Received December 18, 2020, accepted January 23, 2021, date of publication January 29, 2021, date of current version February 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3055513

Compositional Generalization via Parsing Tree Annotation

SEGWANG KIM[1], JOONYOUNG KIM[1], AND KYOMIN JUNG[1,2], (Member, IEEE)
[1]Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
[2]ASRI, Seoul National University, Seoul 08826, South Korea

Corresponding author: Kyomin Jung (kjung@snu.ac.kr)

This work was supported by the Samsung Research Funding and Incubation Center, Samsung Electronics, under Project SRFC-IT1902-06.

ABSTRACT Humans can understand a novel sentence by parsing it into known components like phrases and clauses. To achieve human-level artificial intelligence, compositional generalization tasks are suggested and used to assess machine learning models. Among those tasks, the SCAN tasks are challenging for the standard deep learning models, such as RNN sequence-to-sequence models and Transformers, that show great success across many natural language processing tasks. Even though a long line of deep learning research has developed memory augmented neural networks aimed at the SCAN tasks, their generalities remain questionable for more complex and realistic applications where the standard seq2seq models dominate. Hence, one needs to propose a method that helps the standard models to discover compositional rules. To this end, we propose a data augmentation technique using paring trees. Our technique annotates targets by inserting a new delimiter token in between them according to their parsing trees. For the training stage, the technique needs prior knowledge about the targets' semantic or syntactic compositionality. On the other hand, for the test stage, the technique uses no such knowledge. Experiments show that our technique enables the standard models to achieve compositional generalization on the SCAN tasks. Furthermore, we validate our technique on a synthetic task and confirm the standard models' strong performance gains without using prior knowledge about semantic compositionality. As one way to infuse parsing tree information into sequences, our technique can be used for tasks with structured targets like program code generation tasks.

INDEX TERMS Artificial intelligence, neural networks, natural language processing.

I. INTRODUCTION

Humans can understand natural language by its compositionality [1], [2]. That is, even if a human reads a sentence written as novel combinations of known phrases or clauses, he or she can parse it into semantic or syntactic components.

To achieve artificial intelligence that possesses such ability, a large body of deep learning research [3]–[10] has been carried out using the SCAN tasks [11], de facto standard compositional generalization problems. The SCAN dataset consists of finitely many commands, e.g., "run twice" and "walk right and run", and their corresponding target actions, e.g., "RUN RUN" and "RTURN WALK RUN". In particular, the SCAN dataset is divided into the training and the test set depending on specific compositionality of interest, and yields tasks like the jump-split task, the length-split task, and the MCD-split tasks. For example, in the training stage, the jump-split task shows commands like "run", "run twice", and "run after walk". However, it does not show "jump" with any context, such as "jump twice" or "jump twice after walk", except for "jump" itself. In the test stage, it asks those unobserved commands with "jump". As another example, the length-split task's test set suggests commands requiring longer actions than those that appear in the training set.

These SCAN tasks turn out to be extremely difficult to standard seq2seq deep learning models, such as RNN sequence-to-sequence (seq2seq) models [12]–[14] and self-attention based models [15]–[17]. This seems contradictory to recent advances, which are up to par with the level of human intelligence, in numerous natural language tasks including machine translation, natural language inference (NLI), and question and answering (QA). Moreover, studies about theoretical analysis of RNNs [18], [19] and self-attention [20], [21] have shown that their expressive powers are enough to capture the hierarchical structure of the

The associate editor coordinating the review of this manuscript and approving it for publication was Longzhi Yang.

SCAN tasks' language whose grammar allows only finitely many words.

Fortunately, it has been found that the standard models can solve the jump-split SCAN task with the help of pretraining [22] or data augmentation [9], [23]. This is possible since many additional examples apart from the given training ones enable the standard models to experience enough compositionality. However, none of these data-based approaches succeeds to make the standard models to resolve the other types of the SCAN tasks like the length-split or the MCD-splits.[1]

To tackle those types of the SCAN tasks, there have been attempts to design new architectures largely deviated from the standard seq2seq architectures [6], [10]. These new architectures commonly exploit external memory allowing merging or concatenation operations aimed at the SCAN tasks' compositional rules such as "twice" or "thrice". However, such non-standard architectures with external memory for imposing task-specific inductive bias are only applicable to the SCAN or the SCAN-like tasks. Thus, they cannot be used to solve more complex and realistic applications where the standard models perform well.

In this work, to achieve compositional generalization with the standard seq2seq models, we suggest a novel data augmentation technique using parsing trees. The technique annotates each original target sequence by inserting a new delimiter token "<del>" in between the target for distinguishing its parsed components, as shown in Figure 1. For the training stage, the annotated targets are used instead of the original ones. Here, to obtain those parsed components, the technique uses prior knowledge about the original targets' semantic or syntactic compositionality. On the other hand, in the test stage, the technique does not need any such knowledge.

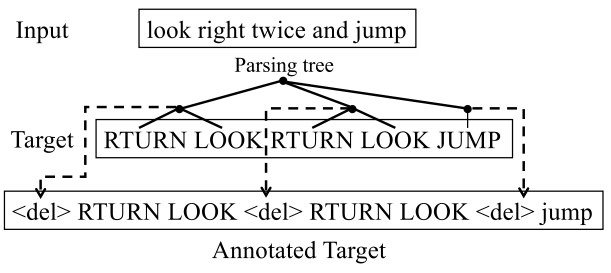
Input
look right twice and jump
Parsing tree
Target
RTURN LOOK RTURN LOOK JUMP
<del> RTURN LOOK <del> RTURN LOOK <del> jump
Annotated Target

FIGURE 1. An example of applying our annotation technique. Delimiter tokens <del> indicate the beginnings of the parsed components obtained from the parsing tree.

Empirically, we show that our technique enables the standard seq2seq models to achieve compositional generalization on the MCD-splits and the length-split of the SCAN dataset. We further validate our technique on a synthetic task and confirm the standard models' strong performance gains even without using prior knowledge about semantic compositionality. This shows our technique's applicability on more challenging compositional natural language tasks where syntactic parsing trees are readily available, such as programming code generation tasks [24].

[1]The detailed description is deferred to Section II-A4

II. BACKGROUND

In this section, we introduce the SCAN tasks by split methods and previous studies to tackle them via architecture developments and data augmentations.

A. THE SCAN TASKS

The goal of the SCAN (Simplified version of the CommAI Navigation) tasks [11] is to translate compositional navigation commands written in synthetic natural language into a sequence of actions. The inputs are commands, a total of 20,910, formed by a predefined grammar (Fig. 2) and the targets are actions that are the translation results of commands by the semantic interpretation mapping (Appx.). Depending on compositional generalization abilities to assess, split methods that divide all command-action pairs into the training or the test set are determined. Accordingly, specific tasks are defined as follows.

1) RANDOM-SPLIT

The training set is a random 80% subset of the total dataset and the test set is the remaining subset. Thus, this task is not for assessing compositional generalization ability but is used to test the given models' typical generalization abilities. Unlike other splits, the standard seq2seq models generalize well on the test set.

2) JUMP-SPLIT

The training set consists of all primitives, e.g., "jump", "walk", "run", and "look", and their composed commands, e.g., "run twice", "walk opposite left and run twice", except for composed commands of "jump". The test set contains the remaining commands like "jump twice" and "jump after run". Hence, to generalize on the test set, compositional understanding of "jump" along with other commands is necessary.

3) LENGTH-SPLIT

The training set has all 16,990 commands (81.3% of the total) requiring actions, i.e., targets, of lengths less than 24 and the test set has all remaining commands. Hence, the test set assesses the compositional generalization abilities about actions' lengths.

4) MCD-SPLIT

As the composition of commands can be explained by their grammar parsing trees, it is natural to consider a distribution over those trees' subgraphs (compounds). To define the compound distribution of the dataset, DBCA (distribution-based compositionality assessment) method [24] captures the extent of how interesting a subgraph is within a parsing tree. Moreover, the method can serve as criteria to divide the SCAN dataset into the training and the test set, yielding three compositional generalization tasks, such as MCD1,

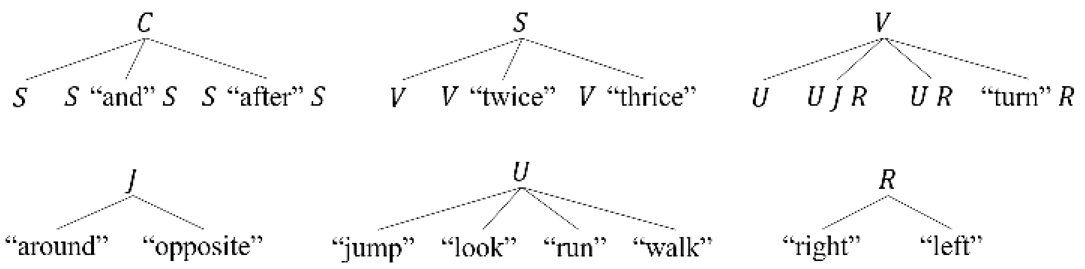

FIGURE 2. **A Grammar for commands.**

MCD2, and MCD3. Whereas the training and test set of each task have similar distributions over nodes (atoms) of parsing trees, they have distinct distributions over subgraphs (compounds).

B. SCAN-INSPIRED DEEP LEARNING ARCHITECTURES

The SCAN tasks inspire many deep learning architecture designs for compositional generalization, which are thoroughly summarized in [22]. In particular, NeSS [10] and LANE [6] achieve perfect accuracy on all splits of the SCAN tasks. NeSS is a memory-augmented neural network (MANN) using a neural stack machine controlled by manually-defined instruction semantics. Among its instructions, $\texttt{CONCAT}_{\texttt{M}}$ or $\texttt{CONCAT}_{\texttt{S}}$ plays a crucial role to handle commands requiring repetitions like "around" or "twice" (Refer to Appx.). Also, LANE is another MANN consisting of the composer and the solver neural networks with memory. After the composer merges repetitive adjacent commands and yields analytic expression, the solver translates and records it on the memory. To sum up, both architectures' common intuition is to capture repeating patterns of actions according to commands. In this work, we annotate those repeating patterns by inserting delimiter tokens based on parsing trees.

C. DATA-BASED APPROACH FOR THE SCAN TASKS

Exposing models to more examples is another way to teach how to handle novel combinations. To solve the jump-split SCAN task, GECA, a data augmentation method, [23], synthesizes new training examples by mixing given training examples' components that have never been collocated. Also, synthesizing more training examples about primitives usages within contexts by introducing hundreds of new primitives is possible [9]. Other than augmenting training examples about the SCAN tasks, pretraining transformers, i.e., T5 [17], on a wide range of other natural language tasks, such as machine translation, question and answering tasks, and natural language inference tasks, turns out to be effective too [22]. However, all of these data-based approaches are not enough for training the standard seq2seq models to generalize on the other splits of the SCAN tasks like the length-split or the MCD-split. In this work, we present a data augmentation technique that works on those uncharted tasks.

III. METHOD

In the next section, we point out that the SCAN dataset allows abundant many-to-one cases. Then, we introduce our annotation technique using parsing trees to handle them.

A. MANY-TO-ONE CASES OF THE SCAN DATASET

Learning the compositions in the SCAN tasks can be regarded as discovering which part of the commands, e.g., the second "*run*" in the "run after *run*" or the first "*run*" in the "*run* and run", corresponds to which part of the actions, e.g, the first "*RUN*" in the "*RUN* RUN". Hence, it is natural to assume that the presence of multiple commands corresponding to the same action sequence makes it harder to learn the compositions, i.e. many-to-one. In fact, such cases are fairly common in the SCAN dataset as its non-injective semantic interpretation function maps 20,910 commands to only 9,228 different actions. In the extreme case, the action sequence "RTURN RTURN RTURN RTURN RTURN RTURN" is the target of 19 different commands such as "turn around right and turn right twice", "turn opposite right thrice", and "turn right twice and turn opposite right twice".

B. OUR ANNOTATION TECHNIQUE

We hypothesize that such abundant many-to-one cases confuse the standard seq2seq models to learn the compositionality. To reduce many-to-one cases, we annotate targets by inserting new delimiter tokens in between the actions according to the commands.

Specifically, our annotation technique can be described as follows. First, we induce a grammar for the target language, possibly via human parser or grammar induction heuristics [25], [26]. At this point, it is desirable to induce the grammar that can capture the compositionality of the target language with the minimum number of relations. Then, we obtain the parsing tree for each target sequence. In some cases, the induced grammar may allow multiple parsing trees, i.e., the grammar is ambiguous. To uniquely decide the parsing tree, we refer to the input sequence and the semantic interpretation function. Finally, we choose a non-terminal variable where a new delimiter token like "<del>" to be attached. We insert the token before every substring generated from the variable. These annotated targets, instead of original ones, are used for the training. See Section IV-B1, and Section IV-C2

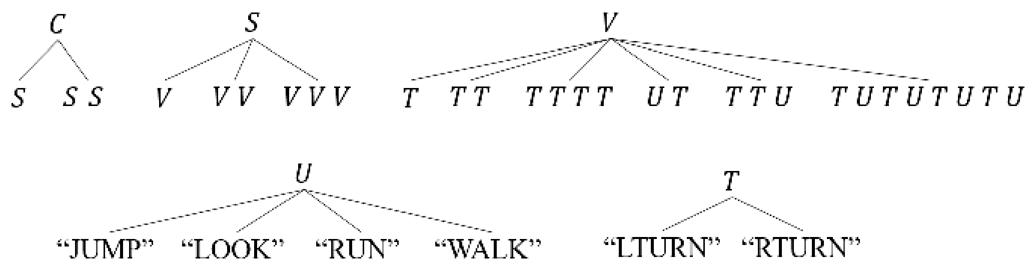

FIGURE 3. A Grammar for actions.

about specific implementations for the SCAN dataset and our synthetic dataset, respectively.

IV. EXPERIMENTAL SETUPS

In this section, we describe the implementation details of the standard seq2seq models. Then, we introduce experimental details, such as specific implementations of our annotation technique, for the SCAN tasks and the multiplicative extension tasks.

A. BENCHMARKS: STANDARD Seq2seq MODELS

We verified our technique with the standard seq2seq models: an LSTM seq2seq model (LSTM), a GRU seq2seq model (GRU), [12], and those with Bahdanau attention [13] (LSTM-Atten, GRU-Atten), a Transformer [15], and a T5 [17]. All the RNN seq2seq models had one layer with the hidden size 50 and the dropout rate 0.5. The Transformer and the T5 consisted of six layers with the hidden size 512 and eight attention heads. We used the ADAM optimizer [27] with a learning rate $1e^{-3}$ to train the RNN seq2seq models and the T5. As for the Transformer, we varied learning rates along the course of the training [15]. All models could fit in a single NVIDIA GTX 1080ti GPU. Our implementations[2] except for the T5 based on the open source library *tensor2tensor*[3] while we used *hugging face trasnformers*[4] for the T5.

B. SCAN

1) ANNOTATION IMPLEMENTATION

For the SCAN tasks, we used the ground-truth interpretation function when (i) inducing grammar for the action language and (ii) obtaining the unique parsing tree for each action sequence.

We annotated the examples as follows. First, we manually induced a grammar for the action language, as shown in Fig. 3, to be similar to the given grammar of the command language. Then, we obtained the unique parsing tree for each action sequence according to its command's parsing tree and the ground truth interpretation function $[\![\cdot]\!]$. For example, the action sequence "RTURN RTURN RTURN RTURN" from "turn opposite right twice" and "turn around right" corresponded to the left and the right of Fig. 4, respectively. Finally, we inserted a new delimiter token "<del>" before every substring generated from the non-terminal variable V. For the aforementioned example, the former action sequence was annotated as "<del> RTURN RTURN <del> RTURN RTURN" while the later one was annotated as "<del> RTURN RTURN RTURN RTURN". This annotation process is summarized as Alg. 1.

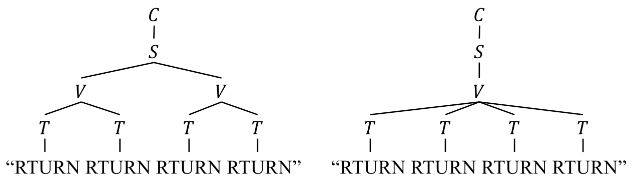

FIGURE 4. Two possible parsing trees for "RTURN RTURN RTURN RTURN". The left and right tree are come from commands "turn opposite right twice" and "turn around right", respectively.

Algorithm 1 Annotation Process for the SCAN Dataset

```
Input: command sequence $C$.
def PTA($C$): // stands for Parsing Tree Annotation
  if "and" in $C$:
    $C1 \leftarrow$ the command before "and" within $C$
    $C2 \leftarrow$ the command after "and" within $C$
    return PTA($C1$) + PTA($C2$)
  if "after" in $C$:
    $C1 \leftarrow$ the command before "after" within $C$
    $C2 \leftarrow$ the command after "after" within $C$
    return PTA($C2$) + PTA($C1$)
  if "twice" in $C$:
    $C \leftarrow$ the command before "twice" within $C$
    return PTA($C$) + PTA($C$)
  if "thrice" in $C$:
    $C \leftarrow$ the command before "thrice" within $C$
    return PTA($C$) + PTA($C$) + PTA($C$)
  return ["<del>"] + $[\![C]\!]$
Output: annotated action sequence PTA($C$).
```

2) TOKENIZATION FOR T5

Whereas we fed all sequences word-by-word for the RNN seq2seq models and Transformer, we had no choice but to use the pretrained tokenizer coupled with the T5. Using the

[2]https://github.com/segwangkim/annotation-of-targets-using-parsing-trees
[3]https://github.com/tensorflow/tensor2tensor
[4]https://github.com/huggingface/transformers

pretrained tokenizer could be problematic as the tokenizer took the raw actions and segmented them into components that capture no semantics. For example, "I_TURN_LEFT"[5] was segmented into ("I", "_", "TUR", "N", "_", "LE", "FT"). Thus, compositional rules or linguistic semantics learned from the pretraining corpus became useless for the fine-tuning on the SCAN tasks. To resolve this issue, we preprocessed actions with straightforward modifications, e.g., "I_TURN_LEFT" to "lturn", before applying the tokenizer. By doing so, the pretrained tokenizer segmented actions more reasonably, e.g., "lturn" was segmented into ("l", "turn"). We denoted a T5 with the above tokenization method as T5*.

C. MULTIPLICATIVE EXTENSION TASKS

Observe that we used prior knowledge about the ground-truth interpretation function and the input and the target sequences' grammars for the SCAN tasks. To validate our technique's applicability under minimal prior knowledge, we further suggest a simple synthetic task that requires neither the interpretation function nor parsing trees of input sequences for applying our technique.

1) TASK DEFINITION

The goal of multiplicative extension tasks is to translate a sequence of alphabet-number alternating terms into alphabet sequences. The input is given as:

$$a_1 d_1 a_2 d_2 \cdots a_k d_k$$

where a_i is an alphabet sampled from an alphabet set $\Sigma = \{\text{"a"}, \ldots, \text{"g"}\}$ without replacement and d_i is one-digit integer. The target under the ground-truth interpretation function f is given as:

$$f(a_1 d_1 a_2 d_2 \cdots a_k d_k) = \underbrace{a_1 \cdots a_1}_{d_1} \cdots \underbrace{a_k \cdots a_k}_{d_k}$$

Thus, this task tests multiplicative compositionality similar to that of the SCAN tasks, i.e., "twice" and "thrice". We define training and test sets according to the maximum length of targets, $n = d_1 + \cdots d_k$, as shown in Table 1.

TABLE 1. The training and test datasets of the multiplicative extension tasks. The number of possible alphabets is fixed as 7 ("a" to "g").

		n	k	$\min(d_i)$	$\max(d_i)$
Training		7, 8, 9	3	0	7
Test	ID	7, 8, 9	3	0	7
	OOD-Easy	12	3	1	7
	OOD-Hard	18	3	1	7

Note that this task shares no linguistic compositionality with natural language corpus so that there is no advantages from pretraining. Hence, we omit to test T5 at this task.

[5]The original SCAN datasets (https://github.com/brendenlake/SCAN.git) represents actions as snake upper case with leading "I" such as "I_TURN_LEFT", "I_LOOK", and "I_JUMP" unlike Fig. 3.

2) ANNOTATION IMPLEMENTATION

We used straightforward parsing trees that explain the target sequences. For given target $a_1 \cdots a_1 \cdots a_k \cdots a_k$ where each a_i repeats d_i times, we inserted a new delimiter token $s \notin \Sigma$, e.g., $s =$ "<del>", before the repetitions of the same a_i's, yielding $sa_1 \cdots a_1\ sa_2 \cdots a_2 \cdots sa_k \cdots a_k$. This is natural as a grammar of the target language can be defined as $S \to V \mid VV \mid VVV \cdots$, $V \to T \mid TT \mid TTT \mid \cdots$ where S is a start symbol, V is a non-terminal symbol, and $V \in \Sigma$ is a terminal symbol. Note that this annotation was independent of parsing trees of inputs and the interpretation function f.

V. RESULTS AND DISCUSSION

In this section, we empirically verify the effectiveness of our annotation technique in various aspects. First, we point out that our technique can reduce many-to-one cases in the SCAN tasks. Then, we present the effects of our technique on the compositional generalization tasks and analyze our technique in the view of automata theory. Finally, we suggest that a target tokenization method should be carefully chosen for the compositional generalization of the standard models.

A. REDUCED MANY-TO-ONE CASES

First, we analyze how much our technique reduces many-to-one cases of the SCAN dataset. To do so, let us formally describe the dataset as follows. Let $\mathcal{L}_C$ be the set of command sequences. Let $\mathcal{L}_A$ and $\mathcal{L}_A^\dagger$ be the sets of action sequences before and after applying our annotation technique, respectively. Accordingly, we also define $[\![\cdot]\!] : \mathcal{L}_C \to \mathcal{L}_A$ and $[\![\cdot]\!]^\dagger : \mathcal{L}_C \to \mathcal{L}_A^\dagger$, i.e. PTA in Alg. 1, as corresponding ground-truth interpretation functions.

To count many-to-one cases, for each interpretation function $f = [\![\cdot]\!]$ or $[\![\cdot]\!]^\dagger$, we first partitioned the domain $\mathcal{L}_C = \cup_{i=1}^k C_i$ by disjoint cells where each cell C_i is the set of commands that are mapped to the identical actions, i.e., $f(x) = f(y)\ \forall x, y \in C_i$. Then, according to the sizes of those cells, i.e., $|C_i|$, we counted the frequency and visualized it as a histogram as shown in Fig. 5.

While the total areas of all histograms under functions $f = [\![\cdot]\!]$ and $[\![\cdot]\!]^\dagger$ are identical as 20,910, the shapes of

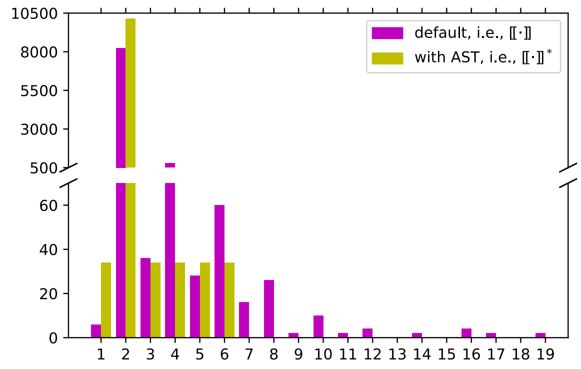

FIGURE 5. Histograms of many-to-one cases for [[·]] and [[·]]*. Each n of x-axis is the number of commands that are mapped to the same actions while its height is the frequency of such n-to-one cases.

TABLE 2. The sequence-level accuracy before and after applying our annotation technique (PTA) on SCAN tasks results (for 5 runs). T5* is a T5 with our tokenization method (refer to IV-B2).

split		GRU	LSTM	GRU+Atten	LSTM+Atten	T5*	T5	Transformer
length	baseline	0.13 ± 0.06	0.14 ± 0.03	0.21 ± 0.02	0.15 ± 0.02	0.18 ± 0.02	0.17 ± 0.02	0.00 ± 0.00
	+PTA	0.96 ± 0.05	0.80 ± 0.15	0.99 ± 0.04	1.00 ± 0.00	0.94 ± 0.03	0.16 ± 0.08	0.00 ± 0.00
mcd1	baseline	0.19 ± 0.08	0.10 ± 0.02	0.32 ± 0.23	0.23 ± 0.04	0.07 ± 0.02	0.16 ± 0.10	0.03 ± 0.01
	+PTA	0.65 ± 0.07	0.71 ± 0.08	0.96 ± 0.08	0.94 ± 0.11	0.60 ± 0.12	0.80 ± 0.07	0.01 ± 0.01
mcd2	baseline	0.11 ± 0.09	0.09 ± 0.01	0.10 ± 0.03	0.11 ± 0.01	0.09 ± 0.02	0.10 ± 0.02	0.04 ± 0.01
	+PTA	0.18 ± 0.11	0.37 ± 0.31	0.52 ± 0.20	0.80 ± 0.15	0.28 ± 0.15	0.58 ± 0.10	0.00 ± 0.00
mcd3	baseline	0.19 ± 0.07	0.16 ± 0.05	0.23 ± 0.07	0.32 ± 0.05	0.08 ± 0.01	0.08 ± 0.01	0.05 ± 0.01
	+PTA	0.49 ± 0.12	0.58 ± 0.15	0.91 ± 0.14	0.76 ± 0.15	0.58 ± 0.12	0.58 ± 0.06	0.02 ± 0.01
mcd (mean)	baseline	0.16 ± 0.08	0.12 ± 0.03	0.22 ± 0.11	0.22 ± 0.03	0.08 ± 0.01	0.11 ± 0.04	0.04 ± 0.01
	+PTA	0.44 ± 0.10	0.55 ± 0.18	0.80 ± 0.14	0.83 ± 0.14	0.49 ± 0.13	0.65 ± 0.08	0.01 ± 0.01

TABLE 3. The sequence-level accuracy on the multiplicative extension task (for 5 runs).

		GRU	LSTM	GRU+Atten	LSTM+Atten	Transformer
ID	baseline	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.85 ± 0.21
	+PTA	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
OOD	baseline	0.03 ± 0.05	0.33 ± 0.14	0.00 ± 0.00	0.04 ± 0.07	0.00 ± 0.00
Easy	+PTA	0.28 ± 0.15	0.35 ± 0.10	0.98 ± 0.05	0.97 ± 0.02	0.00 ± 0.00
OOD	baseline	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Hard	+PTA	0.00 ± 0.00	0.00 ± 0.00	0.32 ± 0.21	0.17 ± 0.15	0.00 ± 0.00

them are different. Note that the histogram for $[\![\cdot]\!]$ has a long tail whereas that for $[\![\cdot]\!]^\dagger$ has a short tail, indicating that one-to-many cases are significantly reduced by our annotation. From this result, we can conclude that our technique helps the standard models discover the training examples' compositionality significantly easier.

B. EFFECTS ON COMPOSITIONAL GENERALIZATION

After applying our annotation technique, the SCAN length-split task is almost perfectly solved by the attentional RNN seq2seq models (GRU+Atten, LSTM+Atten) and the T5 with the manual tokenization (T5*) as shown in Table 2. Moreover, we also obtain huge performance gains at the multiplicative extension tasks using the attentional RNN seq2seq models as shown in Table 3. These imply that the models have sufficient expressive powers to handle test commands requiring longer target sequences. In other words, the standard seq2seq models fail on the length generalization tasks as they learn incorrect compositionality during the training stage.

Rather than the length generalization, our technique also induces huge performance gains for the SCAN MCD-split tasks as shown in Table 2. One may argue that the compound distribution discrepancy between the training and test set can be changed as our technique inserts delimiter tokens for the actions. Thus, our comparison between the results on the MCD-splits before and after applying our technique seems unfair. However, our comparison is still valid since the way of measuring the discrepancy only depends on parsing trees for commands, not actions.

Unfortunately, the Transformers still fail to generalize in all tasks regardless of our annotation. In other words, our technique is effective only for self-attention based models that have experienced sufficient linguistic compositionality from pretraining in advance.

C. AUTOMATA-THEORETIC VIEW

One might think that all aforementioned results are achieved since the tasks become much easier after the ground-truth interpretation functions are modified as shown in Alg. 1. However, in the view of automata theory, our technique makes no difference in the level of difficulty. Indeed, both tasks before and after applying our annotation can be implemented by a Finite State Transducer (FST). To see this, note that FST can implement a rational relation between two regular languages. The source language, e.g., $\mathcal{L}_C$, and the target language, e.g., $\mathcal{L}_A$ or $\mathcal{L}_A^\dagger$, have finitely many words, hence they are all regular languages. Moreover, the interpretation function, e.g., $[\![\cdot]\!]$ or $[\![\cdot]\!]^\dagger$ is a rational relation as the graph of the function is finite. Therefore, the difficulties of both tasks cannot be distinguished.

D. TOKENIZATION MATTERS FOR LEARNING COMPOSITIONALITY

The performance discrepancies between the T5 and the T5* for the SCAN length-split and MCD-split tasks with our annotation are notable. In particular, as for the length-split, our annotation cannot induce any performance gain at all when the actions are naively tokenized without considering their semantics (T5).

Note that we can think of inserting delimiter tokens to the target sentences in the multiplicative extension tasks as another tokenization for the targets. This is because tokenization is transforming a string into a sequence while considering adjacent characters that co-occur frequently.

Henceforth, we can conclude that a tokenization method can significantly influence the compositionality that the standard models learn. To go further, at compositional generalization task whose targets follow strict grammar, it is beneficial to use tokenization based on the grammar. For example, code generation tasks have targets that can be represented as abstract syntax trees (AST). Since our technique annotates the target sequences with delimiters indicating specific non-terminal nodes of the targets' parsing trees, trees' compositionality can be infused into the sequences. Thus, we expect that our technique can promote the standard seq2seq model to achieve compositional generalization on those tasks.

VI. CONCLUSION AND OUTLOOK

We propose a new data augmentation technique using parsing trees to help the standard seq2seq model discover compositions of sequences. We empirically show that training with targets annotated by our technique enables standard seq2seq models, such as RNN seq2seq models or Transformers, to generalize well on the length and MCD splits of the SCAN tasks. Moreover, we validate our technique in a synthetic task that requires no knowledge about semantic compositional rules for the annotation.

In this work, we obtain parsing trees using prior knowledge about compositionality, which may be impossible for general cases. Fortunately, studies about obtaining parsing trees in data-driven ways have been carried out [28], [29]. As our technique can be easily incorporated with such methods, we hope that ours motivates deep learning approaches based on the standard models for compositional generalization without using prior knowledge.

APPENDIX
THE GROUND-TRUTH INTERPRETATION FUNCTIONS OF THE SCAN DATASETS

The semantic interpretation function of the SCAN dataset is defined as follows.

$$[\![\text{walk}]\!] = \text{WALK}$$
$$[\![\text{run}]\!] = \text{RUN}$$
$$[\![\text{look}]\!] = \text{LOOK}$$
$$[\![\text{jump}]\!] = \text{JUMP}$$
$$[\![\text{turn left}]\!] = \text{LTURN}$$
$$[\![\text{turn right}]\!] = \text{RTURN}$$
$$[\![u \text{ left}]\!] = \text{LTURN } [\![u]\!]$$
$$[\![u \text{ right}]\!] = \text{RTURN } [\![u]\!]$$
$$[\![\text{turn opposite left}]\!] = \text{LTURN LTURN}$$
$$[\![\text{turn opposite right}]\!] = \text{RTURN RTURN}$$
$$[\![u \text{ opposite left}]\!] = \text{LTURN LTURN } [\![u]\!]$$
$$[\![u \text{ opposite right}]\!] = \text{RTURN RTURN } [\![u]\!]$$
$$[\![\text{turn around left}]\!] = \text{LTURN LTURN LTURN LTURN}$$
$$[\![\text{turn around right}]\!] = \text{RTURN RTURN RTURN RTURN}$$
$$[\![u \text{ around right}]\!] = \text{LTURN } [\![u]\!] \text{ LTURN } [\![u]\!] \text{ LTURN } [\![u]\!] \text{ LTURN } [\![u]\!]$$
$$[\![u \text{ around right}]\!] = \text{RTURN } [\![u]\!] \text{ RTURN } [\![u]\!] \text{ RTURN } [\![u]\!] \text{ RTURN } [\![u]\!]$$
$$[\![x \text{ twice}]\!] = [\![x]\!][\![x]\!]$$
$$[\![x \text{ thrice}]\!] = [\![x]\!][\![x]\!][\![x]\!]$$
$$[\![x_1 \text{ and } x_2]\!] = [\![x_1]\!][\![x_2]\!]$$
$$[\![x_1 \text{ after } x_2]\!] = [\![x_2]\!][\![x_1]\!]$$

Here, double brackets ($[\![\cdot]\!]$) denote the mappings from commands to actions (denoted by uppercase strings). Symbols x and u denote variables which are limited to primitives like "walk", "look", "run", and "jump". The linear order of movements denotes their temporal sequence.

REFERENCES

[1] N. Chomsky, *Syntactic Structures*. Berlin, Germany: Walter de Gruyter, 2002.

[2] R. Montague, "Universal grammar," *Theoria*, vol. 36, no. 3, pp. 373–398, 1970.

[3] J. Bastings, M. Baroni, J. Weston, K. Cho, and D. Kiela, "Jump to better conclusions: SCAN both left and right," in *Proc. EMNLP Workshop BlackboxNLP, Analyzing Interpreting Neural Netw. NLP*, Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 47–55. [Online]. Available: https://www.aclweb.org/anthology/W18-5407

[4] R. Dessì and M. Baroni, "CNNs found to jump around more skillfully than RNNs: Compositional generalization in Seq2seq convolutional networks," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3919–3923. [Online]. Available: https://www.aclweb.org/anthology/P19-1381

[5] Y. Li, L. Zhao, J. Wang, and J. Hestness, "Compositional generalization for primitive substitutions," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 4293–4302. [Online]. Available: https://www.aclweb.org/anthology/D19-1438

[6] Q. Liu, S. An, J.-G. Lou, B. Chen, Z. Lin, Y. Gao, B. Zhou, N. Zheng, and D. Zhang, "Compositional generalization by learning analytical expressions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–14.

[7] J. Loula, M. Baroni, and B. Lake, "Rearranging the familiar: Testing compositional generalization in recurrent networks," in *Proc. EMNLP Workshop BlackboxNLP, Analyzing Interpreting Neural Netw. (NLP)*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 108–114. [Online]. Available: https://www.aclweb.org/anthology/W18-5413

[8] J. Russin, J. Jo, R. O'Reilly, and Y. Bengio, "Compositional generalization by factorizing alignment and translation," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics, Student Res. Workshop*, 2020, pp. 313–327.

[9] P. P. Kagitha, "Systematic generalization emerges in seq2seq models with variability in data," in *Proc. ICLR Workshop Bridging AI Cogn. Sci.* Bengaluru, India: Akaike Technologies, 2020. [Online]. Available: https://baicsworkshop.github.io/pdf/BAICS_11.pdf

[10] X. Chen, C. Liang, A. W. Yu, D. Song, and D. Zhou, "Compositional generalization via neural-symbolic stack machines," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020.

[11] B. M. Lake and M. Baroni, "Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks," 2017, *arXiv:1711.00350*. [Online]. Available: http://arxiv.org/abs/1711.00350

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*. [Online]. Available: http://arxiv.org/abs/1409.0473

[14] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*. [Online]. Available: http://arxiv.org/abs/1508.04025

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805

[17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[18] K. Gulordava, P. Bojanowski, E. Grave, T. Linzen, and M. Baroni, "Colorless green recurrent networks dream hierarchically," 2018, *arXiv:1803.11138*. [Online]. Available: http://arxiv.org/abs/1803.11138

[19] T. Linzen, E. Dupoux, and Y. Goldberg, "Assessing the ability of LSTMs to learn syntax-sensitive dependencies," *Trans. Assoc. Comput. Linguistics*, vol. 4, pp. 521–535, Dec. 2016. [Online]. Available: https://www.aclweb.org/anthology/Q16-1037

[20] M. Hahn, "Theoretical limitations of self-attention in neural sequence models," 2019, *arXiv:1906.06755*. [Online]. Available: http://arxiv.org/abs/1906.06755

[21] C. Yun, S. Bhojanapalli, A. Singh Rawat, S. J. Reddi, and S. Kumar, "Are transformers universal approximators of sequence-to-sequence functions?" 2019, *arXiv:1912.10077*. [Online]. Available: http://arxiv.org/abs/1912.10077

[22] D. Furrer, M. van Zee, N. Scales, and N. Schärli, "Compositional generalization in semantic parsing: Pre-training vs. Specialized architectures," 2020, *arXiv:2007.08970*. [Online]. Available: http://arxiv.org/abs/2007.08970

[23] J. Andreas, "Good-enough compositional data augmentation," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, Jul. 2020, pp. 7556–7566. [Online]. Available: https://www.aclweb.org/anthology/2020.acl-main.676

[24] D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. van Zee, and O. Bousquet, "Measuring compositional generalization: A comprehensive method on realistic data," 2019, *arXiv:1912.09713*. [Online]. Available: http://arxiv.org/abs/1912.09713

[25] R. K. Ando and L. Lee, "Mostly-unsupervised statistical segmentation of japanese: Applications to kanji," in *Proc. 1st North Amer. Chapter Assoc. Comput. Linguistics Conf.*, 2000, pp. 241–248.

[26] G. Carroll and E. Charniak, "Two experiments on learning probabilistic dependency grammars from corpora," Dept. Comput. Sci., Brown Univ., Providence, RI, USA, AAAI Tech. Rep. WS-92-01, 1992. [Online]. Available: https://www.aaai.org/Papers/Workshops/1992/WS-92-01/WS92-01-001.pdf

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[28] S. Jaf and C. Calder, "Deep learning for natural language parsing," *IEEE Access*, vol. 7, pp. 131363–131373, 2019.

[29] S. Qi, B. Jia, and S.-C. Zhu, "Generalized earley parser: Bridging symbolic grammars and sequence data for future prediction," 2018, *arXiv:1806.03497*. [Online]. Available: http://arxiv.org/abs/1806.03497

SEGWANG KIM received the B.S. degree in mathematics from Seoul National University, Seoul, South Korea, in 2016, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His research interests include compositional generalization and neuro-symbolic deep learning approach.

JOONYOUNG KIM received the B.S. degree in electrical engineering from KAIST. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University. In 2009, he has worked as a Research Internship at LG Electronics. His research interests include crowdsourcing, continual and meta learning, and data analysis.

KYOMIN JUNG (Member, IEEE) received the B.S. degree in mathematics from Seoul National University (SNU), Seoul, South Korea, in 2003, and the Ph.D. degree in mathematics from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2009.

From 2009 to 2013, he was an Assistant Professor with the Department of Computer Science, KAIST. Since 2016, he has been an Associate Professor with the Department of Electrical and Computer Engineering, SNU. He is currently an Adjunct Professor with the Department of Mathematical Sciences, SNU. His research interests include natural language processing, deep learning and applications, data analysis, and web services.