Do Deep Neural Networks Capture Compositionality in Arithmetic Reasoning?

Keito Kudo[1] **Yoichi Aoki**[1] **Tatsuki Kuribayashi**[1,2] **Ana Brassard**[3,1]
Masashi Yoshikawa[1] **Keisuke Sakaguchi**[1,3] **Kentaro Inui**[1,3]
[1]Tohoku University [2]Langsmith, Inc. [3]RIKEN
{keito.kudo.q4, youichi.aoki.p2}@dc.tohoku.ac.jp, kuribayashi@tohoku.ac.jp,
ana.brassard@riken.jp, {yoshikawa,keisuke.sakaguchi,inui}@tohoku.ac.jp

Abstract

Compositionality is a pivotal property of symbolic reasoning. However, how well recent neural models capture compositionality remains underexplored in the symbolic reasoning tasks. This study empirically addresses this question by systematically examining recently published pre-trained seq2seq models with a carefully controlled dataset of multi-hop arithmetic symbolic reasoning. We introduce a *skill tree* on compositionality in arithmetic symbolic reasoning that defines the hierarchical levels of complexity along with three compositionality dimensions: systematicity, productivity, and substitutivity. Our experiments revealed that among the three types of composition, the models struggled most with systematicity, performing poorly even with relatively simple compositions. That difficulty was not resolved even after training the models with intermediate reasoning steps.[1]

1 Introduction

Integrating symbolic reasoning capabilities into neural models has been a crucial goal of artificial intelligence (Marcus, 2003; d'Avila Garcez and Lamb, 2020). With this in mind, many researchers investigated how well modern neural models achieve symbolic reasoning (Lake and Baroni, 2018). However, recent studies have reported conflicting results on this; some suggest that neural models can solve complex multi-hop reasoning (Clark et al., 2020), while others claim that models struggle even with performing simple symbolic operations (Qian et al., 2022).

As a step toward further understanding neural models' symbolic reasoning ability, this study systematically analyzes recently published pre-trained seq2seq models using a carefully controlled dataset of multi-hop arithmetic symbolic reasoning.

[1]Our code and data are available at https://github.com/keitokudo/dentaku_skill_tree.

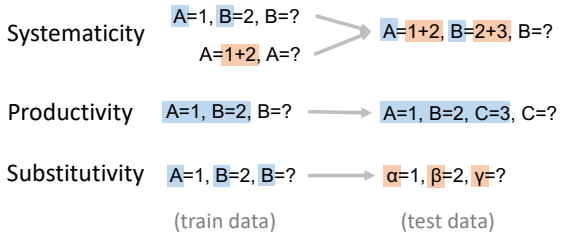

Figure 1: Three dimensionalities of compositionality in arithmetic symbolic reasoning

Specifically, our study empirically evaluates the models' ability to generalize the *compositionality* underlying arithmetic reasoning, where we explore three dimensions of compositionality: (i) systematicity, (ii) productivity, and (iii) substitutivity, as illustrated in Figure 1. Capturing compositionality is crucial in performing symbolic reasoning since compositionality is a pivotal property of generalizability over training instances.

To systematically explore the models' composition ability, we introduce a *skill tree on compositionality* that defined the hierarchical levels of complexity in arithmetic symbolic reasoning, as illustrated in Figure 2. Using this hierarchy as a lens, we identify the limitations of the neural seq2seq models in capturing the compositionality in arithmetic symbolic reasoning. Our major findings can be summarized as follows:

- Among the three types of composition, the models struggled most with **systematicity**, performing poorly even with relatively simple compositions.
- The major difficulty in systematicity was in the access to intermediate information that is not stated in input but produced during the reasoning.
- Capturing systematicity remained hard for the models trained with the information of the intermediate reasoning steps.

Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, pages 1351–1362

2 Skill tree in arithmetic reasoning

We take arithmetic reasoning as the domain for our exploration because it allows us to synthesize questions systematically, as we show in this paper, which helps examine a model's composition ability in a controlled manner. Furthermore, the arithmetic reasoning ability of neural models has gained much attention as modern large language models still struggle with this problems (Rae et al., 2021).

Specifically, we use multi-hop arithmetic reasoning problems as follows:

Question: `A=1, B=2, C=A+2, C=?`
Answer: `3`

Here, the value assigned to the variable `C` is asked.

2.1 Compositionality in multi-hop symbolic reasoning

In this study, we specifically focused on three dimensions: systematicity, productivity, and substitutivity (Hupkes et al., 2020). According to these, we evaluate how well neural models achieve compositional generalization.

Systematicity refers to the ability of combining known different concepts into a more complex concept, i.e., structural composition. To evaluate this ability in models, we first trained with several types of primitive operations (e.g., addition; `A=1+2,A=?` and selection; `A=1,B=2,B=?`). Then, we measured the performance in solving problems consisting of combinations of primitives (e.g., `A=1+2,B=2+3,B=?`).

Productivity refers to the ability to solve longer/complex problems based on shorter/simpler ones. To evaluate this ability in models, we first trained with a short version of a formula (e.g., `A=1+2,B=2+3,B=?`). Then, we measured the performance in solving longer problems (e.g., `A=1+2,B=2+3,C=3+4,C=?`).

Substitutivity refers to the ability to keep the performance even if a particular constituent in a problem is replaced with another (unseen) constituent (i.e., lexical composition). To evaluate this ability in models, we conduct several experiments changing the variable characters between training and test (e.g., train with `A=1+2,A=?`; then evaluated with α`=1+2,` α`=?`).

2.2 Dataset configurations

Typical symbolic reasoning (e.g., procedural programming, assembly language) consists of at least three primitive symbol manipulations: assignment (`a=2`), arithmetic operation (`1+2`), and reference (`a=?`). With this in mind, our dataset is generated by combining the following five basic formulas: (i) `A=1` (assignment), (ii) `A=B` (reference & assignment), (iii) `A=1+2` (arithmetic operation & assignment), (iv) `A=B+2` (arithmetic operation & assignment & reference), (v) `A=?` (reference). The detailed properties are explained in Section 3.

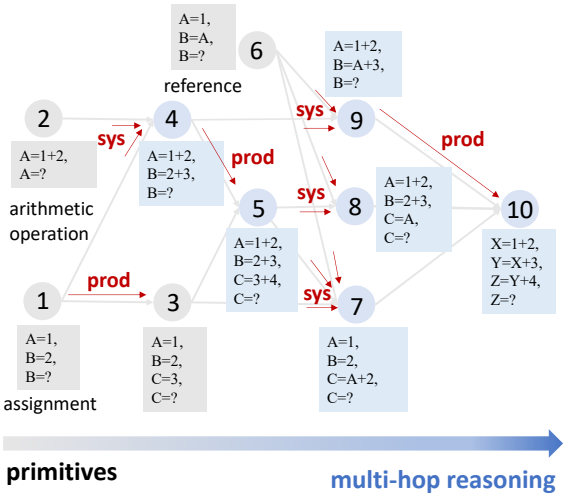

Figure 2: Skill tree to evaluate compositional generalization. The data format of primitive operations is gray and others (complex formulas composed of combinations of primitive operations) are blue .

2.3 Skill tree evaluations

We preliminarily observed that compositionally generalizing complex multi-hop arithmetic reasoning was difficult for neural seq2seq learners (the 1,2,6→9 setting in Section 4). Building on this fact, this study questions what type of composition made it hard for the neural models. To answer this, we designed a *skill tree on compositionality* that organizes the (hierarchical) complexity levels of symbolic reasoning.[2] Evaluating the models using problems with different complexity of composition in a step-wise manner, we elucidate the exact weakness of neural seq2seq models in multi-hop symbolic reasoning.

Specifically, we designed ten versions of symbolic reasoning problems. The hierarchical relationship of their task levels is illustrated in Figure 2; in this skill tree, each vertex, i.e., domain, corresponds to different task settings with different

[2]The term "skill tree" refers to a visualization method of step-by-step learning in the field of pedagogy (Tondello and Nacke, 2019), distinct from the "tree" in graph theory.

Task	Type	base ZA	base WA	large ZA	large WA	x-large ZA	x-large WA
1,2	sys.	42.0	82.1	35.2	89.8	51.7	96.7
→4	+subst.	39.1	80.4	33.2	89.4	50.7	96.7
2,3	sys.	33.6	75.4	32.1	85.6	35.9	94.7
→5	+subst.	33.6	77.0	31.5	87.2	36.5	94.9
2,3,6	sys.	40.8	76.5	39.1	87.7	40.5	94.6
→8	+subst.	39.3	74.7	37.5	86.4	39.2	94.9
2,3,6	sys.	56.5	79.7	51.0	83.7	58.1	94.2
→7	+subst.	57.6	80.2	51.1	85.8	56.7	95.1
1,2,6	sys.	24.1	28.2	23.1	29.3	27.5	32.6
→9	+subst.	25.8	28.0	24.1	31.7	28.5	34.7
7,8	sys.	23.6	21.3	25.3	25.9	22.3	28.2
→10	+subst.	22.3	21.6	24.4	26.4	23.0	30.2
1	prod.	100.0	100.0	100.0	100.0	100.0	100.0
→3	+subst.	100.0	100.0	100.0	100.0	100.0	100.0
4	prod.	100.0	100.0	100.0	100.0	100.0	100.0
→5	+subst.	99.9	99.9	100.0	100.0	100.0	99.9
9	prod.	57.0	59.3	61.7	63.8	60.6	62.7
→10	+subst.	58.4	60.9	62.2	64.1	59.5	64.5

Table 1: Average accuracies in the experiment with 2 different seeds. The "Task" column exhibits (train→test) domains corresponding to the skill-tree (Figure 2). The "Type" column shows the targeted compositionality type in each setting; here, "sys.," "prod," and "subst." denote the systematicity, productivity, and substitutivity generalizations, respectively.

complexity, and edges represent the hierarchical complexity levels.

By adequately selecting a particular combination of training and test domains, we evaluated the compositional generalization ability of the models from various perspectives. Here, the arithmetic expressions used in the test domain are a combination of those in training domains, creating a semi-order relationship in the skill tree. For example, using the settings 1 (`A=1+2,A=?`) and 2 (`A=1,B=2,B=?`) as a training set, and 4 (`A=1+2,B=2+3,B=?`) as a test set, one can evaluate the model's systematicity generalization towards the arithmetic operations (`a+b`) and assignments (`A=`i`,B=`j`,B=?`).

3 Experimental settings

3.1 Data

Dataset: In each experimental setting, we refer to the training domains as $\mathcal{D}_{\text{train}} = \{d_{\text{train}1}, \cdots d_{\text{train}k}\}$ and the test domain as d_{test}. Each domain has 100,000 training data and 3,200 test data; these are randomly generated, and there is no overlapped instance. When the training domain consisted of multiple domains, we used the union of the training data in $\mathcal{D}_{\text{train}}$. In addition, when the training domain is not primitive operations (1, 2, 3, and 6 in Figure 2), we further added the primitive operation data related to the training domain (Appendix A) into the training data.

Arithmetic expressions: As introduced in Section 2, the input is a sequence of arithmetic expressions. Formally, each expression is in the format of `a=`n or `a=`n{`+,-,max,min`}m except that the final expression asks the number assigned to a specified variable (`b=?`). Here, `a` and `b` are a member of a variable name set Σ; n and m are a member of the variable name set or number set $\Sigma \cup \mathcal{N}$. Specifically, Σ consists of 21 alphabets, and N consists of the integer from 0 to 99. The symbol `=` indicates that the result of the left-hand side is substituted into the right-hand side. The operations (`+`, `-`, `max`, and `min`) correspond to arithmetic addition, subtraction, max (returning the larger of its left and right numbers), and min (returning the smaller of its left and right numbers).

The questions are designed so that the answer is unique, and depending on the problem set-up, may include mathematical expressions that are not directly related to the final answer, i.e., distractors. The order of the equations is arbitrary; the first equation should not necessarily be calculated first.

Substitivity test: In each experimental setting, we evaluate the substitutivity generalization performance of the model under the situation where the variable names are replaced with unseen ones, e.g., training with `a=1+2,a=?`; then evaluating with α`=2+4,`α`=?`. In this setting, we replaced each variable name in the test set with one of five alphabets that do not overlap with the training ones.

3.2 Trainig and test

Training: The training stops when the accuracy on the validation dataset does not increase in successive five epochs or until the validation accuracy reaches 100%. Checkpoints with the highest accuracy in the validation dataset are used for evaluation. Note that among the experiments, the accuracy in the training domain reached at least 99.5%; this indicates that the primitive operations were learnable for the models. Detailed settings for training are described in Appendix A.

Evaluation metrics: The accuracy is calculated by the test data in the test domain d_{test}. Here, we used two metrics: (i) zero-shot accuracy (ZA) and (ii) weighted average of accuracies (WA) to measure the efficiency of learning (Talmor et al., 2020). In measuring WA, a model was further trained using the training set in the test domain d_{test}; then, the

weighted average of accuracies at every update was calculated (details are in Appendix B).

3.3 Models:

We used three different sizes (base, large, and xl) of T5 (Raffel et al., 2020), which is a widely used pre-trained seq2seq model in numerical reasoning tasks (Pal and Baral, 2021; Chung et al., 2022; Yang et al., 2021). Note that we began our training using the models with learned parameters. We also evaluated BART variants and randomly initialized models in Appendix C.

4 Experiments and results

We adopted nine combinations of training and test domains as shown in the first column of Table 1 (training domains→test domain). Six of them test the systematicity generalization and the other three test productivity generalization. In each setting, we further tested substitutivity generalization ability using the test domain data with a different variable name set (e.g., α instead of `A`) to that used in the training domain.

Table 1 shows the overall results. We observed the following four trends:

- Systematicity generalization was more difficult than productivity generalization.
- Even in the simple composition (the setting 1,2→4), the models struggle with generalization from zero or few examples.
- Models achieved substitutivity generalization.
- Model size did not incur substantial performance difference.

We identified that the systematicity generalization of reference and arithmetic operations (setting $2, 3 \rightarrow 5$; from `A=1,B=2,C=3,C=?` and `A=1+2,A=?` to `A=1+2,B=2+3,C=4+5,C=?`) was a simple setting, yet difficult to solve (refer to Appendix D for results on other tasks.). To better understand why neural models struggle with this setting, we decomposed the complexity of this setting and analyzed the model performance. Note that Kim and Linzen (2020) also suggested that neural models lack systematicity generalization ability in the context of semantic parsing; our results corroborate their findings from the context of arithmetic multi-hop reasoning.

Is this difficulty specific to *arithmetic* symbolic reasoning? We experimented with the same setting except that the four arithmetic operations are replaced with string operations (`join`, `reserveJoin`, `strSub`, and `stackJoin`; details are in Appendix E.1). The notable difference between arithmetic and string operations is that the string operation could be achieved by only copying selective elements in the input (e.g., `12+34=1234`), while arithmetic operation requires the models to access the arithmetic knowledge stored in their internals (e.g., `1+2=3`) and generate new information not stated in the input context (e.g., 3).

Setting	base ZA	base WA	large ZA	large WA	x-large ZA	x-large WA
2,3→5	33.6	75.4	32.1	85.6	35.9	94.7
String	37.3	94.1	66.1	98.4	86.9	99.3
Steps	26.2	82.1	36.1	89.4	33.7	96.4

Table 2: Ablation study with the 2,3→5 (vanilla) setting. "String" refers to the setting where string operations are used instead of arithmetic operations. "Step" denotes the setting generating intermediate steps.

Larger models tended to overcome the weakness in composition with string operations (e.g., the accuracy of 86.9 in zero-shot evaluation with the x-large model), while they struggled with arithmetic operations. This suggests that the major difficulty in systematicity was in **the access to the arithmetic knowledge (e.g., 1+1=2)**.

Does scratchpad training alleviate the difficulty? Existing studies suggested that showing the intermediate step (scratchpad-style training/inference) improves the multi-hop reasoning ability of neural models (Wei et al., 2022). We tested whether such an explicit generation of intermediate information alleviates the difficulty faced in the previous analysis. Specifically, we trained models with intermediate steps (e,g. `A=1+2,B=2+3,B=?; B=2+3,B=5`. Details are in Appendix E.2) during training.

The accuracy was calculated by the exact match of the answer and intermediate steps (the steps are designed to be uniquely determined). The performance gain due to explicating the intermediate steps was limited (Table 2), at least with our T5-based models. This shows that, in our carefully controlled setting, merely employing the scratchpad-style generation is not substantially effective.

5 Analysis

We conduct a more in-depth analysis of compositional generalization difficulties from another

Complexity dimensions	base		large		x-large		Average
	ZA	WA	ZA	WA	ZA	WA	
Δ#variables	0.098	-0.098	0.488	-0.293	-0.098	-0.488	-0.065
Δ#numbers	0.059	0.265	-0.088	0.647	0.206	0.677	0.294
Δ#operations	-0.507	-0.338	-0.338	0.169	-0.338	0.169	-0.197
Δ#reference	-0.655	-0.655	-0.393	-0.655	-0.655	-0.655	**-0.611**

Table 3: Spearman's rank correlation coefficient between the increase of training–test arithmetic complexity and the compositional generalization performance (accuracy) across the nine settings listed in Table 1. A negative score indicates that the greater the training–test discrepancy in its dimension, the more difficult compositional generalization is. In the case that there are multiple training domains, the maximum value among them is used.

perspective—complexity of arithmetic expressions. Specifically, for each pair of training and test domains listed in Table 1 (e.g., 1,2→4), we quantified the increase of the complexity of arithmetic formulas from several aspects, e.g., how much the formula's number of variables increased in the test domain (setting of 4) compared to the training domain (setting of 1 and 2). Specifically, we focused on the increase of the number of variables (Δ#variables), numbers (Δ#numbers), operations (Δ#operations), and references (Δ#references) from the training to test domains. Here, "#reference" denotes the number of access to a particular variable on the right-hand of equations. For example, the Δ#references is 1 if the training data format is `A=`n`,B=`**`A`**`+`m and the test format is `A=`n`,B=`**`A`**`+`m`,C=`**`B`**`+`l. Then, we identified which dimension strongly relates to the compositional generalization difficulty.

We analyzed the macro trends between formula's complexity increase and the difficulty of generalization across the experimental settings. Table 3 shows Spearman's rank correlation coefficient between each complexity and the test-domain accuracy. We found a notable negative correlation in the Δ#reference; that is, the more references in the test domain compared to the training domain, the more difficult the compositional generalization becomes (the cases of 1,2,6→9 and 9→10 settings). Simply put, this reveals the difficulty of compositional generalization with *multi-hop* reasoning—retaining the results of a calculation and accessing them again for another calculation.

6 Related work

The analysis of the compositional generalization ability of neural models and arithmetic multi-hop reasoning problems have typically been studied separately; this study has merged these two directions. As for composition generalization analysis, several studies analyzed neural models using datasets such as SCAN (Lake and Baroni, 2018), COGS (Kim and Linzen, 2020), and CFQ (Keysers et al., 2020). These mainly focused on compositionality in the context of semantic parsing; the composition ability toward symbol manipulations (e.g., multi-hop arithmetic reasoning) is typically out of focus. As for arithmetic reasoning, neural models' abilities have been analyzed typically using benchmarks such as DROP (Dua et al., 2019). It has recently been reported that such dataset has superficial cues (Al-Negheimish et al., 2021), which made it unclear how much arithmetic reasoning neural model achieves; our study using a carefully controlled dataset contributed to the exact weakness of neural models in this context.

7 Conclusion

In this study, we have empirically investigated the arithmetic multi-hop reasoning ability of modern neural models through the lens of compositional generalization ability. To systematically analyze neural models' ability, we have defined a skill tree that organizes the (hierarchical) complexity levels of the multi-hop symbolic reasoning dataset.

Our experiments have revealed that the major weakness lies in systematicity, even with a relatively simple composition. Through the ablation studies, we also have found that difficulty in systematicity is pronounced in accessing knowledge that is not written in input but stored in models. Furthermore, even in training models with intermediate steps that explicate the composition, they struggle to capture systematicity. We also found the difficulty of multi-hop reasoning in compositional generalization. These highlight the exact weakness of neural models and encourage studies to overcome such limitations.

Limitations

In this work, we explored neural networks' ability to capture compositionality in symbolic arithmetic reasoning in hopes that it may lead to future improvements in more general reasoning. However, arithmetic reasoning may not necessarily generalize to natural language tasks. Furthermore, we explored several aspects of multi-hop arithmetic reasoning, but these were chosen from a relatively human-centric perspective, and models may suffer from unforeseen other difficulties. Finally, while we found several patterns in how model performance degrades, it is difficult to aggregate this into a full picture of what a model can and cannot do. Further experiments are needed to gain a more complete understanding of model performance

Acknowledgements

We thank four anonymous reviewers who provided valuable feedback. We would like to also appreciate the member of Tohoku NLP Group for their cooperation in conducting this research. This work was supported by JST CREST JPMJCR20D2 and JSPS KAKENHI Grant Number JP22H00524, 21K21343.

References

Hadeel Al-Negheimish, Pranava Madhyastha, and Alessandra Russo. 2021. Numerical reasoning in machine reading comprehension tasks: are we there yet? In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9643–9649, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3882–3890. International Joint Conferences on Artificial Intelligence Organization. Main track.

Artur S. d'Avila Garcez and Luís C. Lamb. 2020. Neurosymbolic AI: the 3rd wave. *CoRR*, abs/2012.05876.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. 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 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, Online. Association for Computational Linguistics.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? (extended abstract). In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5065–5069. International Joint Conferences on Artificial Intelligence Organization. Journal track.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations*.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Gary F Marcus. 2003. *The algebraic mind: Integrating connectionism and cognitive science*. MIT press.

Kuntal Kumar Pal and Chitta Baral. 2021. Investigating numeracy learning ability of a text-to-text transfer

model. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3095–3101, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. 2022. Limitations of language models in arithmetic and symbolic induction. *CoRR*, abs/2208.05051.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew J. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. 2020. oLMpics-on what language model pre-training captures. *Transactions of the Association for Computational Linguistics*, 8:743–758.

Gustavo F. Tondello and Lennart E. Nacke. 2019. A pilot study of a digital skill tree in gameful education. In *Proceedings of the 3rd International Symposium on Gamification and Games for Learning - GamiLearn '19*. CEUR-WS.org, CEUR-WS.org.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Peng-Jian Yang, Ying-Ting Chen, Yuechan Chen, and Daniel Cer. 2021. Nt5?! training T5 to perform numerical reasoning. *CoRR*, abs/2104.07307.

A Training configurations

Training data: Data of primitive domains (grey domains in Figure 2) are also added to the training data. Specifically, data in the primitive domains that are reached by traversing the graph (Figure 2) from the training domain to the left is added. For example, when the training domain has the domain of 7, the data of primitive domains of 2, 3, and 6 are added[3]. Note that when the domain of 3 is included, the domain of 1 is not added. Additionally, as the objective of this study does not emphasize the generalization performance in arithmetic ability, the scope of numbers utilized in the test domain is adjusted to ensure that the upper limit of the answers in the test domain does not surpass that of the answers in the train domain.

Hyperparameter: We used the T5 models (v1.1) as pre-trained model[4]. We used three model sizes: base (250 million parameters), large (800 million parameters), and xl (3 billion parameters). Following T5 (Raffel et al., 2020) fine-tuning configurations, we use Adafactor (Shazeer and Stern, 2018) as the optimizer with a constant learning rate. Also, we specify the learning rate 1.0×10^{-5} for training and 5.0×10^{-5} in measuring the WA. The batch size is 32 for all experimental settings. We trained each model on NVIDIA A6000 (48GB memory), A100 (80GB memory). In addition, following previous research about numerical reasoning (Geva et al., 2020), we tokenize numbers in a digit-by-digit manner.

		base		large	
Task	Type	ZA	WA	ZA	WA
1,2	sys.	25.5	71.2	33.8	61.0
→4	+subst.	25.2	72.4	30.9	62.0
2,3	sys.	23.0	48.7	25.4	49.1
→5	+subst.	22.1	47.5	25.7	43.5
2,3,6	sys.	29.6	57.6	35.0	55.1
→8	+subst.	28.7	50.4	34.3	50.2
2,3,6	sys.	36.7	59.8	40.4	48.5
→7	+subst.	39.4	57.0	38.6	39.3
1,2,6	sys.	22.0	26.1	21.1	16.1
→9	+subst.	22.6	26.7	23.5	18.2
7,8	sys.	24.1	28.6	32.5	25.3
→10	+subst.	24.9	27.7	34.9	29.3
1	prod.	92.4	99.8	100.0	83.1
→3	+subst.	91.2	99.8	100.0	82.3
4	prod.	60.4	93.0	89.5	90.8
→5	+subst.	63.8	95.9	91.8	87.4
9	prod.	33.5	49.4	41.6	21.0
→10	+subst.	34.3	48.2	43.5	22.2

Table 4: Experimental results when BART is used as a pre-trained model. The "Task" column exhibits (train→test) domains corresponding to the skill-tree (Figure 2). The "Type" column shows the targeted compositionality type in each setting; here, "sys.," "prod," and "subst." denote the systematicity, productivity, and substitutivity generalizations, respectively.

B How to calculate WA

As described in Section 3, we used weighted average accuracy (WA). This metric quantifies the efficiency (ease) of generalization by assigning a high weight to accuracy in the early training stage. Specifically, the weights w_i (where i is the number of validation steps) were calculated using the following formulae:

$$w_i = -ai + w_{\max} \quad , \tag{1}$$

$$w_{\max} = \alpha w_{\min} \quad , \tag{2}$$

$$w_{\min} = \frac{2}{(N+1)(\alpha+1)} \quad , \tag{3}$$

$$a = \frac{(w_{\max} - w_{\min})}{N} \quad . \tag{4}$$

Here, N is the number of validation steps. α is a hyperparameter that determines how heavily the accuracy in the early stages is weighted. We set α to 1000 in all the experiments. Also, We used the first 100 validation steps to calculate WA, so specify $N = 100$ for all experiments. WA was calculated with accuracy on the held-out validation datasets in the test domain.

[3]Task 7,8→10 do not incorporate data from the primitive domains into the training data to evaluate if compositional generalization can be achieved solely from complex components.

[4]https://huggingface.co/docs/transformers/model_doc/t5v1.1

Task	base	large	x-large
1,2 (→4)	72.5	100.0	100.0
2,3 (→5)	63.3	56.1	99.9
1,3,6 (→7, 8)	68.6	67.9	67.5
1,2,6 (→9)	99.9	76.0	76.8
7,8 (→10)	48.6	37.3	24.0
1 (→3)	51.1	15.1	16.3
4 (→5)	100.0	99.9	99.9
9 (→10)	99.7	99.9	99.7

Table 5: Accuracy on the training held-out dataset when we start training the model from randomly initialized parameters. (When training began with parameters pre-trained on language, the accuracy was almost 100% in all settings.) Each training task includes the primitive operations required to solve train domain tasks as described in section A.

C Model variants

C.1 BART

To confirm the generality of our results obtained with the T5 models, we also conduct the same experiment using BART (Lewis et al., 2020). We use two model sizes[5]: base (140 million parameters) and large (400 million parameters). Table 4 shows the result using BART. The same tendency described in Section 4 was observed when BART was used as a language-pre-training model.

C.2 Training from scratch

We also experimented with the case where we started training from randomly initialized parameters to isolate the effect of the pre-training adopted in T5. We found that these initialized models failed to learn even primitive operations and in-domain tasks (Table 5), at least with the hyperparameter setting[6] used in this study. Thus, we did not proceed to their evaluation of compositionality generalization.

D Detailed analysis of T5-based models

Figure 3 shows the learning curves (accuracy on validation datasets) for all tasks demonstrated in Table 1. The graphs also show that the neural language model struggles to solve tasks that require compositional generalization in the beginning part of the training.

D.1 String operations ablation for all tasks

Table 6 shows the experimental results of the ablation study using string operations for all tasks.

D.2 Scratch pad evaluation for all tasks

Table 7 shows the experimental results of the ablation study using scratchpad for all tasks.

E Dataset details

Table 8 summarizes the characteristics of each task.

E.1 String operations

Table 9 shows the details of the string operations.

E.2 Scratch pad formulation

For the additional study using scratchpad in section 4, we generate problems with ing the intermediate steps. Scratchpad is in the form of straightforward one-by-one calculations of only those calculations necessary to find the target variable. The following is an example of scratchpad reasoning:

Question: `A=1+2, B=A+3, B=?`
Answer: `A=1+2;A=3;B=A+3;B=3+3;B=6`

[5] https://huggingface.co/facebook/bart-base, https://huggingface.co/facebook/bart-large

[6] Unlike in the training domain used for the language pre-trained model, we relax the training stopping criterion. We stopped training when the accuracy did not increase by **10** epochs in the validation dataset.

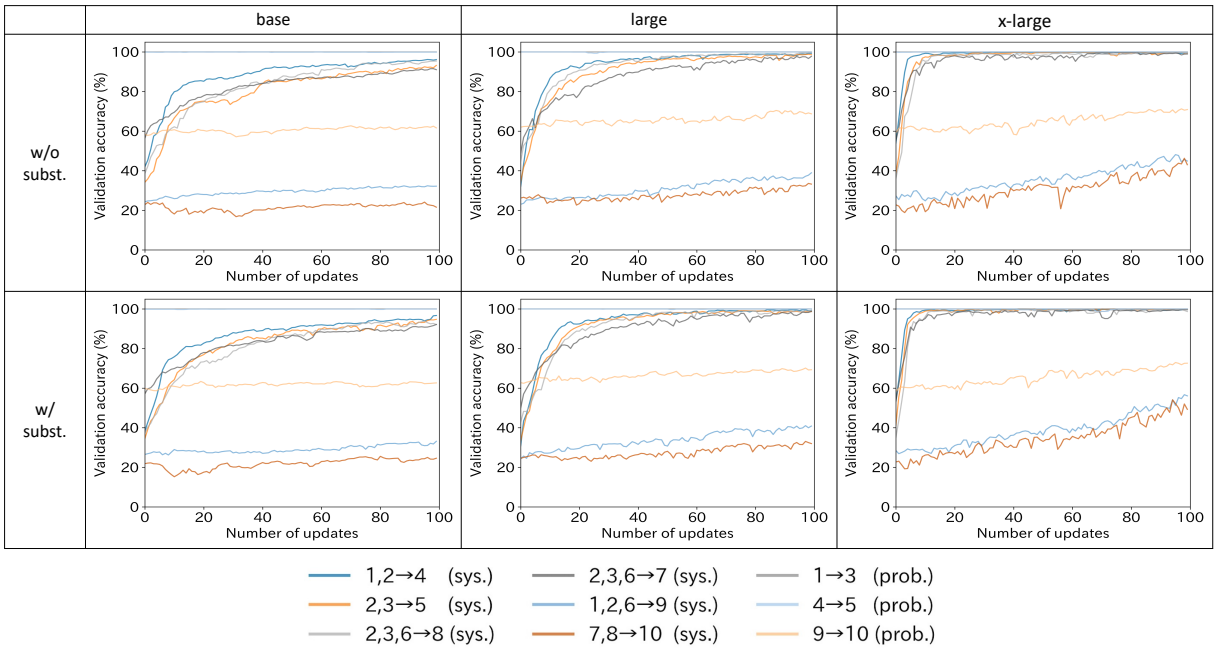

Figure 3: Learning curves of generalization test for all tasks.

		base		large		x-large	
Task	Type	ZA	WS	ZA	WS	ZA	WS
1,2	sys.	59.2	95.9	66.9	98.5	63.9	98.9
→ 4	+subst.	59.0	95.1	67.3	98.2	62.2	99.0
2, 3	sys.	37.3	94.1	66.1	98.4	86.9	99.3
→ 5	+subst.	35.9	93.5	65.2	98.6	84.2	99.0
2,3,6	sys.	59.6	93.9	71.5	98.0	52.5	97.2
→8	+subst.	56.3	92.8	67.1	98.0	52.3	97.0
2,3,6	sys.	34.3	91.0	46.3	93.6	32.0	95.7
→7	+subst.	34.3	91.3	48.1	93.6	34.2	96.5
1,2,6	sys.	13.2	34.3	11.6	38.1	12.3	59.6
→9	+subst.	12.7	34.9	13.0	39.0	12.4	58.8
7,8	sys.	26.2	61.1	20.0	61.7	21.3	67.4
→10	+subst.	26.4	61.4	19.9	60.0	20.5	68.5
1	prod.	100.0	100.0	100.0	100.0	100.0	100.0
→3	+subst.	100.0	100.0	100.0	100.0	99.9	100.0
4	prod.	100.0	100.0	99.9	100.0	100.0	100.0
→5	+subst.	100.0	100.0	100.0	100.0	100.0	100.0
9	prod.	69.6	88.2	71.5	89.9	75.3	92.0
→10	+subst.	67.4	88.8	72.0	89.6	75.2	92.0

Table 6: Experiments result from string operation ablation. The "Task" column exhibits (train→test) domains corresponding to the skill-tree (Figure 2). The "Type" column shows the targeted compositionality type in each setting; here, "sys.," "prod," and "subst." denote the systematicity, productivity, and substitutivity generalizations, respectively.

		base		large		x-large	
Task	Type	ZA	WS	ZA	WS	ZA	WS
1,2	sys.	30.6	81.3	33.9	85.5	35.6	94.9
→4	+subst.	30.3	77.9	32.9	85.5	36.3	95.5
2,3	sys.	26.2	82.1	36.1	89.4	33.7	96.4
→5	+subst.	25.4	81.3	36.1	88.1	32.9	96.4
2,3,6	sys.	0.0	27.6	0.0	47.5	0.0	71.3
→8	+subst.	0.0	26.0	0.0	47.3	0.0	72.9
2,3,6	sys.	0.0	43.1	0.0	59.0	0.0	75.0
→7	+subst.	0.0	40.0	0.0	56.6	0.0	73.3
1,2,6	sys.	0.0	0.2	0.0	2.5	0.0	14.3
→9	+subst.	0.0	0.1	0.0	2.7	0.0	16.4
7,8	sys.	0.0	17.0	0.0	33.2	0.0	55.1
→10	+subst.	0.0	17.4	0.0	32.6	0.0	55.3
1	prod.	100.0	100.0	100.0	100.0	96.0	99.9
→3	+subst.	100.0	100.0	100.0	100.0	95.8	99.9
4	prod.	100.0	100.0	100.0	99.9	100.0	99.9
→5	+subst.	100.0	100.0	100.0	99.9	100.0	99.9
9	prod.	0.0	50.8	0.0	72.2	0.0	80.9
→10	+subst.	0.0	51.8	0.0	69.5	0.0	79.3

Table 7: Experiments result from scratchpad ablation. The "Task" column exhibits (train→test) domains corresponding to the skill-tree (Figure 2). The "Type" column shows the targeted compositionality type in each setting; here, "sys.," "prod," and "subst." denote the systematicity, productivity, and substitutivity generalizations, respectively.

Domains		Required premitives			Targeted composition		
Task	Example	Assign.	Arith.	Ref.	Sys.	Prod.	Subst.
1	A=1, B=2, B=?	✓					
2	A=1+2, A=?	✓	✓				
3	A=1, B=2, C=3, C=?	✓				✓	
3'	α=1, β=2, γ=3, γ=?	✓				✓	✓
4	A=1+2, B=2+3, B=?	✓	✓		✓		
4'	α=1+2, β=2+3, β=?	✓	✓		✓		✓
5	A=1+2, B=2+3, C=3+4, C=?	✓	✓		✓	✓	
5'	α=1+2, β=2+3, γ=3+4, γ=?	✓	✓		✓	✓	✓
6	A=1, B=A, B=?	✓		✓			
7	A=1, B=2, C=B+3, C=?	✓	✓	✓	✓		
7'	α=1, β=2, γ=β+3, γ=?	✓	✓	✓	✓		✓
8	A=1+2, B=2+3, C=B, C=?	✓	✓	✓	✓		
8'	α=1+2, β=2+3, γ=β, γ=?	✓	✓	✓	✓		✓
9	A=1+2, B=A+3, B=?	✓	✓	✓	✓		
9'	α=1+2, β=α+3, β=?	✓	✓	✓	✓		✓
10	A=1+2, B=A+3, C=B+4, C=?	✓	✓	✓	✓	✓	
10'	α=1+2, β=α+3, γ=β+4, γ=?	✓	✓	✓	✓	✓	✓

Table 8: Dataset configurations. The "Task" column exhibits (train→test) domains corresponding to the skill-tree (Figure 2). "Assign." "Arith." and "Ref." in "Required primitives" column mean primitive operations name, "assignment" "arithmetic operations" and "reference" (see subsection 2.2) The "Target composition" column shows the targeted compositionality type in each setting; here, "Sys.," "Prod," and "Subst." denote the systematicity, productivity, and substitutivity generalizations, respectively.

	Description	Example
`join`	String concatenation.	`12 + 34 = 1234`
`reverseJoin`	String concatenation + Reverse	`123 ^ 78 = 87321`
`strSub`	Deletion of duplicate characters. Return 0 if there are no duplicates.	`7873 - 73 = 87`
`stackJoin`	Select one character from the left side of each string alternately and return the combined string.	`12 * 34 = 1324`

Table 9: Detail of string operation.