Compositional generalization with a broad-coverage semantic parser

Pia Weißenhorn and **Lucia Donatelli** and **Alexander Koller**
Department of Language Science and Technology
Saarland Informatics Campus
Saarland University, Germany
`{piaw, donatelli, koller}@coli.uni-saarland.de`

Abstract

We show how the AM parser, a compositional semantic parser (Groschwitz et al., 2018), can solve compositional generalization on the COGS dataset. It is the first semantic parser that achieves high accuracy on both naturally occurring language and the synthetic COGS dataset. We discuss implications for corpus and model design for learning human-like generalization. Our results suggest that compositional generalization can be best achieved by building compositionality into semantic parsers.

1 Introduction

A growing body of recent research investigates *compositional generalization*, the ability of a semantic parser to predict the meaning of unseen sentences by recombining training instances in novel ways. Such generalization is thought to mimic the Principle of Compositionality (Partee, 1984), essential for human language learning and use. For example, COGS (Kim and Linzen, 2020), a dataset based on fragments of English, contains training instances with sentences semantically annotated with up to two recursive PPs; a semantic parser must then predict meaning representations for sentences with three or more recursive PPs (Table 1).

Previous work has shown that compositional generalization on COGS is a difficult and complex task. Intricate sequence-to-sequence (seq2seq) models, which achieve very high accuracy on broad-coverage semantic parsing tasks on naturally occurring language (Bevilacqua et al., 2021), achieve overall accuracy of 88% or less on COGS (Akyürek and Andreas, 2021; Csordás et al., 2021; Zheng and Lapata, 2021). Much of this accuracy is due to *lexical generalization*, tasks that test for generalization to new words in known structures (Sec. 2); when evaluated only on *structural generalization* cases that test novel structures such as the PP example above, the accuracy of most of these models drops to 10% or less.

In contrast, models that achieve high accuracy on synthetic compositional generalization datasets may not be able to generalize to naturally occurring language. For instance, Shaw et al. (2021) describe a synchronous grammar induction approach that achieves perfect accuracy on SCAN (Lake and Baroni, 2018), but has very low accuracy on corpora of naturally occurring text such as GeoQuery (Zelle and Mooney, 1996) and Spider (Yu et al., 2018). Similarly, the compositional LeAR parser (Liu et al., 2021) solves COGS with near-perfect accuracy and performs very well on other synthetic datasets, but has not been evaluated on corpora of naturally occurring text. This points to a fundamental tension between broad-coverage semantic parsing on natural text and the ability to generalize compositionally from structurally limited synthetic training sets (see also Shaw et al., 2021). To our knowledge, the only parser that does well on both is the CSL-T5 system of Qiu et al. (2022), which fine-tunes T5 using a complex data augmentation (DA) method involving synchronous grammars.

In this paper, we show that the AM parser (Groschwitz et al., 2018), a compositional semantic parser that achieves high accuracy across a range of different broad-coverage graphbanks (Lindemann et al., 2019; Donatelli et al., 2019), can also solve COGS at near-perfect accuracy. This high performance is due in large part to handling cases of structural generalization much better than the seq2seq models. The AM parser is thus the first semantic parser shown to perform accurately both on naturally occurring language and on COGS without requiring DA. Given that all semantic parsers that do well on COGS are either compositional (LeAR, AM parser) or perform compositionality-based DA (CSL-T5), we conjecture that building a semantic parser on the Principle of Compositionality is beneficial to solving compositional generalization. We discuss the challenge of structural, as opposed to lexical, generalization for future work on this task.

2 Compositional Generalization in COGS

Compositional generalization is the ability to determine the meaning of unseen sentences using compositional principles. Humans can understand and produce a potentially infinite number of novel linguistic expressions by dynamically recombining known elements (Chomsky, 1957; Fodor and Pylyshyn, 1988; Fodor and Lepore, 2002). For semantic parsers, compositional generalization requires systems to recombine parts of multiple training instances to predict the meaning of a single test instance by learning correct generalizations. Several synthetic datasets for evaluating compositional generalization now exist, notably SCAN (Lake and Baroni, 2018) and CFQ (Keysers et al., 2020).

COGS (Kim and Linzen, 2020) is a synthetic semantic parsing dataset in which English sentences must be mapped to logic-based meaning representations. It distinguishes 21 *generalization types*, each of which requires generalizing from training instances to test instances in a particular systematic and linguistically-informed way.

Lexical generalization cases (18 types) test how known grammatical structures are recombined with words that were not observed in these particular structures during training. For instance, the common noun "hedgehog" is only exposed to the model as subject at training time as part of an 'exposure example' sentence, but generalization requires object usage of the same word based on forming analogies to other common nouns seen in both positions. This is illustrated in Table 1.

Structural generalization cases (3 types) involve generalizing to linguistic structures that were not observed in training. The PP recursion example above is of this type: the COGS training set contains sentences and logic-based semantic representations with up two nested prepositional phrases. In-domain development and test sets also consist of sentences with PP nesting depth up to two, but the *generalization set* contains sentences with 3–12 nested PPs. Additional structural generalization includes CP recursion (predict deeply nested CPs when trained on shallow examples, similar to PPs) and "object PP to subject PP", where PPs modify only objects in training (e.g. "Noah ate *the cake on the plate.*") and only subjects at test time ("*The cake on the table* burned.").

Kim and Linzen themselves show that seq2seq models based on LSTMs and Transformers do not perform well on COGS, achieving exact-match accuracies below 35%. Intensive subsequent work has tailored a wide range of seq2seq models to the COGS task (Tay et al., 2021; Akyürek and Andreas, 2021; Conklin et al., 2021; Csordás et al., 2021; Orhan, 2021; Zheng and Lapata, 2021), but none of these have reached an overall accuracy of 90% on the overall generalization set. On structural generalization in particular, the accuracy of all these models is below 10%, with the exception of Zheng and Lapata (2021), who achieve 39% on PP recursion. By contrast, the compositional model of Liu et al. (2021) and the model of Qiu et al. (2022), which uses compositional data augmentation, achieve accuracies upwards of 98% on the full generalization set.

3 Parsing COGS with the AM parser

3.1 The AM parser

We adapt the broad-coverage AM parser to COGS. The AM parser (Groschwitz et al., 2018) is a compositional semantic parser that learns to map sentences to graphs. It was the first semantic parser to perform with high accuracy across all major graphbanks (Lindemann et al., 2019) and can achieve very high parsing speeds (Lindemann et al., 2020).

Instead of predicting the graph directly, the AM parser first predicts a graph fragment for each token in the sentence and a dependency tree that connects them (Fig. 1a). This dependency tree is then evaluated deterministically into a graph (Fig. 1b) using the operations of the *AM algebra*. The "Apply" (APP) operation fills an argument slot of a graph (drawn in red) by inserting the root node (drawn with a bold outline) of another graph into this slot; for instance, the APP$_s$ operation inserts the "boy" node into the ARG0 of "want". The "Modify" (MOD) operation attaches a modifier to a node; MOD$_m$ attaches the "manner-sound" graph to the "sleep" node. The dependency tree captures how the meaning of the sentence can be compositionally obtained from the meanings of the words.

AM parsing is done by combining a neural dependency parser with a neural tagger for predicting the graph fragments. We follow Lindemann et al. (2019) and rely on the dependency parsing model of Kiperwasser and Goldberg (2016), which scores each dependency edge by feeding neural represen-

Class : Type	Training	Generalization
Lexical: *Subj→Obj (common noun)*	A hedgehog ate the cake. $\star$cake(x_4); hedgehog(x_1) ∧ eat.agent(x_2,x_1) ∧ eat.theme(x_2,x_4)	The baby liked the hedgehog. $\star$baby(x_1); $\star$hedgehog(x_4); like.agent(x_2,x_1) ∧ like.theme(x_2,x_4)
Structural: *PP recursion*	Ava saw a ball in a bowl on the table. $\star$table(x_9); see.agent(x_1,Ava) ∧ see.theme(x_1,x_3) ∧ ball(x_3) ∧ ball.nmod.in(x_3,x_6) ∧ bowl(x_6) ∧ bowl.nmod.on(x_6,x_9)	Ava saw a ball in a bowl on the table on the floor. $\star$table(x_9); $\star$floor(x_{12}); see.agent(x_1, Ava) ∧ see.theme(x_1,x_3) ∧ ball(x_3) ∧ ball.nmod.in(x_3,x_6) ∧ bowl(x_6) ∧ bowl.nmod.on(x_6,x_9) ∧ table.nmod.on(x_9,x_{12})

Table 1: One example of a lexical and a structural generalization type from the COGS dataset.

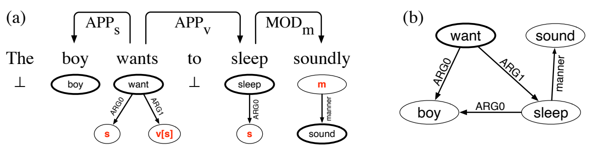


Figure 1: (a) AM dependency tree with (b) its value.

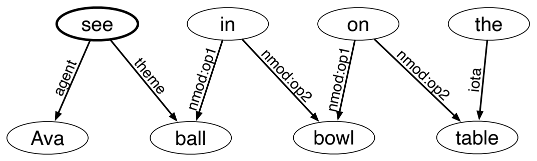


Figure 2: Logical form to graph conversion for "Ava saw a ball in a bowl on the table" (cf. Table 1).

tations for the two tokens to an MLP. We train the parser using the setup of Groschwitz et al. (2021), which does not require explicit annotations with AM dependency trees.

3.2 AM parsing for COGS

We apply the AM parser to COGS by converting the semantic representations in COGS to graphs. The conversion is illustrated in Fig. 2.

Given a logical form of COGS, we create a graph that has one node for each variable x_i and each constant (e.g. Ava). If a variable appears as the first argument of an atom of the form pred.arg(x, y), we assign it the node label pred in the graph. We also add an edge from x to y with label arg. E.g. see.agent(x_1, Ava) turns into an 'agent' edge from 'see' to 'Ava'. Each *iota term* $\star$noun(x_{noun}) is treated as an edge from a node for the preceeding "the" token to the respective noun node. Preposition meaning bowl.nmod.on(x_6, x_9) is represented as a node (labeled 'on') with outgoing edges to the two arguments/nouns ('nmod.op1' to "bowl", 'nmod.op2' to "table"). By encoding the logical form as a graph, we lose the ordering of the conjuncts. The 'correct' order is restored in postprocessing. More details and graph conversion examples are in Appendix C.

4 Experiments on COGS

4.1 Experimental setup

We evaluate the AM parser on COGS and compare its accuracy against a number of strong baselines. We follow standard COGS practice and evaluate on both the (in-distribution) test set and the generalization set. We report exact match accuracies averaged across 5 training runs with their standard deviations.

Training regime. In addition to the regular COGS training set ('train') of 24,155 training instances, we also report numbers for models trained on the extended training set 'train100' of 39,500 instances (Kim and Linzen, 2020, Appendix E.2). These training sets allow to test 1-shot (train) or 100-shot (train100) lexical generalization. For instance, for the "hedgehog" example in Table 1, train contains *exactly one* sentence with this noun, whereas there are 100 different sentences with "hedgehog" in train100 (all in subject position). As this change can only be done for lexical generalization (tied to specific lexical items), structural generalization is not directly modulated by a training set change.

Compositional models. We train the AM parser on the COGS graph corpus (cf. Section 3.2). Most hyperparameter values come from Groschwitz et al. (2021)'s training setup for AMR to make overfitting to COGS less likely; see Appendix A for details.

The AM parser either receives pretrained word embeddings from BERT (Devlin et al., 2019) ('AM+B') or learns embeddings from the COGS

		train		train100	
		Test	Gen	Test	Gen
seq2seq	Kim and Linzen 2020	96	35	94	63
	Csordás et al. 2021	100	81	-	75.4
	Akyürek and Andreas 2021	-	83	99	84.5
	Zheng and Lapata 2021 †	-	89	-	-
compositional	Qiu et al. 2022	-	**99.5**	-	-
	Liu et al. 2021: LeAR[1]	-	98.9±0.9	-	-
	AM	100	59.9± 2.7	100	91.1±2.3
	AM+dist	100	62.6±10.8	100	88.6±4.9
	AM+B †	100	79.6± 6.4	100	93.6±1.4
	AM+B+dist †	100	78.3±22.9	100	**98.4**±0.9

Table 2: COGS exact match scores. †) models use pretraining.

data only ('AM'). We run the training algorithm with up to three argument slots to enable the analysis of ditransitive verbs. For evaluation, we reverse graph conversion to reconstruct the logical forms.

To handle PP recursion, we hypothesize that explicit distance information between tokens could help the AM parser: COGS eliminates potential PP attachment ambiguities and assumes that each PP modifies the noun immediately to its left. Instead of passing only the representations of the potential parent and child node to the edge-scoring model, we also pass an encoding of their relative distance in the string (Vaswani et al., 2017), yielding the AM parser models with the "+dist" suffix. Distance information is then available as an explicit feature for *any* dependency edge decision, and the neural model learns how to weight this feature for different edges.

Finally, we report evaluation results for LeAR, the compositional COGS parser of Liu et al. (2021). LeAR learns to predict trees of corpus-specific algebraic operations using reinforcement learning with an intricate training setup.

4.2 Results

The results are summarized in Table 2. Gray numbers are taken from original papers; black numbers we reproduced in separate experiments. Table 3 shows results by structural and lexical generalization type. See Appendix B for details.

Compositional models solve COGS. We find that when trained on 'train100', the modified AM parser solves COGS with near-perfect accuracy. The evaluation results in Table 2 suggest a clear split between compositional and seq2seq models, with both compositional models outperforming all seq2seq models. This split becomes even clearer when we distinguish different generalization types. On the three structural generalization types, no seq2seq model has an accuracy above 40%, whereas both LeAR and AM+B+dist still achieve near-perfect accuracy.

[1]All LeAR numbers are based on our reproduction of their COGS evaluation; they report an accuracy of 97.7.

PP vs. CP recursion. A closer error analysis on PP recursion reveals (as hypothesized) that the accuracy of the AM+B parser degrades with increasing PP depth. The AM+B+dist parser maintains a high accuracy across all embedding depths.

There is an interesting asymmetry between the behavior of the AM parser on PP recursion and CP recursion: The accuracy of AM+B is stable across recursion depths for CP recursion, and the distance feature is only needed for PPs. This can be explained by the way in which the AM parser learns to incorporate PPs and CPs into the dependency tree: it uses APP edges to combine verbs with CPs, which ensures that only a single CP can be combined with each sentence-embedding verb. By contrast, each NP can be modified by an arbitrary number of PPs using MOD edges. Thus a confusion over attachment is only possible for PPs.

Effect of training regime. Parsers on COGS are traditionally not allowed any pretraining (Kim and Linzen, 2020), in order to judge their ability to generalize from limited observations. We see in the experiments above that the use of pretrained word embeddings helps the AM parser achieve accuracy parity with LeAR, but is not needed to outperform all seq2seq models on 'train100'.

Training on 'train100' helps the AM parser more than any other model in Table 2. The difference between its accuracy on 'train' and 'train100' is due to lexical issues: we found that when trained on 'train', the AM parser typically predicts the correct delexicalized formulas and then inserts an incorrect but related constant or predicate symbol.

For example, when tested on common nouns, "kennel" may be used instead of "hedgehog"; when tested on unaccusative to transitive generalization, the model may choose another verb seen commonly in that pattern instead of the target verb (e.g. "value" instead of "shatter").

We ablate the different model components (pretrained BERT embeddings, +dist) and training setups (train100 vs. train) in Table 3. Trained on

		Class	STRUCTURAL			LEXICAL	
		Gen. type	Obj to Subj PP	CP recursion	PP recursion	mean of 18 other types	Overall
compositional	AM+B+dist	train100	78	100	99	99	98
	AM+B	train100	49	100	41	99	94
	AM+B+dist	train	72	100	97	76	78
	AM+B	train	59	100	36	82	80
	AM+dist	train	26	100	98	61	63
	AM	train	38	100	61	59	60
	LeAR	train	93	100	99	99	99
seq2seq	Kim and Linzen 2020	train	0	0	0	42	35
	Akyürek and Andreas 2021	train	0	0	1	96	82
	Zheng and Lapata 2021	train	0	12	39	99	89
	Kim and Linzen 2020	train100	0	0	0	73	63
	Csordás et al. 2021	train100	0	0	0	88	75

Table 3: Exact match accuracies on the individual generalization types.

'train', AM+B+dist achieves a mean accuracy on structural generalization cases of 89.6 (compared to 92.1 for 'train100'), whereas the mean accuracy on lexical generalization cases drops to 76. This again illustrates that the larger training set compensates for a lexical weakness in the AM parser rather than a structural one. Even without BERT and trained on 'train', AM+dist gets 74.6 on structural cases, drastically outperforming the seq2seq models.

5 Conclusion

The AM parser is the first compositional semantic parser to solve COGS and achieve high accuracy on naturally occurring language.[2] Particularly on complex structural generalization cases, compositionality-based parsers seem to outperform seq2seq models systematically. By contrast, lexical generalization cases are solved easily by most models and do not require a compositionality bias. We suggest that future corpus design and evaluation focus on model accuracy for structural generalization types; an extension to COGS that incorporates a greater variety of these types would allow more insight on the overall task.

Though synthetic datasets like COGS allow focused probing parser performance on specific linguistic phenomena, it remains unclear exactly how accurate performance on such datasets transfers to naturally occurring language, and vice-versa. Another strand of future work is thus extending the broad-coverage AM parser to more compositional generalization datasets. While COGS offers a good starting point to test multiple types of both lexical and structural generalization similar to what is attested for humans, other datasets offer insight into generalization less clearly connected to human linguistic abilities (e.g. CFQ; Keysers et al., 2020) but important for generalization abilities more generally. Additional assessment of models' generalization performance ought to combine broad-coverage parsing and focused evaluation with hand-crafted datasets in a systematic way, yet to be defined.

[2]Our code is available at `https://github.com/coli-saar/am-parser`.

Acknowledgments

We thank Yuekun Yao for his insights from related experiments on COGS and Jonas Groschwitz and Matthias Lindemann for their advice on adapting the AM parser to COGS. We also thank the reviewers for their helpful comments. This research was funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project KO 2916/2-2.

References

Ekin Akyürek and Jacob Andreas. 2021. Lexicon learning for few shot sequence modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4934–4946, Online. Association for Computational Linguistics.

Michele Bevilacqua, Rexhina Blloshmi, and Roberto Navigli. 2021. One SPRING to rule them both: Symmetric AMR semantic parsing and generation without a complex pipeline. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-21)*, volume 35, pages 12564–12573. AAAI Press.

Noam Chomsky. 1957. *Syntactic Structures*. De Gruyter Mouton.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3322–3335, Online. Association for Computational Linguistics.

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 619–634, Online and Punta Cana, Dominican Republic. 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.

Lucia Donatelli, Meaghan Fowlie, Jonas Groschwitz, Alexander Koller, Matthias Lindemann, Mario Mina, and Pia Weißenhorn. 2019. Saarland at MRP 2019: Compositional parsing across all graphbanks. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning*, pages 66–75, Hong Kong. Association for Computational Linguistics.

Jerry A. Fodor and Ernest Lepore. 2002. *The Compositionality Papers*. Oxford University Press.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71.

Jonas Groschwitz, Meaghan Fowlie, and Alexander Koller. 2021. Learning compositional structures for semantic graph parsing. In *Proceedings of the 5th Workshop on Structured Prediction for NLP (SPNLP 2021)*, pages 22–36, Online. Association for Computational Linguistics.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics.

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 (ICLR)*.

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*, pages 9087–9105, Online. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882, Stockholmsmässan, Stockholm Sweden. PMLR.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2019. Compositional semantic parsing across graphbanks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4576–4585, Florence, Italy. Association for Computational Linguistics.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2020. Fast semantic parsing with well-typedness guarantees. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3929–3951, Online. Association for Computational Linguistics.

Chenyao Liu, Shengnan An, Zeqi Lin, Qian Liu, Bei Chen, Jian-Guang Lou, Lijie Wen, Nanning Zheng, and Dongmei Zhang. 2021. Learning algebraic recombination for compositional generalization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1129–1144, Online. Association for Computational Linguistics.

A. Emin Orhan. 2021. Compositional generalization in semantic parsing with pretrained transformers. *Computing Research Repository (CoRR)*, arXiv: 2109.15101.

Barbara H. Partee. 1984. Compositionality. In *Varieties of Formal Semantics: Proceedings of the 4th Amsterdam Colloquium, September 1982*, volume 3, pages 281–311. Foris Publications, Dordrecht.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Paweł Krzysztof Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022. Improving compositional generalization with latent structure and data augmentation. In *Proceedings of NAACL*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics.

Yi Tay, Mostafa Dehghani, Jai Prakash Gupta, Vamsi Aribandi, Dara Bahri, Zhen Qin, and Donald Metzler. 2021. Are pretrained convolutions better than pretrained transformers? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4349–4359, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Curran Associates, Inc.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Hao Zheng and Mirella Lapata. 2021. Disentangled sequence to sequence learning for compositional generalization. *Computing Research Repository (CoRR)*, arXiv: 2110.04655. To appear at ACL2022.

A Training details of the AM parser

Hyperparameters. For the AM parser, we primarily copy hyperparameter values from the AMR experiments of Groschwitz et al. (2021). This helps prevent overfitting on COGS, but we also note that hyperparameter tuning for compositional generalization datasets can be difficult anyways since one can typically easily achieve perfect scores on an indoman dev set. Copied values include for instance the number of epochs (60 due to supervised loss for edge existence and lexical labels), the batch size, the number and dimensionality of neural network layers and not using early stopping (but selecting best model based on per epoch evaluation metric on the dev set). Choosing 3 sources has worked well on other datasets (Groschwitz et al., 2021) and we adopt this hyperparameter choice. We note that with ditransitive verbs (i.e. verbs requiring NPs filling agent, theme, and recipient roles) present in COGS we need at least three sources anyway to account for these.

Deviations from Groschwitz et al. (2021)'s settings. For training on train (but not train100), we set the vocabulary threshold from 7 down to 1 to account for the fact that the lexical generalizations rely on a single occurrence of a word in the training data; on train100 we keep 7 as a threshold since trigger words (e.g. "hedgehog") occur 100 times. For word embeddings, we either use BERT-Large-uncased (Devlin et al., 2019) like Groschwitz et al. (2021) or learn embeddings from the dataset only (embedding dimension 1024, same as for the BERT model). We decrease the learning rate from 0.001 to 0.0001: we observed that the learning curves are still converging very quickly and hypothesize that COGS training set might also be easier than the AMR one used in Groschwitz et al. (2021).

We use the projective A* decoder (Lindemann et al., 2020, §4.2): in pre-experiments this showed better results. In addition, it makes comparison to related work (such as LeAR by Liu et al. (2021)) easier which uses only projective latent trees. We use supervised loss for edge existence and lexical labels.

Relative distance encoding. For the relative distance encodings we use sine-cosine interleaved encoding function introduced by Vaswani et al. (2017, §3.5) and as input to it use the relative distance $dist(i, j) = i - j$ between sentence positions i and j. We use a dimensionality of 64 for the distance encodings (d_{model} in Vaswani et al. (2017) is 512). These distance encodings are then concatenated together with the BiLSTM representations for possible heads and dependents used in the standard Kiperwasser and Goldberg (2016) edge scoring model. This constitutes the input to the MLP emitting a score for each token pair. These models have the suffix 'dist' in the tables.

Runtimes. Training the AM parser took 5 to 7 hours on train with 60 epochs and 6 to 9.5 hours on train100. In general, training with BERT took longer than without, same holds for adding relative distance encodings. Inference with a trained model on the full 21k generalization samples took about 15 minutes using the Astar decoder with the 'ignore aware' heuristic. All AM parser experiments were performed using Intel Xeon E5-2687W v3 10-core processors at 3.10Ghz and 256GB RAM, and MSI Nvidia Titan-X (2015) GPU cards (12GB).

Number of parameters. For their models, Kim and Linzen (2020) tried to keep the number of parameters comparable (9.5 to 11 million) and therefore rule out model capacity as a confound. The number of trainable parameters of the AM parser model used is 10.7 to 11.5 million (lower one is with BERT, higher without. Impact of relative distance encoding is rather minimal: < 17k), so the improved performance is not just due to a higher number of parameters.

Dev set performance. For compositional generalization datasets, it is relatively easy to get (near) perfect results on the (in domain) dev/test sets. We observe this too: all AM parser models had an exact match score of at least 99.9 on the dev set and at least 99.8 on the (in distribution) test set.

Evaluation procedure. Kim and Linzen (2020) do not provide a separate evaluation script but use (string) exact match accuracy on the logical forms as the main evaluation metric. This metric requires models to learn the 'correct' order of conjuncts: even if a logically equivalent form with a different order of conjuncts would be predicted, string exact match would count it as a failure. In lack of an official evaluation script we implemented our own evaluation script to compute exact match.

B Evaluation details

For descriptions of the generalization types we refer to Kim and Linzen (2020, §3 and Fig. 1).

AM parser. Full results for the 8 AM parser configurations (two types of embeddings, two training sets, presence/absence of distance encodings) are displayed in Table 4. Averages and standard deviations were computed across 5 runs for each configuration. For the AM+B+dist configuration trained on the smaller train set, one outlier run was observed with 39.9% overall generalization accuracy, and the other four runs ranging from 76.4% to 96.6%. This outlier therefore greatly contributed to the high variance for this configuration.

LeAR. Due to our reproduction experiment, we can report a breakdown by generalization type for Liu et al.'s LeAR model, displayed in Table 5. We observed that the LeAR model skips 22 sentences in the generalization set due to out-of-vocabulary tokens.[3] We include these sentences in the accuracy computation (as failures) for the generalization set. The published LeAR code does not convert its internally used representation back to logical forms, therefore we evaluate on the logical forms like it is done for other models, but have to rely on accuracy computation done in the LeAR code for the internal representation. From inspecting the published code,[4] LeAR makes the preprocessing choice to ignore the contribution of the definite determiner, treating indefinite and definite NPs equally, resulting in a big conjunction without any iota ('$\star$') prefixes.

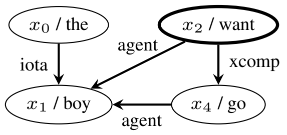
Model numbers copied from other papers. Kim and Linzen (2020) provide three baseline models, among which the Transformer model reached the best performance on train and train100. Per generalization type results can be found in their Appendix F (Table 5 on page 9105) from which we report the Transformer model numbers.

The strongest model of Akyürek and Andreas (2021) is 'Lex:Simple:Soft' (cf. their Table 5) with a generalization accuracy of 83% (also reported in our Table 2), whereas their Lex:Simple model lags 1 point behind. For the latter, the authors provide per generalization type output: link. Numbers in Table 3 are for Lex:Simple, not Lex:Simple:Soft.

For Zheng and Lapata (2021), our reported number was provided directly by the authors after publication of their paper.

[3]The words "gardener" and "monastery" occur zero times in the train set, but in total in 22 sentences of the generalization set. The majority (15) of these appear in PP recursion samples.

[4]`https://github.com/thousfeet/LEAR`



$\star$ `boy(`x_1`) ; want.agent(`x_2`,`x_1`)` $\wedge$ `want.xcomp(`x_2`,`x_4`)` $\wedge$ `go.agent(`x_4`,`x_1`)`

Figure 3: Logical form to graph conversion for "The boy wanted to go" (cf. (1)). For illustration only we use node names (the part before the '/') to outline the token alignment.

Lexical vs. structural generalization. As said above, structural generalization is underrepresented in COGS (3 out of 21 generalization types), and lexical generalization (the remaining 18 types) is therefore dominating the evaluation. As a consequence, an overall generalization accuracy above 80% can be achieved without even touching upon structural generalization. In Table 6 we report the average accuracy of both classes (by averaging over all types of the respective class), along with the overall generalization accuracy. Some models do not report standard deviations.

C Additional information on COGS to graph conversions

This is a more detailed explanation of the COGS logical form to graph conversion described in Section 3.2 based on four additional example sentences:

(1) The boy wanted to go.
$\star$`boy(`x_1`); want.agent(`x_2`,`x_1`)` $\wedge$
`want.xcomp(`x_2`,`x_4`)`
$\wedge$ `go.agent(`x_4`,`x_1`)`

(2) Ava was lended a cookie in a bottle.
`lend.recipient(`x_2`, Ava)`
$\wedge$ `lend.theme(`x_2`,`x_4`)`
$\wedge$ `cookie(`x_4`)`
$\wedge$ `cookie.nmod.in(`x_4`,`x_7`)`
$\wedge$ `bottle(`x_7`)`

(3) Ava said that Ben declared that Claire slept.
`say.agent(`x_1`, Ava)`
$\wedge$ `say.ccomp(`x_1`,`x_4`)`
$\wedge$ `declare.agent(`x_4`, Ben)`
$\wedge$ `declare.ccomp(`x_4`,`x_7`)`
$\wedge$ `sleep.agent(`x_7`, Claire)`

(4) touch
$\lambda a.\lambda b.\lambda e.$ `touch.agent(`e,b`)` $\wedge$
`touch.theme(`e,a`)`

The first of these is used as the main example for now. Its graph conversion can be found in Fig. 3.

Basic ideas. *Arguments* of predicates (variables like x_i or proper names like `Ava`) are translated

	train				train100			
Type	AM	AM+dist	AM+B	AM+B+dist	AM	AM+dist	AM+B	AM+B+dist
Subj to Obj (common noun)	65.8±43.4	88.3±10.9	99.7± 0.1	96.5± 6.8	99.9± 0.1	99.9± 0.1	100.0± 0.1	99.9± 0.2
Subj to Obj (proper noun)	69.9± 9.8	48.1±32.0	66.3±38.8	61.8±47.3	98.9± 1.7	100.0± 0.0	89.6± 8.1	95.8± 9.3
Obj to Subj (common noun)	53.1±45.0	97.9± 4.4	99.9± 0.2	88.0±26.7	99.9± 0.1	99.8± 0.2	100.0± 0.1	99.9± 0.1
Obj to Subj (proper noun)	90.0±21.4	88.3±25.9	88.9±11.2	78.8±42.9	99.8± 0.0	99.8± 0.1	99.9± 0.0	99.9± 0.0
Prim to Subj (common noun)	3.4± 7.6	0.0± 0.0	76.2±42.2	80.3±42.2	98.0± 4.5	59.9±54.7	100.0± 0.0	100.0± 0.0
Prim to Subj (proper noun)	4.7±10.6	1.0± 2.3	99.9± 0.1	100.0± 0.0	99.8± 0.3	99.9± 0.1	100.0± 0.0	100.0± 0.1
Prim to Obj (common noun)	0.2± 0.4	0.0± 0.0	74.5±32.5	80.1±40.7	95.9± 8.9	59.9±54.7	100.0± 0.0	100.0± 0.0
Prim to Obj (proper noun)	10.4± 9.1	22.0±15.6	90.5± 9.9	94.9± 3.7	98.8± 2.4	99.8± 0.4	84.9± 9.1	94.4± 9.0
Prim verb to Infin. arg	59.7±54.2	55.2±50.5	100.0± 0.0	82.9±38.2	17.6±30.8	1.0± 2.2	100.0± 0.0	100.0± 0.0
ObjmodPP to SubjmodPP	38.1±23.1	26.1±15.1	59.0±40.8	71.5±24.0	48.0±17.3	44.8±23.9	49.1±27.5	77.7± 7.1
CP recursion	100.0± 0.0	100.0± 0.1	100.0± 0.0	100.0± 0.0	99.9± 0.1	100.0± 0.0	100.0± 0.0	100.0± 0.0
PP recursion	60.5± 4.2	97.6± 0.9	36.3± 8.0	97.3± 2.0	57.2± 8.3	97.0± 1.1	41.5±11.2	98.6± 0.5
Active to Passive	69.3±42.2	41.7±52.3	83.0±24.8	78.8±31.3	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0
Passive to Active	51.6±45.2	46.6±50.2	45.5±27.2	52.0±43.6	99.6± 0.7	99.9± 0.1	100.0± 0.0	100.0± 0.0
ObjOTrans. to trans.	79.6±33.6	77.8±28.2	22.3±24.0	35.6±33.4	99.9± 0.1	100.0± 0.1	100.0± 0.0	100.0± 0.0
Unacc to transitive	33.2±36.1	51.2±47.2	48.2±35.8	48.9±41.5	99.6± 0.7	100.0± 0.1	100.0± 0.0	100.0± 0.0
Dobj dative to PP dative	99.3± 0.8	98.8± 2.0	99.8± 0.1	95.0±11.0	99.9± 0.1	99.9± 0.1	100.0± 0.0	100.0± 0.0
PP dative to Dobj dative	90.4±11.9	79.5±44.5	85.6±21.7	89.5±11.5	99.7± 0.1	99.8± 0.1	100.0± 0.0	100.0± 0.0
Agent NP to Unacc Subj	78.5±43.4	99.7± 0.6	95.3± 6.4	78.2±43.9	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0
Theme NP to ObjOTrans. Subj	99.9± 0.1	99.2± 1.7	99.9± 0.1	70.5±41.9	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0
Theme NP to Unergative Subj	100.0± 0.1	96.6± 7.6	99.9± 0.1	64.4±49.0	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0
Total	59.9±21.1	62.7±18.7	79.6±15.4	78.3±27.7	91.1± 3.6	88.6± 6.6	93.6± 2.7	98.4± 1.3

Table 4: Exact match accuracy on the generalization set by generalization type for all AM parser models.

Type	train LeAR
Subj to Obj (common noun)	99.8± 0.0
Subj to Obj (proper noun)	93.1±10.2
Obj to Subj (common noun)	100.0± 0.0
Obj to Subj (proper noun)	99.9± 0.0
Prim to Subj (common noun)	100.0± 0.0
Prim to Subj (proper noun)	100.0± 0.0
Prim to Obj (common noun)	99.8± 0.0
Prim to Obj (proper noun)	93.1±10.2
Prim verb to Infin. arg	100.0± 0.0
ObjmodPP to SubjmodPP	92.5± 9.4
CP recursion	100.0± 0.0
PP recursion	98.5± 0.0
Active to Passive	100.0± 0.0
Passive to Active	100.0± 0.0
ObjOTrans. to trans.	100.0± 0.0
Unacc to transitive	100.0± 0.0
Dobj dative to PP dative	99.9± 0.0
PP dative to Dobj dative	90.9± 0.0
Agent NP to Unacc Subj	100.0± 0.0
Theme NP to ObjOTrans. Subj	100.0± 0.0
Theme NP to Unergative Subj	100.0± 0.0
Total	98.9± 0.9

Table 5: Exact match accuracy on the generalization set by generalization type for the LeAR reproduction runs on train.

Model	trained on	Lexical	Structural	Overall
AM	train	58.8± 2.7	66.2± 8.2	59.9± 2.7
AM+dist	train	60.7±12.4	74.5± 5.2	62.7±10.8
AM+B	train	82.0± 7.3	65.1±11.6	79.6± 6.4
AM+B+dist	train	76.5±25.4	89.6± 8.7	78.3±22.9
AM	train100	94.9± 2.1	68.4± 6.7	91.1± 2.3
AM+dist	train100	90.0± 6.0	80.6± 8.2	88.6± 4.9
AM+B	train100	98.6± 0.9	63.5± 9.2	93.6± 1.4
AM+B+dist	train100	99.4± 1.0	92.1± 2.3	98.4± 0.9
LeAR	train	99.2± 1.1	97 ± 3.1	98.9± 0.9
Kim and Linzen 2020	train	41.2±	0 ±	35 ±
Akyürek and Andreas 2021	train	75.7± 1.1	0.5± 0.6	82.1± 0.6
Zheng and Lapata 2021	train	99.8±	16.8±	87.9±
Kim and Linzen 2020	train100	73 ±	0 ±	63 ±
Csordás et al. 2021	train100	88 ±	0 ±	75 ±

Table 6: Lexical vs structural generalization for seq2seq and compositional models

to nodes. The first part of each predicate name (e.g. `boy`, `want`, `go`) is the lemma of the token pointed to by the first argument (e.g. x_1, x_2, x_4), we strip this lemma ('delexicalize') from the predicate and insert it as the node label of the first argument (post-processing reverses this).

Binary predicates (i.e. terms with 2 arguments) are translated into edges, pointing from their first to their second argument, e.g. `want.agent`(x_2, x_1) is converted to an 'agent' edge from node x_2 (the 'want' node) to node x_1.

For *unary predicates* like `boy`(x_1) the delexicalization already suffices, so we don't add any edge (in lack of a proper target node). We restore unary predicates during postprocessing for nodes with no outgoing edges.

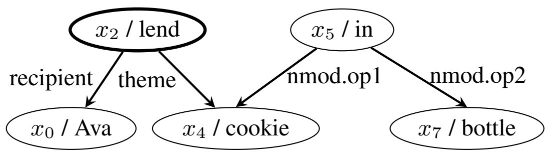
For a definite NP covering input token positions $i-1$ and i (i.e. "the$_{i-1}$ noun$_i$"), COGS includes a *iota term* `*noun`(x_i)`;` in the output. This definite NP meaning is treated as if it was a conjunction of the noun meaning (i.e. `noun`(x_i)) and 'definite determiner meaning' binary predicate `the.iota`(x_{i-1}, x_i).
The AM parser further requires one node to be the *root node*. For non-primitives we select it heuristically as the node with no incoming edges (excluding preposition and determiner nodes).

Prepositions. We 'reify' prepositions so each becomes a node of the graph with outgoing 'nmod' edges to the modified NP and the argument NP.

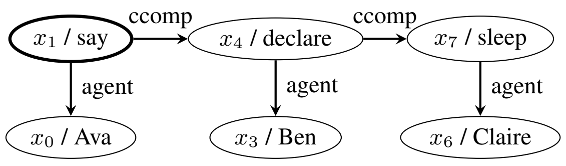
Alignments. For training the AM parser additionally needs *alignments* of the nodes to the input tokens. Luckily all x_i nodes naturally provide alignments (alignment to ith input token). For proper names we simply align them to the first occurrence in the sentence. The determiner node is aligned to the token preceding the corresponding x_{noun}. Edges are implicitly aligned by the blob heuristics, which are pretty simple here; every edge belongs to the blob of the node it originates from.

Primitives. For primitive examples (e.g. "touch" (4)) we mostly follow the same procedure. Unlike non-primitives, however, their resulting graph *can* have open sources beyond the root node, e.g. "touch" would have sources at the nodes b and a (incoming 'agent' or 'theme' edge respectively). These nodes can receive any source out of the three available (S0,S1,S2)[5], so the tree automaton build as part of Groschwitz et al. (2021)'s method would allow any combination of source names for the unfilled 'arguments'. Because there is only one input token, alignment is trivial. Primitives quite closely resemble the 'supertags' of the AM parser.

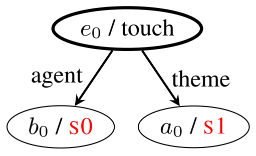
The graph conversion for (1) was already presented in Fig. 3. For the other three examples (2)–(4), we present the graph conversions in Fig. 4.



(a) See also (2).



(b) See also (3).



(c) See also (4).

Figure 4: Results of the logical form to graph conversion for (2)–(4). Actually for (c) the tree automaton contained all possible source name combinations for nodes a and b, not just ⟨s0,s1⟩.

[5]With the restriction that different nodes should have different sources to prevent the nodes from being merged. We don't consider non-empty type requests for these nodes here.