Span-based Semantic Parsing for Compositional Generalization

Jonathan Herzig[1]   **Jonathan Berant**[1,2]

[1]Blavatnik School of Computer Science, Tel-Aviv University
[2]Allen Institute for Artificial Intelligence
{jonathan.herzig,joberant}@cs.tau.ac.il

Abstract

Despite the success of sequence-to-sequence (seq2seq) models in semantic parsing, recent work has shown that they fail in *compositional generalization*, i.e., the ability to generalize to new structures built of components observed during training. In this work, we posit that a *span-based* parser should lead to better compositional generalization. we propose SPANBASEDSP, a parser that predicts a span tree over an input utterance, explicitly encoding how partial programs compose over spans in the input. SPANBASEDSP extends Pasupat et al. (2019) to be comparable to seq2seq models by (i) training from programs, without access to gold trees, treating trees as latent variables, (ii) parsing a class of non-projective trees through an extension to standard CKY. On GEOQUERY, SCAN and CLOSURE datasets, SPANBASEDSP performs similarly to strong seq2seq baselines on random splits, but dramatically improves performance compared to baselines on splits that require compositional generalization: from $61.0 \rightarrow 88.9$ average accuracy.

1 Introduction

The most dominant approach in recent years for semantic parsing, the task of mapping a natural language utterance to an executable program, has been based on sequence-to-sequence (seq2seq) models (Jia and Liang, 2016; Dong and Lapata, 2016; Wang et al., 2020, *inter alia*). In these models, the output program is decoded step-by-step (autoregressively), using an attention mechanism that softly ties output tokens to the utterance.

Despite the success of seq2seq models, recently, Finegan-Dollak et al. (2018) and Keysers et al. (2020) and Herzig and Berant (2019) demonstrated that such models fail at *compositional generalization*, that is, they do not generalize to program structures that were not seen at training time. For example, a model that observes at training time the questions *"What states border China?"* and *"What is the largest state?"* fails to generalize to questions such as *"What states border the largest state?"*. This is manifested in large performance drops on data splits designed to measure compositional generalization (*compositional splits*), and is in contrast to the generalization abilities of humans (Fodor and Pylyshyn, 1988).

In this work, we posit that the poor generalization of seq2seq models is due to fact that the input utterance and output program are only tied *softly* through attention. We revisit a more traditional approach for semantic parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Liang et al., 2011), where partial programs are predicted over short spans in the utterance, and are composed to build the program for the entire utterance. Such explicit inductive bias for compositionality should encourage compositional generalization.

Specifically, we propose to introduce such inductive bias via a span-based parser (Stern et al., 2017; Pasupat et al., 2019), equipped with the advantages of modern neural architectures. Our model, SPANBASEDSP, predicts for every span in the input *a category*, which is either a constant from the underlying knowledge-base, a composition category, or a null category. Given the category predictions for all spans, we can construct a tree over the input utterance and deterministically compute the output program. For example, in Figure 1, the category for the tree node covering the span *"New York borders ?"* is the composition category `join`, indicating the composition of the predicate `next_to_1` with the entity `stateid('new york')`.

Categories are predicted for each span independently, resulting in a very simple training procedure. CKY is used at inference time to find the best *span tree*, which is a tree with a category predicted at every node. The output program is computed from
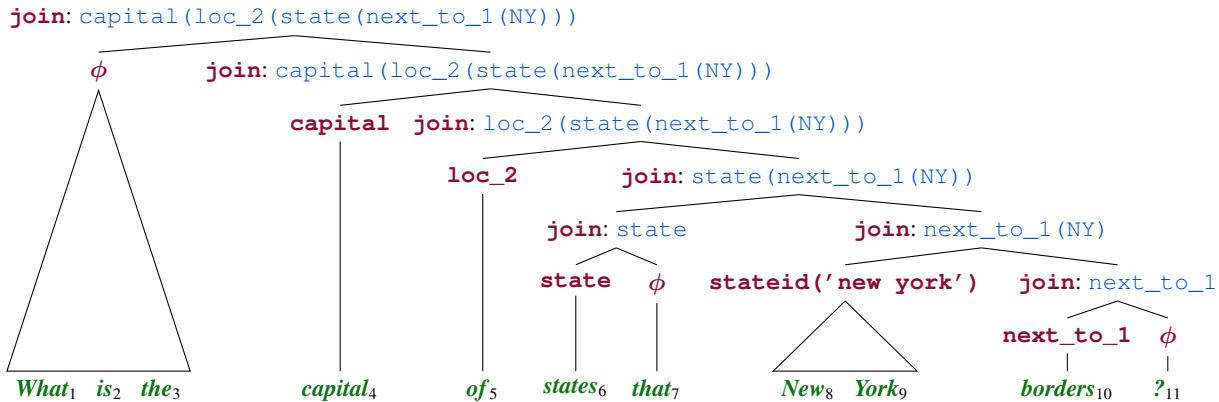
join: `capital(loc_2(state(next_to_1(NY))))`

ϕ **join**: `capital(loc_2(state(next_to_1(NY))))`

capital   **join**: `loc_2(state(next_to_1(NY)))`

loc_2   **join**: `state(next_to_1(NY))`

join: `state`   **join**: `next_to_1(NY)`

state ϕ **stateid('new york')**   **join**: `next_to_1`

next_to_1 ϕ

$What_1$ is_2 the_3 $capital_4$ of_5 $states_6$ $that_7$ New_8 $York_9$ $borders_{10}$ $?_{11}$

Figure 1: An example span tree. Nodes are annotated with categories (in bold). A node with a category `join` over the span (i,j), is annotated with its sub-program $z_{i:j}$. We abbreviate `stateid('new york')` to `NY`.

this tree in a bottom-up manner.

We enhance the applicability of span-based semantic parsers (Pasupat et al., 2019) in terms of both *supervision* and *expressivity*, by overcoming two technical challenges. First, we do not use gold trees as supervision, only programs with no explicit decomposition over the input utterance. To train with latent trees, we use a hard-EM approach, where we search for the best tree under the current model corresponding to the gold program, and update the model based on this tree. Second, some gold trees are non-projective, and cannot be parsed with a binary grammar. Thus, we extend the grammar of CKY to capture a class of non-projective structures that are common in semantic parsing. This leads to a model that is comparable and competitive with the prevailing seq2seq approach.

We evaluate our approach on three datasets, and find that SPANBASEDSP performs similarly to strong seq2seq baselines on standard i.i.d (random) splits, but dramatically improves performance on compositional splits, by 32.9, 34.6 and 13.5 absolute accuracy points on GEOQUERY (Zelle and Mooney, 1996), CLOSURE (Bahdanau et al., 2019), and SCAN (Lake and Baroni, 2018) respectively. Our code and data are available at `https://github.com/jonathanherzig/span-based-sp`.

2 Problem Setup

We define span-based semantic parsing as follows. Given a training set $\{(x^i, z^i)\}_{i=1}^M$, where x_i is an utterance and z_i is the corresponding program, our goal is to learn a model that maps a new utterance x to a *span tree* T (defined below), such that $\text{program}(T) = z$. The deterministic function $\text{program}(\cdot)$ maps span trees to programs.

Span trees A span tree T is a tree (see Figure 1) where, similar to constituency trees, each node covers a span (i,j) with tokens $x_{i:j} = (x_i, x_{i+1}, \ldots, x_j)$. A span tree can be viewed as a mapping from every span (i,j) to a single category $c \in \mathcal{C}$, where categories describe how the meaning of a node is derived from the meaning of its children. A category c is one of the following:

- Σ: a set of domain-specific categories representing domain constants, including entities and predicates. E.g., in Figure 1, `capital`, `state`, `loc_2` and `next_to_1` are binary predicates, and `stateid('new york')` is an entity.
- `join`: a category for a node whose meaning is derived from the meaning of its two children. At most one of the children's categories can be the ϕ category.
- ϕ: a category for (i) a node that does not affect the meaning of the utterance. For example, in Figure 1, the nodes that cover *"What is the"* and *"?"* are tagged by ϕ; (ii) spans that do not correspond to constituents (tree nodes).

Overall, the category set is $\mathcal{C} = \Sigma \cup \{\phi, \text{join}\}$. We also define the *terminal nodes set* $\Sigma^+ = \Sigma \cup \{\phi\}$, corresponding to categories that are directly over the utterance.

Computing programs for span trees Given a mapping from spans to categories specifying a span tree T, we use the function $\text{program}(\cdot)$ to find the program for T. Concretely, $\text{program}(T)$ iterates over the nodes in T bottom-up, and generates a program $z_{i:j}$ for each node covering the span (i,j).

The program $z_{i:j}$ is computed deterministically. For a node with category $c \in \Sigma$, $z_{i:j} = c$. For a `join` node over the span (i,j), we determine $z_{i:j}$ by composing the programs of its

children, $z_{i:s}$ and $z_{s,j}$ where s is the *split point*. As in Combinatory Categorical Grammar (Steedman, 2000), composition is simply function application, where a domain-specific type system is used to determine which child is the function and which is the argument (along with the exact argument position for predicates with multiple arguments). If the category of one of the children is ϕ, the program for $z_{i:j}$ is copied from the other child. E.g., in Figure 1, the span $(8, 9)$, where $z_{8:9} = $ `stateid('new york')` combines with the span $(10, 11)$, where $z_{10:11} = $ `next_to_1`. As $z_{10:11}$ is a binary predicate that takes an argument of type `state`, and $z_{8:9}$ is an entity of type `state`, the output program is $z_{8:11} = $ `next_to_1(stateid('new york'))`. If no combination is possible according to the type system, the execution of `program`(T) fails (§3.2).

Unlike seq2seq models, computing programs with span trees is explicitly compositional. Our main hypothesis is that this strong inductive bias should improve compositional generalization.

3 A Span-based Semantic Parser

Span-based parsing had success in both syntactic (Stern et al., 2017; Kitaev and Klein, 2018) and semantic parsing (Pasupat et al., 2019). The intuition is that modern sequence encoders are powerful, and thus we can predict a category for every span *independently*, reducing the role of global structure. This leads to simple and fast training.

Specifically, our parser is based on a model $p_\theta(T[i, j] = c)$, parameterized by θ, that provides for every span (i, j) a distribution over categories $c \in \mathcal{C}$. Due to the above independence assumption, the log-likelihood of a tree T is defined as:

$$\log p(T) = \sum_{i<j} \log p_\theta(T[i, j]), \qquad (1)$$

where, similar to Pasupat et al. (2019), the sum is over *all* spans $i < j$ and not only over constituents. We next describe the model $p_\theta(T[i, j])$ and its training, assuming we have access to gold span trees at training time (§3.1). We will later (§3.3) remove this assumption, and describe a CKY-based inference procedure (§3.2) that finds for every training example (x, z) the (approximately) most probable span tree T^*_{train}, such that `program`$(T^*_{\text{train}}) = z$. We use T^*_{train} as a replacement for the gold tree. Last, we present an extension of our model that covers a class of span trees that are non-projective (§3.4).

3.1 Model

We describe the architecture and training procedure of our model (SPANBASEDSP), assuming we are given for every utterance x a gold tree T, for which `program(T)` $= z$.

Similar to Pasupat et al. (2019), we minimize the negative log-likelihood $-\log p(T)$ (Eq. 1) for the gold tree T. The loss decomposes over spans into cross-entropy terms for every span (i, j). This effectively results in multi-class problem, where for every span $x_{i:j}$ we predict a category $c \in \mathcal{C}$. Training in this setup is trivial and does not require any structured inference.

Concretely, the architecture of SPANBASEDSP is based on a `BERT-base` encoder (Devlin et al., 2019) that yields a contextual representation $\mathbf{h}_i \in \mathbb{R}^{h_{\dim}}$ for each token x_i in the input utterance. We represent each span (i, j) by concatenating its start and end representations $[\mathbf{h}_i; \mathbf{h}_j]$, and apply a 1-hidden layer network to produce a real-valued score $\mathtt{s}(x_{i:j}, c)$ for a span (i, j) and category c:

$$\mathtt{s}(x_{i:j}, c) = [\mathbf{W}_2 \text{relu}(\mathbf{W}_1[\mathbf{h}_i; \mathbf{h}_j])]_{\text{ind}(c)}, \quad (2)$$

where $\mathbf{W}_1 \in \mathbb{R}^{250 \times 2h_{\dim}}$, $\mathbf{W}_2 \in \mathbb{R}^{|\mathcal{C}| \times 250}$, and $\mathtt{ind}(c)$ is the index of the category c. We take a softmax to produce the probabilities:

$$p_\theta(T[i, j] = c) = \frac{\exp[\mathtt{s}(x_{i:j}, c)]}{\sum_{c'} \exp[\mathtt{s}(x_{i:j}, c')]}, \quad (3)$$

and train the model with a cross-entropy loss averaged over all spans, as mentioned above.

3.2 CKY-based Inference

While we assume span-independence at training time, at test time we must output a valid span tree. We now describe an approximate K-best CKY algorithm that searches for the K most probable trees under $p(T)$, and returns the highest-scoring one that is *semantically valid*, i.e., that can be mapped to a program.[1] As we elaborate below, some trees cannot be mapped to a program, due to violations of the type system.

We start by re-writing our objective function, as proposed in Pasupat et al. (2019). Given our

---

[1] The requirement that trees are semantically valid is what prevents exact search.

```
S := join join | φ join
join := join join | join φ
```

Figure 2: CKY grammar defining the possible output trees.

definition for $p_\theta(T[i,j] = c)$, the log-likelihood is:

$$\log p(T) = \sum_{i<j} \log p_\theta(T[i,j]) =$$

$$\sum_{i<j} \left[\mathtt{s}(x_{i:j}, T[i,j]) - \log \sum_{c'} \exp[\mathtt{s}(x_{i:j}, c')] \right].$$

We shift the scoring function $\mathtt{s}(\cdot)$ for each span, such that the score for the ϕ category is zero:

$$\mathtt{s}'(x_{i:j}, \cdot) := \mathtt{s}(x_{i:j}, \cdot) - \mathtt{s}(x_{i:j}, \phi).$$

Because softmax is shift-invariant, we can replace $\mathtt{s}(\cdot)$ for $\mathtt{s}'(\cdot)$ and preserve correctness. This is motivated by the fact that ϕ nodes, such as the one covering *"What is the"* in Figure 1, do not affect the semantics of utterance. By shifting scores such that for all spans $\mathtt{s}'(x_{i:j}, \phi) = 0$, their score does not affect the overall tree score. Spans that do not correspond to tree nodes are labeled by ϕ and also do not affect the tree score.

Furthermore, as $\sum_{i<j} \log \sum_{c'} \exp[\mathtt{s}'(x_{i:j}, c')]$ does not depend on T at all, maximizing $\log p(T)$ is equivalent to maximizing the tree score:

$$\mathtt{S}(T) := \sum_{i<j} \mathtt{s}'(x_{i:j}, T[i,j]).$$

This scoring function can be maximized using CKY (Cocke, 1969; Kasami, 1965; Younger, 1967). We now propose a grammar, which imposes further restrictions on the space of possible output trees at inference time.

We use a small grammar $G = (N, \Sigma^+, R, S)$, where $N = \{\mathtt{S}, \mathtt{join}\}$ is the set of non-terminals, Σ^+ is the set of terminals (defined in §2), R is a set of four rules detailed in Figure 2, and S is a special start symbol. The four grammar rules impose the following constraints on the set of possible output trees: (a) a $\mathtt{join}$ or S node can have at most one ϕ child, as explained in §2; (b) nodes with no semantics combine with semantic elements on their left; (c) except at the root where they combine with elements on their right. Imposing such consistent tree structure is useful for training SPANBASEDSP when predicted trees are used for training (§3.3).

Algorithm 1: CKY inference algorithm

Input: $\forall i, j, c : s(x_{i:j}, c), G = (N, \Sigma^+, R, S), x$
Output: π - scores for each span and non-terminal
1 **for** $1 \leq i \leq j \leq |x|$ **do**
2    $\pi(i, j, \mathtt{join}) \leftarrow \max_{c \in \Sigma} \mathtt{s}'(x_{ij}, c)$
3    $\pi(i, j, \phi) \leftarrow \mathtt{s}'(x_{ij}, \phi)$ // equals zero
4 **for** $1 \leq i \leq j \leq |x|$ **do**
5   **for** $X \in N$ **do**
6      $\text{temp} \leftarrow \max_{\substack{(X \rightarrow YZ) \in R \\ s \in i...(j-1)}} [\mathtt{s}'(x_{ij}, \mathtt{join}) +$
     $\pi(i, s, Y) + \pi(s+1, j, Z)]$
7      $\pi(i, j, X) \leftarrow \max(\text{temp}, \pi(i, j, \mathtt{join}))$
8 **return** π

The grammar G can generate trees that are not semantically valid. For example, we could generate the program `capital(placeid('mount mckinley'))`, which is semantically vacuous. We use a domain-specific type system and assign the score $\mathtt{S}(T) = -\infty$ to every tree that yields a semantically invalid program. This global factor prevents exact inference, and thus we perform K-best parsing, keeping the top-K ($K = 5$) best trees for every span (i, j) and non-terminal.

Alg. 1 summarizes CKY inference, that outputs $\pi(i, j, X)$, the maximal score for a tree with non-terminal root X over the span (i, j). In Lines 1-3 we initialize the parse chart, by going over all spans and setting $\pi(i, j, \mathtt{join})$ to the top-K highest scoring domain constants (Σ), and fixing the score for ϕ to be zero. We then perform the typical CKY recursion to find the top-K trees that can be constructed through composition (Line 6), merge them with the domain constants found during initialization (Line 7), and keep the overall top-K trees.

Once inference is done, we retrieve the top-K trees from $\pi(1, |x|, S)$, iterate over them in descending score order, and return the first tree T^* that is semantically valid.

3.3 Training without Gold Trees

We now remove the assumption of access to gold trees at training time, in line with standard supervised semantic parsing, where only the gold program z is given, without its decomposition over x. This can be viewed as a *weakly-supervised* setting, where the correct span tree is a discrete latent variable. In this setup, our goal is to maximize

$$\log p(z \mid x) = \log \sum_{T:program(T)=z} p(T)$$

$$\approx \log \underset{T:program(T)=z}{\text{argmax}} p(T).$$

Because marginalizing over trees is intractable, we take a hard-EM approach (Liang et al., 2017; Min et al., 2019), and replace the sum over trees with an argmax. More concretely, to approximately solve the argmax and find the highest scoring tree, T^*_{train}, we employ a constrained version of Alg. 1, that prunes out trees that cannot generate z.

We first remove all predictions of constants that do not appear in z by setting their score to $-\infty$:

$$\forall c \in \{\Sigma \setminus \mathtt{const}(z)\}, i, j : s'(x_{i:j}, c) := -\infty,$$

where $\mathtt{const}(z)$ is the set of domain constants appearing in z. Second, we allow a composition of two nodes covering spans (i, s) and (s, j) only if their sub-programs $z_{i:s}$ and $z_{s:j}$ can compose according to z. For instance, in Figure 1, a span with the sub-program $\mathtt{capital}$ can only compose with a span with the sub-program $\mathtt{loc\_2}(\cdot)$. After running this constrained CKY procedure we return the highest scoring tree that yields the correct program, T^*_{train}, if one is found. We then treat the span structure of T^*_{train} as labels for training the parameters of SPANBASEDSP.

Past work on weakly-supervised semantic parsing often used maximum marginal likelihood, especially when training from denotations only (Guu et al., 2017). In this work, we found hard-EM to be simple and sufficient, since we are given the program z that provides a rich signal for guiding search in the space of latent trees.

Exact match features The challenge of weakly-supervised parsing is that SPANBASEDSP must learn to map language phrases to constants, and how the span tree is structured. To alleviate the language-to-constant problem we add an exact match feature, based on a small lexicon, indicating whether a phrase in x matches the language description of a category $c \in \Sigma$. These features are considered in SPANBASEDSP when some phrase matches a category from Σ, updating the score $\mathtt{s}(x_{i:j}, c)$ to be: $[\mathbf{W}_2 \text{relu}(\mathbf{W}_1[\mathbf{h}_i; \mathbf{h}_j])]_{\text{ind}(c)} + \lambda \delta(x_{i:j}, c)$, where $\delta(x_{i:j}, c)$ is an indicator that returns 1 if $c \in \mathtt{lexicon}[x_{i:j}]$, and 0 otherwise, and λ is a hyper-parameter that sets the feature's importance.

We use two types of $\mathtt{lexicon}[\cdot]$ functions. In the first, the lexicon is created automatically to map the names of entities (not predicates), as they appear in Σ, to their corresponding constant (e.g., $\mathtt{lexicon}[\textit{"new york"}] = \mathtt{stateid('new york')}$). This endows SPANBASEDSP with a *copying mechanism*, similar to
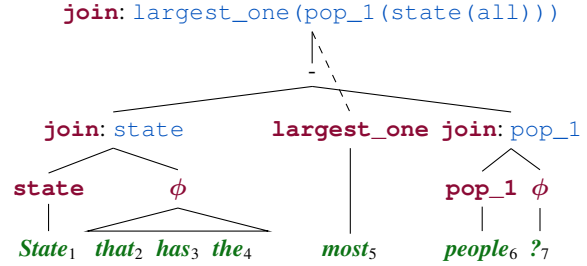

Figure 3: An example of a non-projective tree. The corresponding program z is at the root.

seq2seq models, for predicting entities unseen during training. In the second lexicon we manually add no more than two examples of language phrases for each constant in Σ. E.g., for the predicate $\mathtt{next\_to\_1}$, we update the lexicon to include $\mathtt{lexicon}[\textit{"border"}] = \mathtt{lexicon}[\textit{"borders"}] = \mathtt{next\_to\_1}$. This requires minimal manual work (if no language phrases are available), but is done only once, and is common in semantic parsing (Zettlemoyer and Collins, 2005; Wang et al., 2015; Liang et al., 2017).

3.4 Non-Projective Trees

Our span-based parser assumes composition can only be done for adjacent spans that form together a contiguous span. However, this assumption does not always hold (Liang et al., 2011). For example, in Figure 3, while the predicate $\mathtt{pop\_1}$ should combine with the predicate $\mathtt{state}$, the spans they align to (*"people"* and *"state"* respectively) are not contiguous, as they are separated by *"most"*, which contributes the semantics of a superlative.

In constituency parsing, such non-projective structures are treated by adding rules to the grammar G (Maier et al., 2012; Corro, 2020; Stanojević and Steedman, 2020). We identify one specific class of non-projective structures that is frequent in semantic parsing (Figure 3), and expand the grammar G and the CKY Algorithm to support this structure. Specifically, we add the ternary grammar rule $\mathtt{join} := \mathtt{join}\ \mathtt{join}\ \mathtt{join}$. During CKY, when calculating the top-K trees for spans (i, j) (line 6 in Alg. 1), we also consider the following top-K scores for the non-terminal $\mathtt{join}$:

$$\max_{\substack{s_1 \in i...(j-2) \\ s_2 \in (s_1+1)...(j-1)}} [\mathtt{s}'(x_{ij}, \mathtt{join}) + \pi(i, s_1, \mathtt{join})$$
$$+ \pi(s_1 + 1, s_2, \mathtt{join}) + \pi(s_2 + 1, j, \mathtt{join})].$$

These additional trees are created by going over all possible ways of dividing a span (i, j) into three

912

Dataset	Split	train	dev	test
SCAN-SP	IID	13,383	3,345	4,182
	RIGHT	12,180	3,045	4,476
	AROUNDRIGHT	12,180	3,045	4,476
CLEVR	IID	694,689	5,000	149,991
	CLOSURE	694,689	5,000	25,200
GEOQUERY	IID	540	60	280
	TEMPLATE	544	60	276
	LENGTH	540	60	280

Table 1: Number of examples for all datasets.

parts. The score of the sub-tree is then the sum of the score of the root added to the scores of the three children. To compute the program for such ternary nodes, we again use our type system, where we first compose the programs of the two outer spans (i, s_1) and $(s_2 + 1, j)$ and then compose the resulting program with the program corresponding to the span $(s_1 + 1, s_2)$. Supporting ternary nodes in the tree increases the time complexity of CKY from $O(n^3)$ to $O(n^4)$ for our implementation.[2]

4 Experiments and Results

We now present our experimental evaluation, which demonstrates the advantage of span-based parsing for compositional generalization. We compare to baseline models over two types of data splits: (a) IID *split*, where the training and test sets are sampled from the same distribution, and (b) *compositional split*, where the test set includes structures that are unseen at training time. Details on the experimental setup are given in Appendix A.

4.1 Datasets

We evaluate on the following datasets (Table 1).

GEOQUERY Contains 880 questions about US geography (Zelle and Mooney, 1996), using the FunQL formalism (Kate et al., 2005). For the IID split, we use the standard train/test split, randomly sampling 10% of the training set for development. We additionally use two compositional splits based on program templates (TEMPLATE) and on program lengths (LENGTH).

For the compositional split, TEMPLATE, we use the procedure from Finegan-Dollak et al. (2018) and split the 880 examples by templates. A template is created by anonymizing entities in the program to their type (both stateid('new

---

york') and stateid('utah') are anonymized to STATE). We then split to train/development/test sets, such that all examples that share a template are assigned to the same set. We also verify that the sizes of theses sets are as close as possible to the IID split.

For the compositional split, LENGTH, we sort the dataset by program token length and take the longest 280 examples to be the test set. We then randomly split the shortest 600 examples between the train and development set, where we take 10% of the 600 examples for the latter.

CLEVR and CLOSURE CLEVR (Johnson et al., 2017) contains synthetic questions, created using 80 templates, over synthetic images with multiple objects of different shapes, colors, materials and sizes (example in Fig. 4 in the Appendix). The recent CLOSURE dataset (Bahdanau et al., 2019), includes seven new question templates that are created by combining referring expressions of various types from CLEVR in new ways.

We use the semantic parsing version of these datasets, where each image is described by a *scene* (knowledge-base) that holds the attributes and positional relations of all objects. We use programs in the DSL version from Mao et al. (2019).

For our experiments, we take $5K$ examples from the original CLEVR training set and treat them as our development set. We use the other $695K$ examples as training data for our baselines. Importantly, we *only* use $10K$ training examples for SPANBASEDSP to reduce training time. We then create an IID split where we test on the CLEVR original development set (test scenes are not publicly available). We additionally define the CLOSURE split, that tests compositional generalization, where we test on CLOSURE.

SCAN-SP SCAN (Lake and Baroni, 2018) contains natural language navigation commands that are mapped to action sequences (x and y in Fig. 5 in the Appendix). As SCAN lacks programs, we automatically translate the input to programs (z in Fig. 5) to crate the semantic parsing version of SCAN, denoted SCAN-SP (more details are given in Appendix B). We experiment with the random SIMPLE split from Lake and Baroni (2018) as our IID split. we further use the *primitive right* (RIGHT) and *primitive around right* (AROUNDRIGHT) compositional splits from Loula et al. (2018). For each split we randomly assign 20% of the training set

Model	SCAN-SP						CLEVR				GEOQUERY					
	IID		RIGHT		AROUNDRIGHT		IID		CLOSURE		IID		TEMPLATE		LENGTH	
	dev	test	dev	test	dev	test	dev	test	dev	test	dev	test	dev	test	dev	test
SEQ2SEQ	100	99.9	100	11.6	100	0.0	100	100	100	59.5	83.3	78.5	71.6	46.0	86.7	24.3
+ELMo	100	100	100	54.9	100	41.6	100	100	100	64.2	83.3	79.3	83.3	50.0	86.7	25.7
BERT2SEQ	99.9	100	99.9	77.7	99.9	95.3	100	100	100	56.4	88.3	81.1	85.0	49.6	90.0	26.1
GRAMMAR	100	100	100	0.0	100	4.2	100	100	100	51.3	78.3	72.1	76.7	54.0	81.7	24.6
BART	100	100	100	50.5	100	100	100	100	100	51.5	93.3	87.1	86.7	67.0	90.0	19.3
END2END	-	-	-	-	-	-	99.9	99.8	99.9	63.3	-	-	-	-	-	-
SPANBASEDSP	100	100	100	**100**	100	**100**	97.0	96.7	98.9	**98.8**	88.3	86.1	93.3	**82.2**	95.0	**63.6**
-lexicon	100	100	100	**100**	100	**100**	99.4	99.3	98.5	88.6	88.3	78.9	86.7	65.9	90.0	41.4
-non projective	-	-	-	-	-	-	-	-	-	-	85.0	80.0	90.0	80.2	93.3	59.3
+gold trees	100	100	100	100	100	100	100	96.8	100	96.7	91.2	86.4	100	81.8	96.7	68.6

Table 2: Denotation accuracies for all models, including SPANBASEDSP ablations. For both CLEVR splits, SPANBASEDSP only trains on $10K$ examples, in comparison to $695K$ for the baselines.

for development.

4.2 Baselines

SEQ2SEQ Similar to Finegan-Dollak et al. (2018), our baseline parser is a standard seq2seq model (Jia and Liang, 2016) that encodes the utterance x with a BiLSTM encoder over pre-trained GloVe (Pennington et al., 2014) or ELMO (Peters et al., 2018) embeddings, and decodes the program with an attention-based LSTM decoder (Bahdanau et al., 2015) assisted by a copying mechanism for handling entities unseen during training time (Gu et al., 2016).

BERT2SEQ Same as SEQ2SEQ, but we replace the BiLSTM encoder with `BERT-base`, which is identical to the encoder of SPANBASEDSP.

GRAMMAR Grammar-based decoding has been shown to improve performance on IID splits (Krishnamurthy et al., 2017; Yin and Neubig, 2017). Because decoding is constrained by the grammar, the model outputs only valid programs, which could potentially improve performance on compositional splits. We use the grammar from (Wong and Mooney, 2007) for GEOQUERY, and write grammars for SCAN-SP and CLEVR + CLOSURE. The model architecture is identical to SEQ2SEQ.

BART We additionally experiment with `BART-base` (Lewis et al., 2020), a seq2seq model pre-trained as a denoising autoencoder.

END2END Semantic parsers generate a program that is executed to retrieve an answer. However, other end-to-end models directly predict the answer from the context without an executor, where the context can be an image (Hudson and Manning, 2018; Perez et al., 2018), a table (Herzig et al., 2020), etc. Because CLEVR and CLOSURE have a closed set of 28 possible answers and a short context (the scene), they are a good fit for end-to-end approaches. To check whether end-to-end models generalize compositionally, we implement the following model. We use `BERT-base` to encode the concatenation of the input x to a representation of all objects in the scene. Each scene object is represented by *adding* learned embeddings of all of its attributes: shape, material, size, color, and relative positional rank (from left to right, and from front to back). We fine-tune the model on the training set using cross-entropy loss, where the `[CLS]` token is used to predict the answer.

4.3 Main Results

Table 2 shows denotation accuracies for all baselines (top part) and our SPANBASEDSP model (middle part). For SPANBASEDSP, We also ablate the use of the manually constructed lexicon (§3.3) and the non-projective extension to CKY (§3.4), which is relevant only for GEOQUERY, where non-projective structures are more frequent.

The table shows that all baselines generalize well on the IID split, but suffer from a large accuracy drop on the compositional splits (except BERT2SEQ and BART on AROUNDRIGHT). For instance, on the compositional CLOSURE split, all baselines achieve accuracy in the range of $51.3 - 64.2$, while performing perfectly on the IID split. Conversely, SPANBASEDSP performs almost identically on both splits. SPANBASEDSP

attains near-perfect performance on all SCAN-SP and CLEVR splits, despite training on *only* $10K$ examples from CLEVR compared to $695K$ training examples for the baselines (70x less data). On GEOQUERY, SPANBASEDSP performs similarly to other semantic parsers on the IID split (Dong and Lapata, 2016), and loses just 4 points on the compositional TEMPLATE split. On the LENGTH split, SPANBASEDSP yields an accuracy of 63.6, substantially outperforming all baselines by more than 37 accuracy points.

Our ablations show that the lexicon is crucial for GEOQUERY, which has a small training set. In this setting, learning the mapping from language phrases to predicates is challenging. Ablating non-projective parsing also hurts performance for GEO-QUERY, and leads to a reduction of 2-6 points for all of the splits.

4.4 Decomposition Analysis

We now analyze whether trees learned by SPAN-BASEDSP are similar to gold trees. For this analysis we semi-automatically annotate our datasets with gold trees. We do this by manually creating a domain-specific lexicon for each dataset (extending the lexicon from §3.3), mapping domain constants to possible phrases in the input utterances. We then, for each example, traverse the program tree (rather than the span tree) bottom-up and annotate join and ϕ categories for spans in the utterance, aided by manually-written domain-specific rules. In cases where the annotation is ambiguous, e.g., examples with more than two instances of a specific domain constant, we do not produce a gold tree.

We manage to annotate 100%/94.9%/95.9% of the examples in SCAN-SP/ GEOQUERY/ CLEVR + CLOSURE respectively in this manner. We verify the correctness of our annotation by training SPANBASEDSP from our annotated gold trees (bottom part of Table 2). The results shows that training from these "gold" trees leads to similar performance as training only from programs.

We then train SPANBASEDSP from gold programs, as explained in §3.3, and calculate F_1 test scores, comparing the predicted span trees to the gold ones. F_1 is computed between the two sets of labeled spans, taking into account both the spans and their categories, but excluding spans with the ϕ category that do not contribute to the semantics.

Table 3 shows that for GEOQUERY the trees SPANBASEDSP predicts are similar to the gold

Dataset	Split	F_1
SCAN-SP	IID	100
	RIGHT	100
	AROUNDRIGHT	100
CLEVR	IID	70.6
	CLOSURE	70.6
GEOQUERY	IID	94.7
	TEMPLATE	91.6
	LENGTH	93.7

Table 3: F_1 scores on the test set w.r.t to the semi-automatically annotated gold trees.

trees (with 94.7, 91.6 and 93.7 F_1 scores for the IID, TEMPLATE and LENGTH splits respectively), and in SCAN-SP we predict perfect trees. On CLEVR, we get a lower F_1 score of 70.6 for both the IID and CLOSURE splits. However, when manually inspecting predicted trees on the IID split, we notice that predicted trees that are not identical to gold trees, are actually correct. This happens in cases where multiple gold trees are possible. For instance, in Figure 4 (in the Appendix), the span $x_{13:15} =$ *"matte block ?"* can be either parsed as [matte [block ?]], as in the figure, or [[matte block] ?]. This phenomena is common in CLEVR and CLOSURE, as span trees tend to be deep, and thus have more ambiguity.

4.5 Limitations

Our approach assumes a one-to-one mapping between domain constants and their manifestation as phrases in language. This leads to strong results on compositional generalization, but hurts the flexibility that is sometimes necessary in semantic parsing. For example, in some cases predicates do not align explicitly to a phrase in the utterance or appear several times in the program but only once in the utterance (Berant et al., 2013; Pasupat and Liang, 2015). This is evident in text-to-SQL parsing, where an utterance such as *"What is the minimum, and maximum age of all singers from France?"* is mapped to `SELECT min(age) , max(age) FROM singer WHERE country='France'`. Here, the constant `age` is mentioned only once in language (but twice in the program), and `country` is not mentioned at all. Thus, our approach is more suitable for formalisms where there is tighter alignment between the natural and formal language.

In addition, while we handle a class of non-projective trees (§3.4), there are other non-projective structures that SPANBASEDSP can not

parse. Extending CKY to support all structures from Corro (2020) leads to a time complexity of $O(n^6)$, which might be impractical.

5 Related Work

Until the neural era, semantic parsers used a lexicon and composition rules to predict partial programs for spans and compose them until a full program is predicted, and typically scored with a log-linear model given features over the utterance and the program (Zettlemoyer and Collins, 2005; Liang et al., 2011). In this work, we use a similar compositional approach, but take advantage of powerful span representations based on modern neural architectures.

The most similar work to ours is by Pasupat et al. (2019), who presented a neural span-based semantic parser. While they focused on training using projective gold trees (having more supervision and less expressivity than seq2seq models) and testing on i.i.d examples, we handle non-projective trees, given only program supervision, rather than trees. More importantly, we show that this approach leads to dramatic gains in compositional generalization compared to autoregressive parsers.

In recent years, work on compositional generalization in semantic parsing mainly focused on the poor performance of parsers in compositional splits (Finegan-Dollak et al., 2018), creating new datasets that require compositional generalization (Keysers et al., 2020; Lake and Baroni, 2018; Bahdanau et al., 2019), and proposing specialized architectures mainly for the SCAN task (Lake, 2019; Nye et al., 2020; Gordon et al., 2020; Liu et al., 2020; Gupta and Lewis, 2018). In this work we present a general-purpose architecture for semantic parsing that incorporates an inductive bias towards compositional generalization. Finally, concurrently to us, Shaw et al. (2020) induced a synchronous grammar over program and utterance pairs and used it to introduce a compositional bias, showing certain improvements over compositional splits.

6 Conclusion

Seq2seq models have become unprecedentedly popular in semantic parsing but struggle to generalize to unobserved structures. In this work, we show that our span-based parser, SPANBASEDSP, that precisely describes how meaning is composed over the input utterance leads to dramatic improvements in compositional generalization. In future work, we plan to investigate ways to introduce the explicit compositional bias, inherent to SPANBASEDSP, directly into seq2seq models.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Dzmitry Bahdanau, Harm de Vries, Timothy J O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron Courville. 2019. Closure: Assessing systematic generalization of clevr models. *arXiv preprint arXiv:1912.05783*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

John Cocke. 1969. *Programming Languages and Their Compilers: Preliminary Notes*. New York University, USA.

Caio Corro. 2020. Span-based discontinuous constituency parsing: a family of exact chart-based algorithms with time complexities from O(n^6) down to O(n^3). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2753–2764, Online. 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.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. *Connectionism and Cognitive Architecture: A Critical Analysis*, page 3–71. MIT Press, Cambridge, MA, USA.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2020. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Nitish Gupta and Mike Lewis. 2018. Neural compositional denotational semantics for question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2152–2161, Brussels, Belgium. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Drew A Hudson and Christopher D Manning. 2018. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations (ICLR)*.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*.

T. Kasami. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA†.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, page 1062–1068. AAAI Press.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *ICLR*.

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.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *EMNLP*.

Brenden M Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems 32*, pages 9791–9801. Curran Associates, Inc.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining 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.

C. Liang, J. Berant, Q. Le, and K. D. F. N. Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Association for Computational Linguistics (ACL)*.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599, Portland, Oregon, USA. Association for Computational Linguistics.

Q. Liu, Shengnan An, Jianguang Lou, B. Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. *ArXiv*, abs/2006.10627.

João Loula, Marco Baroni, and Brenden Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 108–114, Brussels, Belgium. Association for Computational Linguistics.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. PLCFRS parsing revisited: Restricting the fan-out to two. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, pages 126–134, Paris, France.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard EM approach for weakly supervised question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2851–2864, Hong Kong, China. Association for Computational Linguistics.

Maxwell I Nye, Armando Solar-Lezama, Joshua B Tenenbaum, and Brenden M Lake. 2020. Learning compositional rules via neural program synthesis. *arXiv preprint arXiv:2003.05562*.

Panupong Pasupat, Sonal Gupta, Karishma Mandyam, Rushin Shah, Mike Lewis, and Luke Zettlemoyer. 2019. Span-based hierarchical semantic parsing for task-oriented dialog. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1520–1526, Hong Kong, China. Association for Computational Linguistics.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *AAAI*.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.

Miloš Stanojević and Mark Steedman. 2020. Span-based LCFRS-2 parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 111–121, Online. Association for Computational Linguistics.

Mark Steedman. 2000. *The syntactic process*, volume 24. MIT press Cambridge, MA.

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

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

D.H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, USA. AUAI Press.
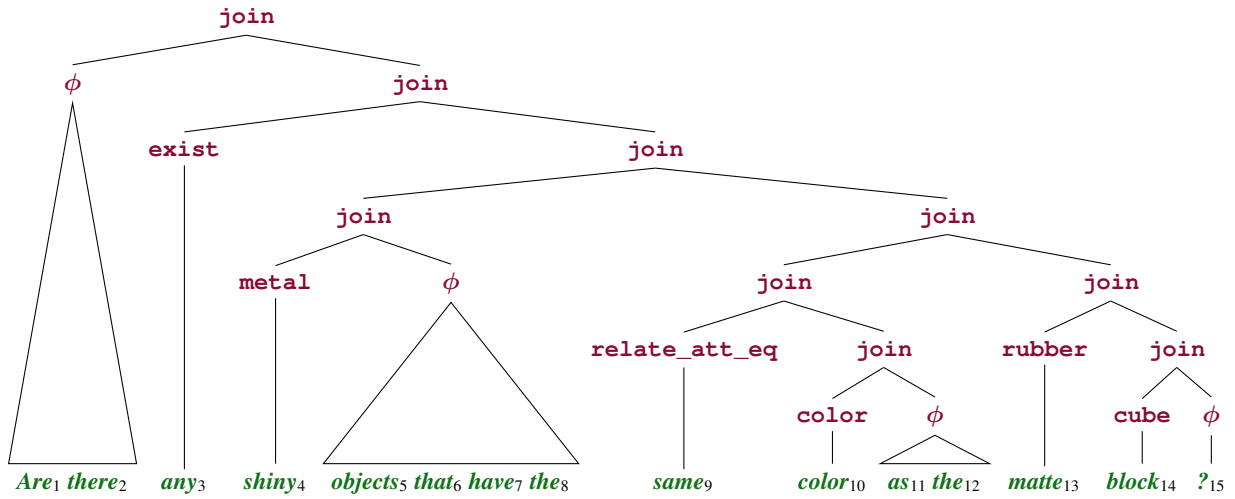
Appendix

A   Experimental Setup

We evaluate models with denotation accuracy, that is, the proportion of questions for which the denotations of the predicted and gold programs are identical. For SPANBASEDSP, we selected a learning rate of $1e^{-5}$, considering the values $[1e^{-4}, 1e^{-5}, 1e^{-6}]$, and use a batch size of 5. For our baselines, we tune the learning rate, batch size, and dropout. We choose all hyper-parameters by early-stopping with respect to development set denotation accuracy. Training SPANBASEDSP takes between 2 hours for GEOQUERY up to 20 hours for CLEVR on a single GeForce GTX 1080 GPU. Our seq2seq baselines are from AllenNLP (Gardner et al., 2018), and all `BERT-base` (110M parameters) implementations are from the Transformers library (Wolf et al., 2019).

We additionally implement executors that calculate the denotation of a program with respect to the corresponding scene for CLEVR +CLOSURE and retrieve an action sequence as the denotation for SCAN-SP.

B   Generating SCAN-SP

To create a semantic parsing version of SCAN, we introduce the binary predicates `and`, `after`, `walk`, `jump`, `run`, `look` and `turn`. We additionally introduce the unary predicates `twice` and `thrice`. Finally, we introduce the constants `left`, `right`, `opposite` and `around`. We then construct a synchronous context-free grammar (SCFG) that parses utterances in SCAN into programs in SCAN-SP by utilizing the constants above and simple composition rules. Finally, we use our grammar to parse all utterances in SCAN to generate the programs in SCAN-SP.

Figure 4 tree:

```
                              join
              ┌────────────────┴──────────────┐
              φ                               join
         ┌────┴────┐                ┌───────────┴───────────┐
       exist                      join                     join
                            ┌───────┴───────┐        ┌───────┴───────┐
                          join            join      join            join
                      ┌─────┴─────┐   ┌─────┴───┐ ┌───┴───┐     ┌──────┴──────┐
                    metal        φ  relate_att_eq join rubber  join
                                              ┌────┴────┐          ┌────┴────┐
                                            color       φ        cube       φ
```

Are₁ there₂   any₃   shiny₄   objects₅ that₆ have₇ the₈   same₉   color₁₀ as₁₁ the₁₂   matte₁₃   block₁₄   ?₁₅

x: *Are there any shiny objects that have the same color as the matte block?*

z: `exist(filter(metal,relate_att_eq(color,filter(rubber,cube,scene()))))`

Figure 4: An example span tree from CLEVR, along with its utterance x and program z. Here, the type system is used in `join` nodes to deterministically invoke the predicates `filter` and `scene` where needed. Sub-programs are omitted due to space reasons.

Figure 5 tree:

```
J: after(walk(r),twice(turn(l,op)))
        ┌──────────────┴──────────────┐
J: after(walk(r),·)          J: twice(turn(l,op))
     ┌──────┴──────┐              ┌──────┴──────┐
J: walk(r)  after          J: turn(l,op)  twice
   ┌──┴──┐                    ┌──────┴──────┐
  walk   r              J: turn(·,op)     l
                          ┌────┴────┐
                        turn       op
```

Walk₁   right₂   after₃   turn₄   opposite₅   left₆   twice₇
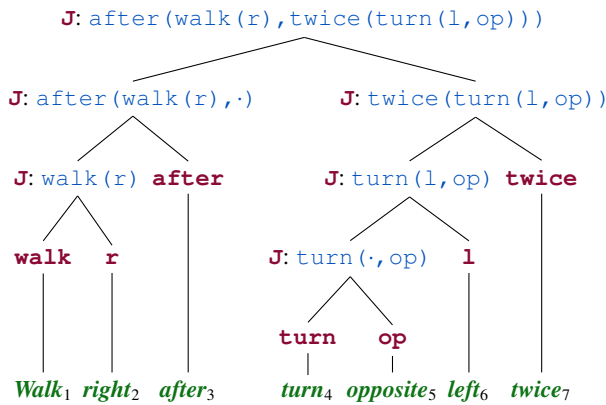
x: *Walk right after turn opposite left twice*

z: `after(walk(r),twice(turn(l,op)))`

y: `LTURN LTURN LTURN LTURN RTURN WALK`

Figure 5: An example span tree from SCAN-SP, along with its utterance x, program z and action sequence y. The category `join` is abbreviated to `J`.