall for OOV lexical categories on the test data.

5.1 OOV categories

Another interesting point of our method is the possibility of generating OOV categories. Table 3 shows the recall for OOV lexical categories. We obtained a similar result with previous research. Our method correctly assigned OOV categories for 4 words.[8]

[4] In the CCGBank, adjuncts and type-raised categories take an argument category using feature unification. Our method treats this feature unification in the last two rows in Table 1. `SBRlabel` does not allow the X occurring in $X/(X\beta)$ and $X\backslash(X\beta)$ to have any feature, and `recoverCat` removes features from the X. For example, $\texttt{SBRlabel}(\texttt{S/S}, \texttt{S}_\texttt{dcl}, \texttt{S}_\texttt{dcl}) = (>^0, \texttt{beta} := [])$ and $\texttt{RecoverCat}((>^0, \texttt{beta} := []), \texttt{S}_\texttt{dcl}) = (\texttt{S/S}, \texttt{S}_\texttt{dcl})$.

[5] The code is available at https://github.com/yosihide/span-based-ccg-derivation.

[6] The training data has 1639 categories including 1285 lexical ones (supertags).

[7] The new `markedup` file was generated automatically.

[8] There are only 22 occurrences of OOV categories in the test data.

We can say that our proposed approach can treat OOV categories.

6 Conclusion

This paper proposed a new representation for CCG derivations. Our proposed representation realizes a span-based CCG parser that follows the CCG binary rule schemata. Furthermore, the parser can generate OOV categories. One remaining problem in the proposed method is to treat unary rule schemata in CCG. Our method encodes unary rules using the additional information described in Section 3.1.1, but this approach may violate the unary rule schemata. In the future, we will extend the method to treat CCG unary rules validly.

Acknowledgements

We thank anonymous reviewers for their helpful comments.

References

Aditya Bhargava and Gerald Penn. 2020. Supertagging with CCG primitives. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 194–204, Online. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's going on in neural constituency parsers? an analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010, New Orleans, Louisiana. Association for Computational Linguistics.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Marco Kuhlmann and Giorgio Satta. 2014. A new parsing algorithm for Combinatory Categorial Grammar. *Transactions of the Association for Computational Linguistics*, 2:405–418.

Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural CCG parsing with optimality guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2366–2376, Austin, Texas. Association for Computational Linguistics.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, California. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014. Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Yufang Liu, Tao Ji, Yuanbin Wu, and Man Lan. 2021. Generating CCG categories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13443–13451.

Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. Supertagging the Long Tail with Tree-Structured Decoding of Complex Categories. *Transactions of the Association for Computational Linguistics*, 9:243–260.

Miloš Stanojević and Mark Steedman. 2019. CCG parsing algorithm with incremental tree rotation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 228–239, Minneapolis, Minnesota. Association for Computational Linguistics.

Miloš Stanojević and Mark Steedman. 2020. Max-margin incremental CCG parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4111–4122, Online. Association for Computational Linguistics.

Mark Steedman. 2000. *The Syntactic Process*. The MIT press.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Yuanhe Tian, Yan Song, and Fei Xia. 2020. Supertagging Combinatory Categorial Grammar with attentive graph convolutional networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6037–6044, Online. Association for Computational Linguistics.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California. Association for Computational Linguistics.

Wenduan Xu. 2016. LSTM shift-reduce CCG parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1764, Austin, Texas. Association for Computational Linguistics.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 250–255, Beijing, China. Association for Computational Linguistics.

Masashi Yoshikawa, Hiroshi Noji, and Yuji Matsumoto. 2017. A* CCG parsing with a supertag and dependency factored model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 277–287, Vancouver, Canada. Association for Computational Linguistics.