Compositional Generalization Requires Compositional Parsers

Anonymous ACL submission

Abstract

A rapidly growing body of research on *compositional generalization* investigates the ability of a semantic parser to dynamically recombine linguistic elements seen in training into unseen sequences. We present a systematic comparison of sequence-to-sequence models and models guided by compositional principles on the recent COGS corpus (Kim and Linzen, 2020). Though seq2seq models can perform well on lexical tasks, they perform with near-zero accuracy on *structural generalization* tasks that require novel syntactic structures; this holds true even when they are trained to predict syntax instead of semantics. In contrast, compositional models achieve near-perfect accuracy on structural generalization; we present new results confirming this from the AM parser (Groschwitz et al., 2021). Our findings show structural generalization is a key measure of compositional generalization and requires models that are aware of complex structure.

1 Introduction

Compositionality is a fundamental principle of natural language semantics: "The meaning of a whole [expression] is a function of the meanings of the parts and of the way they are syntactically combined" (Partee, 1984). A growing body of research focuses on *compositional generalization*, the ability of a semantic parser to combine known linguistic elements in novel structures in ways akin to humans. For example, observing the meanings of *"The hedgehog ate a cake"* and *"A baby liked the penguin,"* can a model predict the meaning of *"A baby liked the hedgehog"*? Dynamic, compositional recombination helps explain efficient human language learning and usage, and investigating whether NLP models make use of the same property offers important insight into their behavior.

Current research on compositional generalization shows the task to be challenging and complex. Such research centers around a number of corpora designed specifically for the task, including SCAN (Lake and Baroni, 2018) and CFQ (Keysers et al., 2020). We focus on COGS (Kim and Linzen, 2020), a synthetic semantic parsing corpus of English whose test set consists of 21 *generalization types* such as the example above (Section 2). Kim and Linzen report that simple sequence-to-sequence (seq2seq) models such as LSTMs and Transformers struggle with many of their generalization types, achieving an overall highest accuracy on the generalization set of 35%. Subsequent work has improved accuracy on the COGS generalization set considerably (Tay et al., 2021; Akyürek and Andreas, 2021; Conklin et al., 2021; Csordás et al., 2021; Orhan, 2021; Zheng and Lapata, 2021), but the accuracy of even the best seq2seq models remains below 88%. By contrast, Liu et al. (2021) report an accuracy of 98%, using an algebraic model that implements compositionality (Section 3).

Here, we investigate whether this difference in compositional generalization accuracy is incidental, or whether there is a systematic difference between seq2seq models and models that are guided by compositional principles and aware of complex structure. Comparisons between entire classes of models must be made with care. Thus in order to make claims about the class of compositional models, we first work out a second compositional model for COGS (in addition to Liu et al.'s). We apply the AM parser (Groschwitz et al., 2021), a compositional semantic parser which can parse a variety of graphbanks fast and accurately (Lindemann et al., 2020), to COGS after minimal adaptations (Section 4). The AM parser achieves a generalization accuracy above 98%, making it the first semantic parser shown to perform accurately on both COGS and broad-coverage semantic parsing.

We then compare these two compositional models to all published seq2seq models for COGS. We find that the difference in generalization accuracy

can be attributed specifically to *structural* types of compositional generalization, which require the parser to generalize to novel syntactic structures that were not observed in training. While the compositional parsers achieve excellent accuracy on these generalization types, all known seq2seq models perform very poorly, with accuracies close to zero. This is even true for BART (Lewis et al., 2020), which we apply to COGS for the first time; this is surprising because BART achieves very high accuracy on broad-coverage semantic parsing tasks (Bevilacqua et al., 2021). We conclude that seq2seq models, as a class, seem to have a weakness with regard to structural generalization that compositional models overcome (Section 5).

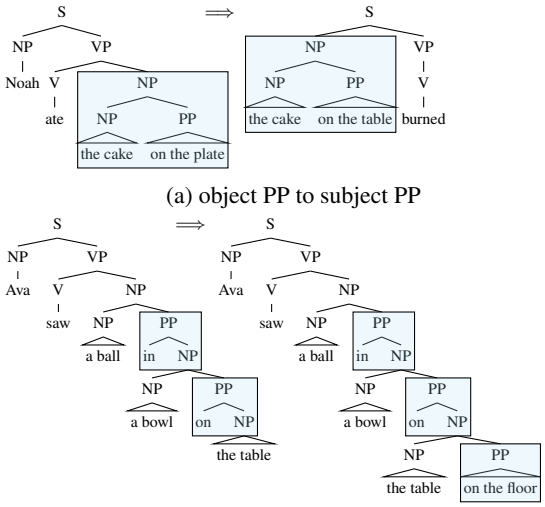
Finally, we investigate the role of syntax in compositional generalization (Section 6). We show that parsers which explicitly model syntactic tree structures can easily learn structural generalization when trained to predict syntax trees on COGS, whereas BART again performs poorly. BART does not learn structural generalization even if we enrich its input with syntactic information. Thus, the poor performance of seq2seq models on structural generalization is not specifically due to representational choices in COGS, or even to the specific compositional demands of semantic parsing; structural generalization requires structure-aware models.

We discuss implications for future work on compositional generalization in Section 7. All code will be made publicly available upon acceptance.

2 Compositional Generalization

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 tasks systematically vary language use between the training and the generalization set; as such, the system must recombine parts of multiple training instances to predict the meaning of a single test instance.

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. We follow the authors and distinguish two classes of generalization types; we further comment on a third class based on data from model performance.



(a) object PP to subject PP

(b) PP recursion

Figure 1: Structural generalization in COGS.

Lexical generalization involves recombining known grammatical structures with words that were not observed in these particular structures in training. An example is the generalization type "subject to object (common)" (Table 1a), in which a common noun ("hedgehog") is only seen as a subject in training, whereas it is only used as on object in the generalization testset. Note that the syntactic structure at generalization time (e.g. that of a transitive sentence) was already observed in training. On the semantics side, the meaning representations are identical, except for replacing some constants and quantifiers and renaming some variables. Thus, lexical generalization in COGS amounts to learning how to fill fixed templates.

By contrast, *structural generalization* involves generalizing to linguistic structures that were not seen in training (cf. Table 1c,d). Examples are the generalization types "PP recursion", where training instances contain prepositional phrases of depth up to two and generalization instances have PPs of depth 3–12; and "object PP to subject PP", where PPs modify only objects in training and only subjects at test time. These structural changes are illustrated in Fig. 1.

A third class we observe involves generalizing to *object usage of proper nouns* (Table 1b). Though technically a subset of lexical generalization, this subgroup is harder than types of the same class

	Training	Generalization
(a) LEX	A hedgehog ate the cake. *cake(x4); hedgehog(x1) ∧ eat.agent(x2, x1) ∧ eat.theme(x2, x4)	The baby liked the hedgehog. *baby(x1); *hedgehog(x4); like.agent(x2, x1) ∧ like.theme(x2, x4)
(b) PROP	Charlie ate the cake. *cake(x3); eat.agent(x1, Charlie) ∧ eat.theme(x1, x3)	The monster ate Charlie. *monster(x1); eat.agent(x2, x1) ∧ eat.theme(x2, Charlie)
(c) STRUCT	Ava saw a ball in a bowl on the table. *table(x9); see.agent(x1, Ava) ∧ see.theme(x1, x3) ∧ ball(x3) ∧ ball.nmod.in(x3, x6) ∧ bowl(x6) ∧ bowl.nmod.on(x6, x9)	Ava saw a ball in a bowl on the table on the floor. *table(x9); *floor(x12); see.agent(x1, Ava) ∧ see.theme(x1, x3) ∧ ball(x3) ∧ ball.nmod.in(x3, x6) ∧ bowl(x6) ∧ bowl.nmod.on(x6, x9) ∧ table.nmod.on(x9, x12)
(d) STRUCT	Noah ate the cake on the plate. *cake(x3); *plate(x6); eat.agent(x1, Noah) ∧ eat.theme(x1, x3) ∧ cake.nmod.on(x3, x6)	The cake on the table burned. *cake(x1); *table(x4); cake.nmod.on(x1, x4) ∧ burn.theme(x3, x1)

Table 1: Some examples from the COGS dataset. Examples (a) represent lexical generalization (LEX); (b), to object proper noun generalization (PROP); and (c-d), structural generalization (STRUCT).

(cf. Section 5); we report and discuss it separately.

Lexical generalization captures a very limited fragment of compositionality, in that it only requires to fill a fixed number of slots with new values. The key point about compositionality in semantics is that language is infinitely productive, and humans can assign meaning to new grammatical structures based on finite experience. Assigning meaning to unseen structures is exercised only by structural types. This distinction is borne out in model performance (Section 5): while lexical generalization can be handled by many neural architectures, structural generalization requires parsing architectures aware of complex sentence structure.

3 Related Work

Kim and Linzen (2020) demonstrate that simple seq2seq models (LSTMs and Transformers) struggle with all generalization types in COGS. Subsequent work with novel seq2seq architectures achieve a much higher mean accuracy on the COGS generalization set (Akyürek and Andreas, 2021; Csordás et al., 2021; Conklin et al., 2021; Tay et al., 2021; Orhan, 2021; Zheng and Lapata, 2021), but their accuracy on the generalization set still lags more than ten points behind that on the in-domain test set.

COGS can also be addressed with *compositional models*, which directly model linguistic structure and implement the Principle of Compositionality. The LeAR model of Liu et al. (2021) achieves a generalization accuracy of 98%, outperforming all known seq2seq models by at least ten points. LeAR also sets new states of the art on CFQ and Geoquery, but has not been demonstrated to be applicable to broad-coverage semantic parsing.

Compositional semantic parsers for other tasks include the AM parser (Groschwitz et al., 2018; Lindemann et al., 2020) (Section 4) and SpanBasedSP (Herzig and Berant, 2021). The AM parser has been shown to achieve high accuracy and parsing speed on broad-coverage semantic parsing datasets such as the AMRBank. SpanBasedSP parses Geoquery, SCAN, and CLOSURE accurately through unsupervised training of a span-based chart parser. Shaw et al. (2021) combine quasi-synchronous context-free grammars with the T5 language model to obtain even higher accuracies on Geoquery, demonstrating some generalization from easy training examples to hard test instances.

Structural generalization has also been probed in syntactic parsing tasks. Linzen et al. (2016) define a number-prediction task that requires learning syntactic structure and find that LSTMs perform with some success; however, Kuncoro et al. (2018) find that structure-aware RNNGs perform this task more accurately. McCoy et al. (2020) found that hierarchical representations are necessary for human-like syntactic generalizations on a question formation task, which seq2seq models cannot learn.

4 Parsing COGS with the AM parser

4.1 The AM parser

To better understand how compositional models perform on compositional generalization, we adapt

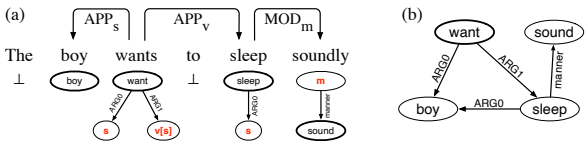


Figure 2: (a) AM dependency tree with (b) its value.

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). Thus, though not yet tested on synthetic generalization sets, the AM parser exhibits the ability to handle natural language and related generalizations in the wild.

Instead of predicting the graph directly, the AM parser first predicts a graph fragment for each token in the sentence and a (semantic) dependency tree that connects them. This is illustrated in Fig. 2a; words that do not contribute to the sentence meaning are tagged with $\perp$. This dependency tree is then evaluated deterministically into a graph (Fig. 2b) using the operations of the *AM algebra*. The "Apply" 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, this is how the APP_s operation inserts the "boy" node into the ARG0 of "want". The "Modify" operation attaches a modifier to a node; this is how the MOD_m operation 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 representations for the two tokens to an MLP. We follow the setup of Groschwitz et al. (2021), which does not require explicit annotations with AM dependency trees, to train the parser.

4.2 AM parsing for COGS

We apply the AM parser to COGS by converting the semantic representations in COGS to graphs.

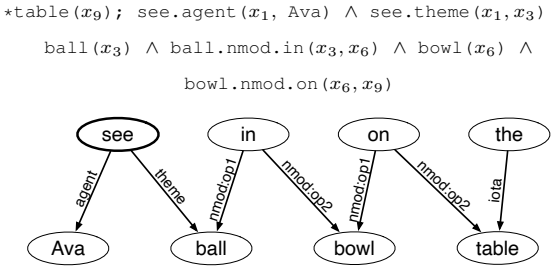


Figure 3: Logical form to graph conversion for "Ava saw a ball in a bowl on the table" (cf. Table 1c).

The conversion is illustrated in Fig. 3.

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 $\texttt{pred.arg}(x, y)$, we assign it the node label `pred` in the graph. We also add an edge from x to y with label `arg`. E.g. $\texttt{see.agent}(x_1, \texttt{Ava})$ turns into an 'agent' edge from 'see' to 'Ava'. Each *iota term* $\star\texttt{noun}(x_{\text{noun}})$ is treated as an edge from a fresh node with label "the" to x_{noun}. Preposition meaning $\texttt{bowl.nmod.on}(x_6, x_9)$ 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 E.

5 Experiments on COGS

With two compositional models available on COGS, we can now compare compositional semantic parsers, as a class, to seq2seq models, as a class, on compositional generalization in COGS.

5.1 Experimental setup

We follow standard COGS practice and evaluate all models on both the (in-distribution) test set and the generalization set. 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). The 'train100' set extends 'train' with 100 copies of each exposure example. For instance, for the generalization instance in Table 1a, 'train100' will contain 100 different sentences in which "the/a hedgehog" appears as subject (rather than just one in 'train'). We report exact match accuracies, aver-

		train		train100	
		Test	Gen	Test	Gen
seq-to-seq	Kim and Linzen 2020	96	35	94	63
	Conklin et al. 2021	99	66.7	99	75.4
	Csordás et al. 2021	100	81	-	75.4
	Akyürek and Andreas 2021	-	83	99	84.5
	Zheng and Lapata 2021 †	-	87.9	-	-
	Orhan 2021 †	-	84.6	-	-
	Tay et al. 2021 †	95	77.5	-	-
	BART †	100	77.5±0.4	100	82.7±1.4
	BART+syn †	100	80.2±0.4	100	85.9±0.3
compositional	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: Exact match accuracies on COGS. Results in gray are taken from the respective papers. †) models that use pretraining.

aged across 5 training runs, along with their standard deviations.

Sequence-to-sequence models. We train BART (Lewis et al., 2020) as a semantic parser on COGS. This is a strong representative of the family of seq2seq models, as a slightly extended form of BART (Bevilacqua et al., 2021) set a new state of the art on semantic parsing on the AMR corpus (Banarescu et al., 2013). To apply BART on COGS, we directly fine-tune the pretrained *bart-base* model on it with the corresponding tokenizer. Training details are described in Appendix C.

We also report results for all other published seq2seq models for COGS (Kim and Linzen, 2020; Conklin et al., 2021; Csordás et al., 2021; Akyürek and Andreas, 2021; Tay et al., 2021; Orhan, 2021; Zheng and Lapata, 2021). We retrained some of these models on train100 to measure the impact of the training set.

Compositional models. We train the AM parser on the COGS graph corpus (cf. Section 4.2) and copied most hyperparameter values from Groschwitz et al. (2021)'s training setup for AMR to make overfitting to COGS less likely; details are described in Appendix B.

The AM parser either receives pretrained word embeddings from BERT (Devlin et al., 2019) ('AM+B') or learns embeddings from the COGS data only ('AM'). We run the training algorithm with up to three argument slots to enable the analysis of ditransitive verbs. For evaluation, we revert the graph conversion to reconstruct the logical forms.

For PP recursion, COGS eliminates potential PP attachment ambiguities and assumes that each PP modifies the noun immediately to its left. We hypothesize that explicit distance information between tokens could help the AM parser learn this regularity: 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.

Finally, we report evaluation results for LeAR, the compositional COGS parser of Liu et al. (2021).

5.2 Results

The results are summarized in Table 2.

Compositional outperforms seq2seq. While all models achieve near-perfect accuracy on the in-distribution test sets, we find that when trained on 'train100', all compositional models outperform all seq2seq models on the generalization set, by a wide margin. This includes the very strong BART baseline, which holds the state of the art in broad-coverage parsing for AMR.

LeAR even achieves its near-perfect accuracy when trained on 'train', and outperforms all seq2seq models trained on either dataset. See below for a detailed discussion of the AM parser.

Performance by generalization type. To understand this result more clearly, we break down the accuracy by generalization type. This analysis is shown in Table 3. We will explain "BART+syn" in Section 6.2 and the "syntax" rows in Section 6.1. We compare the compositional models against all seq2seq models that report these fine-grained numbers or for which they were easy to reproduce (see Appendices C and D for details).

The results group neatly with the three classes of generalization types outlined in Section 2: LEX, STRUCT, and PROP. All recent models achieve near-perfect accuracy on each of the 16 lexical generalization types. On structural generalization types, seq2seq models achieve very low accuracies, whereas the compositional parsers (AM+B+dist and LeAR) are still very accurate. The proper-noun object cases are somewhere in the middle, with the seq2seq models reporting middling numbers.

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

		Class	STRUCT			PROP		LEX	
		Gen. type	Obj to Subj PP	CP recursion	PP recursion	prim to obj (proper)	subj to obj (proper)	all 16 other types	Overall
	AM+B	train100	49	100	41	85	90	100	94
	AM+B+dist	train100	78	100	99	94	96	100	98
	LeAR	train	93	100	99	93	93	100	99
semantics	Kim and Linzen 2020	train	0	0	0	0	30	45	35
semantics	Akyürek and Andreas 2021	train	0	0	1	66	64	100	82
semantics	Orhan 2021	train	0	0	10	84	86	100	85
semantics	Zheng and Lapata 2021	train	0	12	39	92	91	100	89
semantics	Kim and Linzen 2020	train100	0	0	0	23	54	77	63
semantics	Conklin et al. 2021	train100	0	0	0	20	66	94	75
semantics	Csordás et al. 2021	train100	0	0	0	55	62	92	75
semantics	BART	train100	0	0	10	55	86	99	83
semantics	BART+syn	train100	0	7	8	98	95	100	86
syntax	Benepar	train100	84	95	98	99	100	100	99
syntax	BART	train100	1	4	8	97	94	96	83

Table 3: Exact match accuracies on the individual generalization types. We have compressed all 16 generalization types of the LEX class into a single column and report the average accuracy.

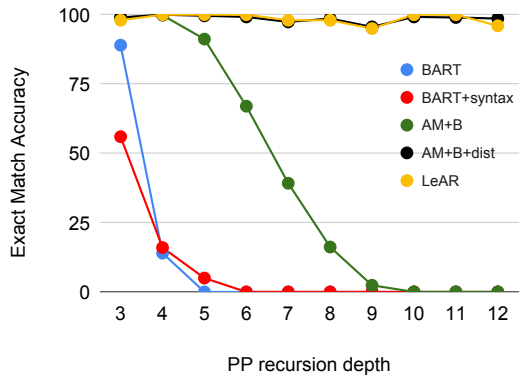


Figure 4: Influence of PP recursion depth on overall PP depth generalization accuracy.

Depth generalization (recursion). There is a particularly pronounced difference between compositional and seq2seq models on the two "recursion" generalization types (cf. Fig. 1b). In these cases, the training data contains examples up to depth two and the generalization data has depths 3–12. Figure 4 shows the accuracy of several models on PP recursion in detail. As we see, the accuracy of BART (even when informed by syntax, cf. Section 6.2) degrades quickly with recursion depth. By contrast, both LeAR and AM+B+dist maintain their high accuracy across all recursion depths. This suggests that they learn the correct structural generalizations even from training observations of limited depth.

Effect of distance encoding for AM parser. As illustrated in Fig. 4, the accuracy of the unmodified AM parser without the distance feature degrades with increasing PP recursion depth. An error analysis showed that this is because the AM parser is uncertain about the attachment of PPs in the middle of the string, confirming our hypothesis that it does not learn the idiosyncratic treatment of PPs in COGS (always attach low). Adding the distance feature solves this problem.

There is an interesting asymmetry between the behavior of the AM parser on PP recursion and CP recursion, which nests sentential complements within each other ("Emma said that Noah knew that the cat danced"): The accuracy of the unmodified AM parser 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, not CPs.

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 (e.g. "Emma" instead of "Charlie" in Table 1b). Trained on 'train', AM+B+dist achieves a mean accuracy on STRUCT of 89.6 (compared to 92.3 for 'train100'), whereas the mean accuracy on LEX drops to 76. Even without BERT and trained on 'train', AM+dist gets 74.6 on STRUCT, drastically outperforming the seq2seq models (Appendix D).

6 The role of syntax

Our finding that seq2seq models perform so poorly on structural generalization in COGS begs the question: Is there anything special about the meaning representations in COGS that makes structural generalization hard, or would seq2seq models struggle similarly on other target representations for these generalization types? Do seq2seq models have a specific weakness regarding semantic compositionality? Or is it because they systematically lack a bias that would help them generalize over structure in language? In this section, we investigate these questions by recasting COGS as a syntactic corpus.

6.1 Syntactic generalization

We obtain a syntactic annotation for each instance in COGS from the (unambiguous) original PCFG grammar used to generate COGS (cf. Fig. 1). We replace the very fine-grained non-terminals (e.g. `NP_animate_dobj_noPP`) of the original PCFG with more general ones (e.g. `NP`) and remove duplicate rules (e.g. `NP`→`NP`) resulting from this. We train BART on predicting linearized constituency trees from the input strings. For comparison, we also train the Neural Berkeley Parser (Kitaev and Klein, 2018) on COGS syntax ("Benepar" in the tables). This parser consists of a self-attention encoder and a chart decoder. It is therefore *structure-aware*, in that it explicitly models tree structures; this is the analogue of a compositional parser for semantics.

Results are shown in the two bottom rows of Table 3. We find the same pattern as in the semantic parsing case: the seq2seq model does well on PROP and LEX, but struggles with STRUCT. The structure-aware Berkeley parser handles all three generalization types well. Thus, the difficulties that seq2seq models have on structural generalization on COGS are not limited to semantics: rather, they seem to be a general limitation in the ability of seq2seq models to learn linguistic structure from structurally simple examples and use it productively. Not only does compositional generalization require compositional parsers; structural generalization in semantics or syntax seems to require parsers which are aware of that structure.

6.2 Compositional generalization from correct syntax

But perhaps the poor performance of seq2seq semantic parsers on STRUCT is caused *only* by their inability to learn to generalize syntactically? Would their accuracy catch up with that of compositional models if we gave them access to syntax?

We retrained BART on predicting semantic representations, but instead of feeding it the raw sentence, we provide as input the linearized gold constituency tree ("`(NP (Det a) (N rose))`"), both for training and inference. This method is similar to Li et al. (2017) and Currey and Heafield (2019), but we allow attention over special tokens such as "`(`" during decoding.

We report the results as "BART+syn" in Table 2 and Table 3; the overall accuracy increases by 3.2% over BART. This is mostly because providing the syntax tree allows BART to generalize correctly on PROP. However, STRUCT remains out of reach for BART+syn, confirming the deep difficulty of structural generalization for seq2seq models.

We also explored other ways to inform BART with syntax, through multi-task learning (Sennrich et al., 2016; Currey and Heafield, 2019) and syntax-based masking in the self-attention encoder (Kim et al., 2021). Neither method substantially improved the accuracy of BART on the COGS generalization set (+1.4% and +2.1% overall accuracy, respectively). More detailed results are in Appendix D.

7 Discussion

Compositional generalization requires compositional parsers. Table 3 paints a clear picture: compositional generalization in COGS can be solved by semantic parsers that have compositionality built in, but seq2seq models perform poorly on structural generalization. This remains true even for seq2seq models that are known to perform well on semantic parsing, for syntactic rather than semantic generalization, and for seq2seq models that are biased towards learning structure-aware representations by incorporating information about syntax. Obviously, statements about entire classes of models must be made with care. But when despite

the best efforts of an active research community *all* seq2seq models underperform the compositional models, that seems like rather strong evidence.

Our results are surprising, in that seq2seq models have been shown through probing tasks to learn some linguistic structure, both with respect to syntax (Blevins et al., 2018) and semantics (Tenney et al., 2019). At the same time, as mentioned above, seq2seq models like BART perform very well on broad-coverage tasks such as AMR parsing. It is an interesting question for future research to reconcile the ability of seq2seq models to learn soft structural information with their apparent difficulties in exploiting this ability to generalize structurally; perhaps their ability to learn structure rests on the variety of structures observed in broad-coverage training sets, but not in COGS.

Focus on structural generalization. Our experiments indicate that STRUCT is consistently harder than PROP and LEX with respect to generalization accuracy. Not only is LEX essentially a solved problem; but as we discussed in Section 2, the infinitely productive nature of full compositionality is only captured by structural types of generalization. Compositionality is not just about using new and similar words in known structures (slot filling), but also about building new, acceptable structures based on known ones.

When papers only report the mean accuracy of a system across all generalization types, the accuracy on the 16 lexical generalization types overshadows the accuracy on the three structural generalization types. The overall accuracy can make systems look more capable of compositional generalization than they really are.

Future work on compositional generalization will benefit from (i) reporting the accuracy on structural generalization tasks separately and (ii) expanding datasets that test compositional generalization to include more types of structural generalization. We hope to offer such a dataset in future research.

What's so difficult about objPP to subjPP? "ObjPP to subjPP" is the most challenging generalization type across all models. It is illuminating to investigate the errors that happen here, as they differ across models.

Table 4 shows typical errors of BART and the AM parser. The AM parser chooses to use the most recent simple NP ("the house") as the agent of

Gold	$*\text{baby}(x_1)$; $*\text{house}(x_7)$; $\text{baby.nmod.on}(x_1, x_4)$ $\wedge$ $\text{tray}(x_4)$ $\wedge$ $\text{tray.nmod.in}(x_4, x_7)$ $\wedge$ $\text{scream.agent}(x_8, x_1)$
BART	$*\text{baby}(x_1)$; $*\text{house}(x_7)$; $\text{scream.agent}(x_2, x_1)$ $\wedge$ $\text{scream.theme}(x_2, x_4)$ $\wedge$ $\text{tray}(x_4)$ $\wedge$ $\text{tray.nmod.in}(x_4, x_7)$
AM	$*\text{baby}(x_1)$; $*\text{house}(x_7)$; $\text{baby.nmod.on}(x_1, x_4)$ $\wedge$ $\text{tray}(x_4)$ $\wedge$ $\text{tray.nmod.in}(x_4, x_8)$ $\wedge$ $\text{scream.agent}(x_8, x_7)$

Table 4: Error analysis for the sentence "The baby on a tray in the house screamed".

"scream" and then attaches "the baby on a tray" in some random place. By contrast, BART analyzes the sentence as "the baby screamed the tray on the house", preferring to reuse the pattern for object-PP sentences even if the intransitive verb does not license it. BART also displays an unawareness of word order that is reminiscent of the difficulties that seq2seq models otherwise face in relating syntax to word order (McCoy et al., 2020).

We see from both examples that "objPP to subjPP" involves major structural changes to the formula that must be grounded in both lexical (verb valency) and structural (word order) information. Developing a model that learns to do this with perfect accuracy remains an interesting challenge.

8 Conclusion

We have shown that compositional semantic parsers systematically outperform recent seq2seq models on structural generalization in COGS. While both BART and the AM parser support accurate broad-coverage semantic parsing, we find that BART struggles with structural compositional generalization as much as other seq2seq models, whereas the compositional AM parser achieves state-of-the-art generalization accuracy on COGS.

These results suggests that even powerful seq2seq models lack a structural bias that is required to generalize across linguistic structures as humans do. This lack of bias is not limited to semantics; our findings indicate that seq2seq models struggle just as hard to learn syntactic generalizations that are easy for structure-aware models. Given that all recent models are accurate on most generalization types, we suggest focusing future evaluations on a model's accuracy on structural generalization types, and perhaps extend COGS to a corpus that offers a greater variety of these.

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.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186. 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.

Terra Blevins, Omer Levy, and Luke Zettlemoyer. 2018. Deep RNNs encode soft hierarchical syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 14–19, Melbourne, Australia. Association for Computational Linguistics.

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.

Anna Currey and Kenneth Heafield. 2019. Incorporating source syntax into transformer-based neural machine translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 24–33, Florence, Italy. 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.

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.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. 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 908–921, Online. 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)*.

Juyong Kim, Pradeep Ravikumar, Joshua Ainslie, and Santiago Ontañón. 2021. Improving compositional generalization in classification tasks via structure annotations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 637–645, Online. Association for Computational Linguistics.

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.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *Computing Research Repository (CoRR)*, arXiv:1412.6980. Published as a conference paper at ICLR 2015.

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.

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

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436, Melbourne, Australia. Association for Computational Linguistics.

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.

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.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697, Vancouver, Canada. Association for Computational Linguistics.

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.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

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.

R. Thomas McCoy, Robert Frank, and Tal Linzen. 2020. Does syntax need to grow on trees? sources of hierarchical inductive bias in sequence-to-sequence networks. *Trans. Assoc. Comput. Linguistics*, 8:125–140.

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.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Controlling politeness in neural machine translation via side constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 35–40, San Diego, California. Association for Computational Linguistics.

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.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. 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.

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

A COGS dataset statistics

The COGS dataset contains English declarative sentences mapped with logical forms. It was created by Kim and Linzen (2020) and is publicly available at `https://github.com/najoungkim/COGS` (MIT license). We use the version from April 2nd 2021 commit `6f66383` and use the dataset as-is (no datapoints excluded or changed, use their data set splits), except for the AM parser for which we conduct the logical form to graph preprocessing described in Section 4.2.
The normal training set ('train') consists of 24,155 samples (24k in distribution, 143 primitives, 12 exposure examples), the dev and test set both contain 3k in distribution samples each. Primitives and exposure examples contain 'lexical trigger words' necessary for all but the three structural generalization types: these lexical trigger words each appear only once and in one sample in the whole training set. Primitives are one-word sentences, therefore presenting word-meaning mapping without context of a sentence (necessary for the types Primitive to *). In contrast, exposure examples are full sentences e.g. for the subject to object (common noun) generalization this sentence contains "hedgehog" as the subject. In the generalization set this word appears in 1k samples, but in a different syntactic configuration compared to the exposure example (e.g. "hedgehog" in object position). There is also an additional larger training set ('train100') with 39,500 samples containing the lexical trigger words in 100 samples each, instead of just in one sample. The out-of-distribution generalization set contains 21k samples, 1k per generalization type.

B Training details of the AM parser

The corresponding code will be made publicly available upon acceptance.

Hyperparameters. For the AM parser, we mostly copied the hyperparameter values from the AMR experiments of Groschwitz et al. (2021). This should help against 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 in-doman 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 the trigger words occur 100 times in there). Furthermore, the COGS dataset doesn't have part-of-speech, lemma or named-entity annotations, so we just don't use embeddings for these. For the word embeddings we either use BERT-Large-uncased (Devlin et al., 2019) or learn embeddings from the dataset only (embedding dimension 1024, same as for the BERT model). We also decreased 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).
Unlike them we didn't use the fixed-tree decoder (described in Groschwitz et al. 2018), but opted for 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 also use supervised loss for edge existence and lexical labels: we can use supervised loss for both as they do not depend on the source names to be learnt. In preliminary experiments this yielded better results than using the automaton-based loss for them too. The supervised loss wasn't described in Groschwitz et al. (2021), but already implemented in their code base and they note there that the effect on performance was mixed in their experiments (similar for

SDP, worse for AMR).

Relative distance encoding. For the relative distance encodings we added to the dependency edge existence scoring, we used 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. In other words, for each token pair $\langle i, j \rangle$ the MLP has to decide edge existence based on the representations of the tokens at positions i and j, and an encoding of the relative distance $dist(i, j) = i - j$. 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 11.5million (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. As usual for compositional generalization datasets, it is relatively easy to get (near) perfect results on the (in domain) dev/test sets. We observed 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. Unfortunately, Kim and Linzen (2020) didn't provide a separate evaluation script. As a main evaluation metric they use (string) exact match accuracy on the logical forms which we adopt. Note that this 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.

C Training details of Seq2seq

Hyperparameters. We use the same hyperparameter setting for BART on both syntactic and semantic experiments. We use *bart-base*[2] model in all our experiments. Our batch size is 64. We use Adam optimizer (Kingma and Ba, 2015) with learning rate 1e-4 and gradient accumulation steps 8. Loss averaged over tokens is used as the validation metric for early stopping following Kim and Linzen (2020). During inference, we use beam search with beam size 4.

Dev set performance. The exact match accuracy is at least 99.6 for both dev set and (in-distribution) test set in all experiments.

Other details. Training took 4 hours for BART with about 80 epochs on train and 5 hours with about 50 epochs on train100. Inference on generalization set took about 1 hour. All BART experiments were run on Tesla V100 GPU cards (32GB). The number of parameters in our BART model is 140 million.

Syntactic annotations. To obtain syntactic annotations, we use NLTK[3] to parse each sentence in COGS with PCFG grammar generating COGS. In our experiments, we found this parsing process did not yield any ambiguous tree. The original PCFG grammar contains rules such as `NP→NP_animate_dobj_noPP`. We replace such fine-grained nonterminals (e.g. `NP_animate_dobj_noPP`) with general nonterminals (e.g. `NP`). This results in duplicate patterns (e.g. `NP→NP`) and we further remove such patterns from the output tree.

[2] https://huggingface.co/facebook/bart-base

[3] https://www.nltk.org/

Results from other papers. Conklin et al. (2021)[4], Akyürek and Andreas (2021)[5], Csordás et al. (2021)[6] and Tay et al. (2021) did not report performance of their model on train100 set. To report these numbers, we additionally use their published code to train their model on train100 for 5 runs. We use seed 6-10 for Conklin et al. (2021) and random number seeds for Csordás et al. (2021), following their default setting. We use their default configuration file for their best model to set the hyperparameters. Tay et al. (2021), did not publish their code so we did not report that. Orhan (2021)[7] and Zheng and Lapata (2021) are the two most recently published seq2seq approaches. Both did not provide numbers for train100 training and because of their recency we weren't able to run their models on the train100 set so far. We thus only report their published results for train set.

D Detailed evaluation results

The main results are summarized in the main paper in Section 5.2 with Table 2 and Table 3. Here we present AM parser (Table 5), LeAR (Table 6) and BART (Table 7) performance for each of COGS' 21 generalization types separately with the usual mean and standard deviation of 5 runs. For descriptions of the generalization types we refer to Kim and Linzen (2020, §3 and Fig. 1).

On accuracy computation for LeAR. We observed that the LeAR model skips 22 sentences in the generalization set due to out-of-vocabulary tokens.[8] We do 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. Furthermore we would like to note that–based on inspecting the published code[9]–, LeAR made the preprocessing choice to ignore the contribution of the definite determiner, basically treating indefinite and definite NPs equally, resulting in a big conjunction without any iota ('$\star$') prefixes.

On 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 actually '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, but not for the former, the authors provide per generalization type output in their accompanying GitHub repository as part of a jupyter notebook. Therefore numbers in Table 3 are for Lex:Simple, not Lex:Simple:Soft.
We picked the best performing model of Orhan (2021): According to their Table 2 the `t5-3b mt5_xl` model shows the best generalization performance (84.6% average accuracy). From the accompanying GitHub repository[10] we copy the model's results, specifically we average over the 5 runs of the model 3b-cogs-mt5-epochs10 (commit `04a2508`). We note that other models reported in Orhan (2021) showed the same performance pattern with respect to our three generalization classes LEX, PROP, and STRUCT.
For Zheng and Lapata (2021), our reported number is slightly different from the original paper. This is because we asked the authors for detailed results and they provide us with their newest results averaged over 5 runs.

Abbreviations in the tables. 'Subj' means 'subject', 'Obj' means 'object', 'Prim' means 'primitive', 'Infin. arg' means 'infinitival argument', 'ObjmodPP to SubjmodPP' means 'object-modifying PP to subject-modifying PP', 'ObjOTrans.' means 'object omitted transitive', 'trans.' means 'transitive', 'unacc' means 'unaccusative', 'Dobj' means 'Double Object'.

[4] https://github.com/berlino/tensor2struct-public
[5] https://github.com/ekinakyurek/lexical
[6] https://github.com/robertcsordas/transformer_generalization
[7] https://github.com/eminorhan/parsing-transformers
[8] The words "gardner" 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.
[9] https://github.com/thousfeet/LEAR
[10] https://github.com/eminorhan/parsing-transformers

	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 5: 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 6: Exact match accuracy on the generalization set by generalization type for the LeAR reproduction runs on train.

E Additional information on COGS to graph conversions

This is a more detailed explanation of the COGS logical form to graph conversion described in Section 4.2 based on four additional example sentences:

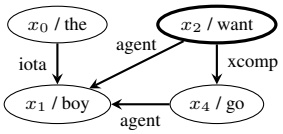
(1) The boy wanted to go.
$\star$boy(x_1); want.agent(x_2, x_1) $\wedge$ want.xcomp(x_2, x_4)
$\wedge$ go.agent(x_4, x_1)

(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. 5.



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

Figure 5: 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.

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

	train		train100			
Type	BART	BART+syn	BART	BART+syn	BART+mtl	BART+mask
Subj to Obj (common noun)	98.6±0.8	99.6± 0.2	99.2± 0.2	99.8± 0.1	99.6± 0.1	92.9± 1.2
Subj to Obj (proper noun)	68.7±1.1	87.0± 3.2	85.7± 6.7	94.7± 5.3	80.1± 5.5	93.3± 3.4
Obj to Subj (common noun)	99.2±0.6	99.8± 0.1	99.1± 1.3	99.7± 0.1	99.6± 0.2	98.7± 0.4
Obj to Subj (proper noun)	99.4±0.4	99.8± 0.0	99.5± 0.2	99.8± 0.1	97.8± 1.5	99.3± 0.3
Prim to Subj (common noun)	98.4±1.3	99.9± 0.0	95.0± 9.0	99.9± 0.0	99.7± 0.0	99.6± 0.2
Prim to Subj (proper noun)	98.6±0.9	100.0± 0.1	95.5± 4.3	100.0± 0.0	99.9± 0.1	98.9± 1.1
Prim to Obj (common noun)	98.9±0.6	99.5± 0.2	99.4± 0.2	99.8± 0.0	99.6± 0.1	96.1± 0.9
Prim to Obj (proper noun)	65.2±4.4	88.6± 4.3	55.2±27.1	98.1± 2.1	94.6± 0.3	94.8± 2.0
Prim verb to Infin. arg	99.9±0.1	100.0± 0.0	100.0± 0.0	100.0± 0.0	100.0± 0.0	99.9± 0.0
ObjmodPP to SubjmodPP	0.0±0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0	0.0± 0.0
CP recursion	0.3±0.3	5.9± 1.2	0.2± 0.4	6.5± 0.5	0.2± 0.2	1.1± 0.5
PP recursion	11.2±1.7	6.7± 0.2	10.2± 1.8	7.5± 0.4	11.7± 0.3	10.6± 1.4
Active to Passive	99.9±0.0	99.9± 0.0	99.9± 0.0	99.9± 0.0	100.0± 0.0	99.9± 0.0
Passive to Active	99.5±0.2	99.9± 0.0	99.9± 0.0	99.9± 0.0	99.9± 0.1	99.8± 0.2
ObjOTrans. to trans.	99.6±0.3	100.0± 0.0	99.9± 0.1	100.0± 0.0	99.9± 0.1	99.9± 0.2
Unacc to transitive	0.0±0.0	0.0± 0.0	99.9± 0.0	100.0± 0.0	99.9± 0.1	99.7± 0.2
Dobj dative to PP dative	98.3±1.2	99.4± 0.3	99.2± 0.2	99.5± 0.2	99.3± 0.0	99.1± 0.1
PP dative to Dobj dative	98.6±1.6	99.8± 0.0	99.5± 0.1	99.9± 0.1	99.6± 0.2	99.2± 0.3
Agent NP to Unacc Subj	96.2±1.4	99.1± 1.0	99.8± 0.2	99.6± 0.3	100.0± 0.0	96.2± 0.9
Theme NP to ObjOTrans. Subj	98.8±0.8	99.8± 0.3	99.6± 0.2	99.9± 0.0	100.0± 0.0	92.5± 5.5
Theme NP to Unergative Subj	99.1±0.7	99.8± 0.3	99.8± 0.2	99.8± 0.1	100.0± 0.0	94.1± 4.1
Total	77.5±0.4	80.2± 0.4	82.7± 1.3	85.9± 0.3	84.8± 0.2	84.1± 0.4

Table 7: Exact match accuracy on the generalization set by generalization type for all BART models.

strip this lemma ('delexialize') 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. $\texttt{want.agent}(x_2, x_1)$ is converted to an 'agent' edge from node x_2 (the 'want' node) to node x_1. Because of the delexicalization described above, there are only 8 different edge labels: 'agent', 'theme', 'recipient', 'xcomp', 'ccomp', 'iota' and 2 preposition-introduced edges described below.

For *unary predicates* like $\texttt{boy}(x_1)$ 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.

Each *iota term* $\star\texttt{noun}(x_{\texttt{noun}})\texttt{;}$ is treated as if it was a conjunction of the noun meaning (i.e. $\texttt{noun}(x_{\texttt{noun}})$) and 'definite determiner meaning' binary predicate $\texttt{the.iota}(x_{\texttt{the}}, x_{\texttt{noun}})$.

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. Instead of being treated as an edge as the above would suggest, we 'reify' them, so each preposition becomes a node of the graph with outgoing 'nmod' edges to the modified NP and the argument NP. So for "cookie in the bottle" (cf. (2) and Fig. 6a) we create a node with label 'in' and draw an outgoing 'nmod.op1' edge to the 'cookie'-node and an 'nmod.op2' edge to the 'bottle'-node.

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[11], the special determiner node is aligned to the token preceding the corresponding x_{noun}.[12] The 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)[13], 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, the alignment is trivial. In fact,

[11] this works because it seems that a name never appears more than once within a sentence. Names in the logical forms also seem to be ordered based on their token position.

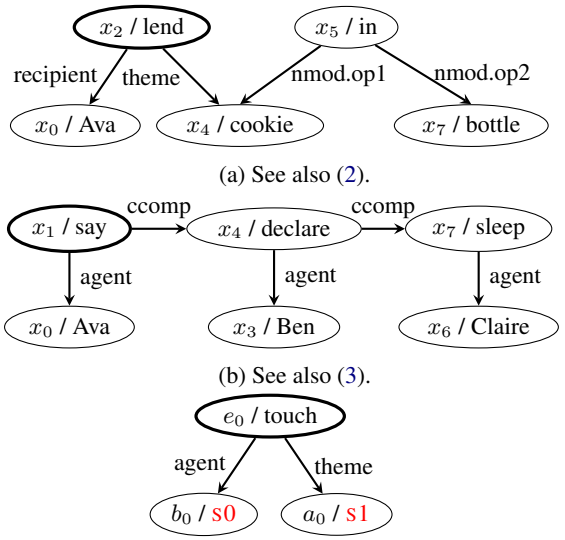
[12] we can do so because there are –beyond "the" and "a"– no pre-nominal modifiers like adjectives in this dataset.

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

primitives quite closely resemble the ‘supertags’ of the AM parser.

Note that by encoding the logical form as a graph we get rid of the ordering of the conjuncts. The ‘correct’ order (crucial for exact match evaluation) is restored during postprocessing.

The graph conversion for (1) was already presented in Fig. 5. For the other three examples (2)–(4), we present the graph conversions in Fig. 6.



(a) See also (2).

(b) See also (3).

(c) See also (4).

Figure 6: 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 $\langle$s0,s1$\rangle$.