THE UNIVERSITY *of* EDINBURGH

Enhancing Structural Inductive Biases of Sequence-to-Sequence Models for Semantic Parsing and Beyond

Matthias Moritz Lindemann

Doctor of Philosophy
Institute for Language, Cognition and Computation
School of Informatics
University of Edinburgh
2025

Abstract

In recent years, sequence-to-sequence models such as Transformers have been very successfully applied to an incredibly wide range of problems in Natural Language Processing, ranging from low-level tasks such as grapheme-to-phoneme conversion to more high-level tasks such as semantic parsing and machine translation. Sequence-to-sequence models that are commonly applied to such tasks have relatively weak *inductive biases*, i.e. they have little prior knowledge about the nature of the task they are applied to and learn virtually everything from data. While this makes them extremely versatile, it also makes them brittle when the training data provides only a weak signal. In particular, this is the case when (i) there is only a small amount of training data or (ii) if the model is applied *outside* of the training distribution. Sequence-to-sequence models with weak inductive biases struggle with *structural* generalization, e.g. generalization to unseen combinations of syntactic structures and deeper recursion than seen during training. While scaling pretraining to ever larger datasets helps, as of yet, scaling alone does not seem to close the gap completely. The goal of this thesis is to design, implement and evaluate methods for introducing inductive biases into sequence-to-sequence models to enable structural generalization.

This thesis consists of two parts. The first part develops two sequence-to-sequence models whose inductive biases for structure derive from their architecture. The idea is to explicitly model correspondences between fragments of the input and fragments of the output. This is achieved by a two-stage process: First, for each input token, the tokens it 'contributes' to the output are predicted without committing to their final order. Second, the output tokens are rearranged into the right order. These methods improve structural generalization in the context of semantic parsing and also perform well for other syntax-sensitive sequence-to-sequence tasks.

In the second part, we present a general framework that injects structural inductive biases into a standard sequence-to-sequence architecture, the Transformer, by means of a particular pretraining and fine-tuning procedure with synthetic data. The approach is based on the observation that it is often possible to operationalize an inductive bias as a *family* of symbolically defined functions, and it tends to be cheap to generate instances of this family automatically. The pretraining objective is to match the behavior of the entire family of functions. This framework is applied in two settings. First, taking Finite State Transducers (FSTs) as the family of functions, we pretrain a model to match the behavior of FSTs and thereby inject an FST-like inductive bias into a Transformer. Analysis of the hidden representation reveals that this procedure makes the model *simulate* transitions between FST states in its hidden representations without being explicitly trained to do so. Second, for semantic parsing, we operationalize the inductive bias of interest as transformations of syntax trees. We further pretrain a model to perform these transformations, which enhances structural generalization for semantic parsing and learning from small amounts of labeled data for syntactic tasks.

Lay Summary

THE field of Natural Language Processing (NLP) has made tremendous progress in recent years, with systems such as ChatGPT making the headlines. These systems are powerful statistical models that are based on large artificial neural networks. Artificial neural networks are loosely inspired by how brains work and very good at learning patterns from data. In a procedure known as pretraining, models like ChatGPT have learned about language and the subjects people talk about from vast amounts of text data from the internet, which gives these models unprecedented capabilities to generate and understand text. What do we mean by a model 'understanding' text? In this thesis, I view 'understanding' in a narrow technical sense as the ability of the model to translate a sentence into a well-structured machine-readable representation of the meaning. For example, this means a sentence like 'Show me train journeys from London to Edinburgh on Saturday' could be translated into a machine-readable instruction. This instruction can then be executed on a database with timetables and return a list of train journeys to the user.

Despite their tremendous success, previous research has shown that pretrained models tend to misunderstand sentences that do not look like the ones they have encountered before during training, but use the same underlying rules of language. For example, if a model understands the sentences 'Which trains from London to Edinburgh take less than 5 hours?' and 'Which trains arrive in Edinburgh before 2 pm and have a dining car?' as well as 'Which trains stop in Dunbar?', it might still fail to understand a question that combines all requests, i.e. 'Which trains from London to Edinburgh stop in Dunbar, have a dining car, arrive before 2 pm and take less than 5 hours?'. Commonly used neural network models can struggle to understand in such situations because they have insufficient knowledge about the structural principles underlying language, including that there are arbitrarily long sentences, e.g. composed of phrases combined with 'and' or commas. Arguably, a model that truly understands language should have a firm grasp of such structural principles and should be able to understand long sentences despite being trained with short sentences only.

In this thesis, I equip neural network models with more knowledge about structural principles of language so that they generalize better and as a result understand sentences more robustly. This thesis consists of two parts with different angles on this. In the first part, I propose an approach that frames the translation of a sentence to its meaning representation as *editing* the sentence, e.g. translating individual words into fragments of the meaning representation or re-arranging the fragments. The technical challenge is to learn from data which kind of edit to apply to which part of the sentence. In the second part, I introduce a method that can be used to inform an existing pretrained model about structural principles by further pretraining it on data that follows the structural principles we want to teach. Notably, this data is generated with straightforward programs, so we can generate as much of it as we like with little effort.

By evaluating how often models correctly process hundreds of sentences they have never encountered before, I show that both approaches make models generalize better. The approach in the second part is easier to apply more generally and requires less expertise to adapt to new tasks that we wish a model to perform, such as 'translating' a written word to a description how it is actually pronounced.

Acknowledgements

First, I would like to thank my advisor, Ivan Titov. I couldn't have been more fortunate with my advisor. Thank you so much for all the thoughts, suggestions, comments on drafts, and critical perspective when it was needed, but also the kindness, generosity, and encouragement, such as during a memorable conversation $1\frac{1}{2}$ years and a streak of three failed ideas into my PhD. I learned so much from Ivan about finding and approaching research questions, and I'm also grateful for his incredible breadth of knowledge and countless pointers to the literature, including to seemingly obscure Soviet-era maths.

I'd also like to express a big thank you to my other advisor, Alexander Koller. His advice has been invaluable for developing my writing skills, for ideas on how to debug my models, and most of all for my research direction over the years, starting with my first steps in the world of research. I'm grateful for Alexander's perspective, that has so often been complementary to Ivan's. To name just one example, I think his views and remarks on recursion without trees were an instrumental part in our ACL 2023 paper receiving some formal recognition.

I also thank Jacob Andreas and Mark Steedman for examining my thesis and the interesting discussion during my viva.

Many thanks to Miloš Stanojević for being an incredible mentor and collaborator during my internship at Google DeepMind. Thanks also to Chris Dyer for his comments and support for the internship project. I'm also grateful to Sohee Yang for exciting technical discussions, excellent advice, and friendship.

I'm also grateful to have worked with amazing collaborators like Guillem Ramírez Santos and John Gkountouras on other exciting research projects besides this thesis.

Thank you to Bailin Wang, Jonas Groschwitz and Henry Conklin for many fun and insightful discussions about grammars and later LLMs. Henry's views on inductive biases and language from the perspective of language evolution have enriched my own and had an important effect on the second part of this thesis. Many thanks to Verna Dankers for countless insightful discussions about interpretability papers and compositionality. Thanks also to Yuekun Yao and Hao Zheng for fruitful discussions about compositional generalization.

I'm grateful to Agostina Calabrese and Verna Dankers for their comments on drafts, the friendship, and all the fun times in the office and at conferences. Thanks also to everybody else at the CDT and everybody who has been part of Ivan's research group, in particular: Aida, Amr, Anil, Antonio, Dennis, Hosein, Jason, Masaru, Mattia, Max, Nikita, Parag, Pedro, Rochelle, Rohit, Tom, Victor, Xinnuo, Zeyu and Zheng.

Thank you to my parents, Brigitte and Manfred, as well as Arthur, Christin, Josefine and Mathilde for being such an amazing family. Finally, none of this would have been possible without my partner, Christine. Thank you!

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Matthias Moritz Lindemann*)

Table of Contents

Chapter 1

Introduction

INDUCTIVE biases play a key role in Natural Language Processing (NLP) and neighboring fields such as machine learning, linguistics and cognitive science. Inductive biases refer to the preferences, assumptions and knowledge that a learner, human or machine, brings to a task before having seen data for that task. Conceptually, inductive biases 'fill the gaps' in the training data with assumptions or prior knowledge about the nature of the task and thereby determine how the learner generalizes from the finite set of data it receives. Hence, inductive biases are very important when the training data provides only a weak signal. This is the case when there is only a small amount of training data or if the model is applied outside of the training distribution. Consequently, inductive biases are an important consideration for NLP and machine learning to design data-efficient systems that generalize robustly and can handle distribution shifts. For linguistics and cognitive science, the inductive biases of humans are of key interest in how children can learn language from a small amount of linguistic data (Chomsky, 1965), and how our inductive biases shape what language looks like (e.g. Reali and Griffiths, 2009).

Traditionally, grammar-based and statistical approaches dominated in structured NLP tasks such as machine translation (Koehn et al., 2003; Chiang, 2005, 2007) and semantic parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2006), i.e. the task of mapping sentences, questions or instructions to precise formal – potentially executable – representations of their meaning. These traditional approaches tend to have strong inductive biases and considerable built-in prior knowledge about the structural principles of language, such as recursion. However, grammar-based and statistical methods tend to be inflexible and brittle, e.g. in handling paraphrases or unseen words, which leads to suboptimal performance.

Over recent years, NLP has instead made tremendous progress with neural sequence-to-sequence models, e.g. based on Transformers (Vaswani et al., 2017; Lewis et al., 2020a; Raffel et al., 2020). These models outperform grammar-based and statistical methods by large margins on standard benchmarks. Remarkably, this success has been achieved with *general-purpose* architectures across scenarios ranging from low-level tasks such as grapheme-to-phoneme conversion (Wu et al., 2021a) to machine translation (Vaswani et al., 2017) and semantic parsing (Dong and Lapata, 2016; Jia and Liang, 2016; van Noord and Bos, 2017; Wang et al., 2020). Notably, these neural architectures have relatively weak inductive biases for the structural principles of language.

Consequently, despite their high performance on in-distribution test sets, they struggle with *structural generalization*, i.e. making generalizations using key structural principles of language to extrapolate beyond the training distribution (Finegan-Dollak et al., 2018; Lake and Baroni, 2018; Keysers et al., 2020; Kim and Linzen, 2020; Li et al., 2023a). This thesis takes inspiration from grammar-based approaches and aims to improve the ability for structural generalization of neural sequence-to-sequence models by enhancing their inductive biases for important structural principles of language.

Two major structural principles of language that we consider in this thesis are compositionality and recursion. Compositionality systematically relates sentences to their meaning and thereby enables us to understand sentences we have never encountered before. Hence, it is also crucial for generalization in all NLP tasks that relate to sentence-level meaning. In a common formulation, the principle of compositionality states that 'the meaning of a compound expression is a function of the meanings of its parts and of the way they are syntactically combined' (Partee, 1984). For example, the meaning of 'Emma doubted that Paul slept' is determined by the meaning of 'Emma', 'doubted' and the meaning of 'Paul slept' as well as how these phrases syntactically combine. Recursion is another important structural property of language. It enables certain phrase types to be embedded into each other an arbitrary number of times, such as in 'Mary remembered that John said that Emma doubted that Paul slept'. A learner equipped with knowledge of compositionality and recursion can understand this example sentence even if it has previously only encountered shallower forms of recursion with fewer embedded phrases. This is an instance of structural, or compositional, generalization that is very challenging for sequence-to-sequence semantic parsers with weak inductive biases (Kim and Linzen, 2020; Li et al., 2023a). Phrases appearing in contexts that they did not appear in during training form another class of structural generalizations that such models struggle with (Lake and Baroni, 2018; Keysers et al., 2020). For example, Kim and Linzen (2020) show that models struggle with correctly processing *subject* NPs that are modified by PPs when only *object* NPs were modified by PPs in the training data (see Table 1.1 for an example).

From the perspective of Machine Learning, such evaluations for linguistic principles test if a model uses *shortcuts*. Shortcuts are unintended solutions to learning problems that perform well in-distribution but perform poorly under *covariate shift*, i.e. when the distribution of test inputs differs substantially from the distribution of training inputs. In recent years, there has been considerable interest in shortcuts (Geirhos et al., 2020), covariate shift and out-of-distribution generalization more generally (Liu et al., 2023) as it affects machine learning in all domains. However, methods developed for handling covariate shift in machine learning typically are not compatible with the aims of this thesis because they only consider classification setups (Arjovsky et al., 2019; Krueger et al., 2021), rather than sequential outputs used more commonly in NLP.

In conjunction with flexible general-purpose neural architectures, advancements in NLP over the last couple of years have been driven mainly by *pretraining* increasingly larger models on massive text corpora (Devlin et al., 2019; Raffel et al., 2020; Brown et al., 2020; Touvron et al., 2023a,b). Large-scale pretraining equips models with linguistic knowledge (Tenney et al., 2019; Hewitt and Manning, 2019; Mueller et al., 2022) as well as world knowledge (Petroni et al., 2019). Since these models can be engaged with conversationally, this has made them practically useful to a broad audience

Training Data	**Generalization Instance**
Generalization to **unseen combinations of phrases**.	
How many rivers does Wyoming have? What states border Colorado ?	How many rivers do the states bordering Colorado have?
Generalization to **deeper recursion**.	
Ava believed **that** Emma said **that** a fish froze.	Ava said **that** Emma liked **that** Max believed **that** Noah found **that** Liam saw **that** the cat slept.
Only objects are modified by PPs → Subjects are modified by PPs	
Noah ate **the cake on the plate**.	**The cake on the table** burned.

Table 1.1: Examples for three different kinds of **structural generalization** challenges. The examples come from semantic parsing benchmarks and for each instance, a model has to predict a formal representation of its meaning. For conciseness, we only show the inputs and omit the formal meaning representations.

and radically changed the field of NLP over the course of my PhD. Large pretrained models have also shown clear and consistent improvements at structural generalization (Furrer et al., 2020; Herzig et al., 2021). However, to date, simply scaling up to larger models and more pretraining data has not addressed this issue completely, and structural generalization remains challenging (Yao and Koller, 2022; Li et al., 2023a). While scaling up tends to improve structural generalization, there are indications of diminishing returns (Qiu et al., 2022b) and even models like GPT-4 struggle with capturing concepts such as recursion (Dziri et al., 2023).

1.1 Aims of this Thesis

The goal of this thesis is to design, implement and evaluate methods for introducing structural inductive biases for language into neural network models. These inductive biases aim to enable the models to learn effectively from small amounts of task-specific data and to generalize systematically beyond the distribution that they were trained on. A particular focus of this thesis are inductive biases that enable *structural generalization*, i.e. the ability to generalize to longer inputs, deeper recursion and phrases in contexts in which they did not appear during training (see Table 1.1 for examples). More specifically, this thesis focuses on sequence-to-sequence models, which are applicable to many NLP tasks, and works towards the following two goals:

1. Integrating inductive biases into sequence-to-sequence models that make them better at structural generalization for semantic parsing.

2. A general and efficient framework for injecting task-appropriate structural inductive biases into standard Transformers that is applicable to semantic parsing and beyond.

In this thesis, semantic parsing plays a major role as an NLP task that can benefit from structural generalization. However, the methods developed here also aim to be

applicable more widely. For this reason, we focus on the sequential outputs, rather than tree-structured or graph-structured outputs (Herzig and Berant, 2021; Petit et al., 2023). This makes our methods easily applicable to other structure-sensitive tasks, such as grapheme-to-phoneme conversion or rephrasing sentences to emphasize a verb/adjective and phrase chunking.

While we focus on sequence-to-sequence models, the technical contributions of this thesis are also applicable beyond this context. For instance, Goal 2 and the methods we propose towards it could be compatible with decoder-only language models.

1.2 Thesis Overview

This thesis consists of two content parts, each with a different approach taken to integrate structural inductive biases into sequence-to-sequence models. Part I focuses on Goal 1 and investigates architectures that have inductive biases for structure by design. Inspired by grammar-based and statistical methods to NLP, we conceive of the output being the result of a sequence of edit operations applied to the input. We make these edit operations differentiable and turn them into neural network layers that can be composed and trained end-to-end. Part II addresses Goal 2 and proposes a framework for imparting structural inductive biases into standard Transformer models. Here, we operationalize an inductive bias as a family of symbolically defined functions and pretrain models with synthetic data for these functions. The two approaches taken in the two parts may seem fundamentally different at first glance. However, there is a common theme in that we take a discrete computational process and find a continuous, differentiable relaxation of it that provides guidance for a model. In the first part, this relaxation is hand-designed, while in the second part, the relaxation is learned from synthetic data and provides much softer guidance for the parameter estimation.

Part I - Architectures with Structural Inductive Biases

The first part, consisting of Chapters 3 and 4, presents sequence-to-sequence models whose structural inductive bias derives from their architectures. The main idea is to automatically identify reusable fragments of the input and explicitly model the correspondence between these input fragments and fragments of the output sequence. Conceptually, both chapters break down the sequence-to-sequence setup into two stages: (i) predicting the elements that will appear in the output irrespective of their order, and (ii) predicting their order. Crucially, both stages explicitly involve an alignment, or correspondence, between their respective inputs and outputs: in the first stage, these are input tokens and elements they 'contribute' to the output; in the second stage, these are sequences and their re-ordered variants. The explicit alignment is a key design decision that drives the inductive bias of the model. In addition, the alignments make the models more interpretable and easier to debug because errors can often be attributed clearly to a single stage. Central technical challenges that are addressed in both chapters deal with the fact that the alignments are not known at training time because annotation would be too costly. Hence, we use techniques for structured latent variable models to induce these as part of training.

Chapter 3 presents the first version in this thesis of a sequence-to-sequence model following this paradigm. Inspired by statistical machine translation (Brown et al., 1990), its main technical contribution is a differentiable fertility layer acting as the first stage in the two-stage process. For every token in the input, the fertility layer creates zero or more copies of the representation of the corresponding input token. To make the process of copying token representations differentiable, we assign probability distributions to the number of times any token is copied and compute an expected alignment by marginalizing over all possible options to obtain a specific output with a dynamic program. This can be seen as an instance of Structured Attention (Kim et al., 2017). This new layer is composed with an existing differentiable phrase reordering layer from Wang et al. (2021a). Consequently, the model is end-to-end differentiable. In contrast to other sequence-to-sequence models, the length of the output is *not* determined by a dedicated end-of-sequence token but is simply the total number of token representations that are copied. As such, it also captures that longer inputs tend to require longer outputs.

We evaluate this approach on three semantic parsing datasets (Geoquery, ATIS, Okapi) and demonstrate its efficacy at structural generalization. In addition, we also find it to perform well in a low-resource scenario on several syntactic transformation tasks that require systematically rephrasing a sentence, e.g. from active to passive voice.

Chapter 4 presents a sequence-to-sequence model based on the same intuition that addresses some of the limitations of the approach in Chapter 3. In particular, the phrase-reordering model used in Chapter 3 makes hard assumptions which kind of phrase reorderings are possible (Wang et al., 2021a), and these assumptions are too restrictive for some datasets or semantic formalisms. In addition, the high computational and memory cost of this reordering approach limits its applicability.

Chapter 4 introduces a flexible method to predict and parameterize permutations that does not place any hard restrictions on possible permutations that can be predicted. We do so by giving a score to every pair of tokens for how much they prefer to be adjacent in the output. Predicting a permutation then amounts to finding the permutation that maximizes the overall score. This permutation approach gives rise to two technical challenges: (i) this combinatorial optimization problem resembles the Travelling Salesman Problem and is NP-hard, (ii) on its own, this way of predicting permutations is unsuitable as a neural network component because its derivatives are zero almost anywhere, which would prevent gradient-based training. To overcome these challenges, we construct a continuous relaxation of this optimization problem and present an algorithm to approximately solve it based on Bregman's method (Bregman, 1967). This algorithm is a generalization of Sinkhorn's algorithm (Sinkhorn, 1964) and makes good use of the parallel processing power of GPUs. Despite being an approximation of an NP-hard problem, the method scales better empirically than the phrase reordering approach, in particular in terms of memory requirements.

We evaluate on several semantic parsing datasets and show that this approach performs well in terms of structural generalization. We show for the first time that a model without an inductive bias provided by trees can achieve high accuracy on generalization to deeper recursion.

Part II - Injecting Structural Inductive Biases by Simulation

In the second part of this thesis, we present a framework that can be used for introducing and modifying inductive biases of models by means of a particular pretraining setup. In contrast to the first part, we do not modify the architecture of the model. Instead, we use standard encoder-decoder Transformers and simply continue their pretraining with a structure-aware setup. The framework is based on the key observation that it is often possible to operationalize an inductive bias as a *family* of symbolically defined functions, and it tends to be cheap to automatically generate members of this family. Provided with a description of any function from that family, the pretraining objective is to match the input/output behavior of that function. Since pretraining covers the entire family, the model has inductive biases that are well-suited for a range of downstream tasks with different properties – as long as they are sufficiently similar to the family of functions considered during pretraining.

This approach has several practical advantages in comparison to designing new model architectures. In particular, it is easier to adjust the inductive biases of a model when demands change, i.e. by additional pretraining, without having to modify the model architecture and re-train from scratch. Hence, it also requires less expertise than the design of model architectures. Using existing Transformer architectures also leverages the infrastructure around the Transformer, such as the widely used HuggingFace library and model hub, making it easy for others to download the weights and use standard libraries to fine-tune these models.

Chapter 5 focuses on the inductive biases of Finite State Transducers (FSTs), which are useful for tasks such as grapheme-to-phoneme conversion (mapping orthographic representations of words to how they are pronounced) and simple text editing (`Jane Doe` $\mapsto$ `Doe, J.`). We pretrain a Transformer to *simulate* the behavior of FSTs by giving it a description of a randomly generated FST and an input string, and train it to predict the corresponding output of the given FST.

We show that a model pretrained with this procedure (called SIP, for simulation-induced prior) has an inductive bias that improves systematic generalization and few-shot learning for 'FST-like' downstream tasks. SIP improves systematic generalization on FST tasks similar to those seen during pretraining, but also on ones that are structurally more complex. The same pretrained model also transfers well to natural data and achieves strong results on few-shot learning of text editing and grapheme-to-phoneme conversion.

Our probing experiments give insights into how the inductive bias is injected: SIP not only leads to the imitation of the input/output behavior of FSTs, but encourages dynamics to emerge that simulate transitions between FST states in the hidden representations – without the model being supervised explicitly to do so. Fine-tuning can then leverage these dynamics, providing the inductive bias, and learn representations that resemble those of ground truth FSTs.

Finally, we relate this method to the fertility and permutation models introduced in Part I by viewing them all as continuous relaxations of discrete functions. While the fertility and permutation models are manually designed, SIP takes a Transformer and *learns* an approximation of the discrete functions represented by FSTs.

Chapter 6 applies this methodology to semantic parsing and syntax-sensitive tasks. We operationalize the inductive bias of interest as transformations of syntax trees. Traditionally, NLP has heavily relied on syntactic theories and has phrased many tasks as transformations of syntax trees, ranging from conversion of a sentence from active to passive voice (Oliva, 1988) to constructing a semantic representation for a sentence relying on the principle of compositionality (Montague, 1970). Transformations of syntax trees can address a task in a very generalizable way by using the right abstractions. For example, when constructing the semantic representation of an NP, the same transformations can be used for NPs whether they serve as direct objects or as indirect objects.

In this chapter, we start from a pretrained T5 model (Raffel et al., 2020) and further pretrain it on a dataset of automatically generated syntactic transformations of dependency trees. Given a description of the transformation and an input sentence, the model is pretrained to predict the output of the transformation without access to the underlying dependency tree. This procedure encourages the model to strengthen its syntactic representations and acquire reusable dynamics of syntactic transformations that can be leveraged for downstream tasks. Our experiments confirm that this improves structural generalization for semantic parsing and also helps with few-shot learning of syntactic tasks such as chunking. We then analyze how the intermediate pretraining affects the model internals and find that it leads to attention heads keeping track of which syntactic transformation needs to be applied to which token. We also find that the model can leverage these attention heads on downstream tasks after fine-tuning.

1.3 Published Work

The following chapters are based on already published work:

Chapter 3 Matthias Lindemann, Alexander Koller, and Ivan Titov. 2023a. Compositional Generalisation with Structured Reordering and Fertility Layers. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*.

Chapter 4 Matthias Lindemann, Alexander Koller, and Ivan Titov. 2023b. Compositional Generalization without Trees using Multiset Tagging and Latent Permutations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Chapter 5 Matthias Lindemann, Alexander Koller, and Ivan Titov. 2024a. SIP: Injecting a Structural Inductive Bias into a Seq2Seq Model by Simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Chapter 6 Matthias Lindemann, Alexander Koller, and Ivan Titov. 2024b. Strengthening Structural Inductive Biases by Pre-training to Perform Syntactic Transformations. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Chapter 2

Background

2.1 Sequence-to-Sequence Models

Sequence-to-sequence models are an important class of neural network models that have been applied to many NLP problems. As the name suggests, a sequence-to-sequence model is a model that takes a sequence of symbols as input (e.g. a sentence in one language) and produces a sequence of symbols as output, e.g. a sentence in a different language. In this section, we assume familiarity with the Transformer architecture (Vaswani et al., 2017) as used for language modeling, e.g. with GPT models (Brown et al., 2020). We provide a high-level review of sequence-to-sequence models and the encoder-decoder architecture (Sutskever et al., 2014; Bahdanau et al., 2014; Vaswani et al., 2017), highlight differences to autoregressive language models, and briefly discuss common challenges of encoder-decoder models as well as how they are pretrained. Throughout the thesis, we will use the notation below in Table 2.1:

i, j, k, l, m, n	Integers
$\mathcal{D}, \mathcal{A}$	Sets
x, y	Sequences, e.g. $x = x_1, \ldots, x_n$
$\mathbf{h}, \mathbf{x}$	Vectors and sequences of vectors, e.g. $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_n$
$[\mathbf{h}; \mathbf{x}]$	Vector concatenation
$\mathbf{A}, \mathbf{B}$	Matrices and tensors
$\mathbf{x}^*, \mathbf{U}^*$	Vector/matrix that is solution to an optimization problem
$\mathbf{y}^*(l)$	Solution to optimization problem as a function of a parameter l
$\mathbb{E}_{P(z\|x)} f(z)$	Expected value, $\mathbb{E}_{P(z\|x)} f(z) = \sum_z P(z\|x) f(z)$

Table 2.1: Notation used throughout the thesis.

Encoder-decoder models are neural networks consisting of an encoder that reads an input sequence x and a decoder that produces the output y. In probabilistic terms, an encoder-decoder model is a distribution $P_\theta(y|x)$ over output sequences y that conditions on the input sequence x. This distribution is parameterized by the neural network parameters θ, partitioned into the parameters ϕ of the encoder and separate parameters ψ for the decoder.

The encoder reads a sequence of input tokens $x_1, \ldots, x_n$ from an input vocabulary $\mathcal{V}$, embeds each token individually into a d-dimensional vector, and then encodes them

into a sequence of continuous hidden states $\mathbf{h}$:

$$\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^d = \text{EMBED}(x) = \text{EMBED}(x_1), \ldots, \text{EMBED}(x_k)$$
$$\mathbf{h} = \mathbf{h}_1, \ldots, \mathbf{h}_k \in \mathbb{R}^d = \text{ENCODER}_\phi(\mathbf{x})$$

Hence, ENCODER_ϕ is a function that reads a list of d-dimensional vectors and returns a list of d-dimensional vectors. Typically, ENCODER_ϕ is implemented as a Transformer encoder or an LSTM (Hochreiter and Schmidhuber, 1997), and there are usually as many hidden states as there are tokens, i.e. $n = k$. Since the encoder conditions on the entire input, it typically encodes the input *bidirectionally*, i.e. each hidden state $\mathbf{h}_i$ is a function of *all* tokens $x_1, \ldots, x_n$. This enables rich, highly contextualized representations and is in contrast to a Transformer decoder used in language models that uses a causal mask to prevent peeking at future tokens.

Once the hidden states have been produced by the encoder, they are passed on to the decoder, which defines an autoregressive distribution over output tokens $y_1, \ldots, y_m$ from an output vocabulary $\mathcal{V}'$ via the chain rule of probability:

$$P_\theta(y|x) = P_\psi(\text{EOS}|y, \text{ENCODER}_\phi(\mathbf{x})) \prod_{i=1}^{m} P_\psi(y_i|y_{<i}, \text{ENCODER}_\phi(\mathbf{x}))$$

Here, EOS is a token that marks the end of the output sequence. The decoder predicts the next output token given the embeddings $\text{EMBED}(y_{<i})$ of the partial output $y_{<i} = y_1, \ldots, y_{i-1}$ and additionally conditions on the encoder hidden states:

$$\begin{aligned} P_\psi(y_i|y_{<i}, \text{ENCODER}_\phi(\mathbf{x})) &= P_\psi(y_i|y_{<i}, \mathbf{h}) \\ &= \text{softmax}(\mathbf{W}_{\text{output}}\, \text{DECODER}_\psi(\text{EMBED}(y_{<i}), \mathbf{h})) \end{aligned}$$

If the input and output vocabulary are identical ($\mathcal{V} = \mathcal{V}'$), then the output layer $\mathbf{W}_{\text{output}}$ typically shares parameters with the embeddings (EMBED).

Decoders use *cross-attention* to condition on the hidden states $\mathbf{h}$ produced by the encoder. In cross attention, query vectors are derived from decoder representations, but key and value vectors are computed from encoder hidden states $\mathbf{h}$. Conceptually, this enables the decoder to use the query vectors to 'search' for specific relevant information in the encoder hidden states and then incorporate it. If a position needs to attend to multiple places at the same time, the decoder can spread the attention weight among those places. Alternatively, in models with multi-head attention (e.g. Transformers), each position can also attend to multiple places without having to spread the attention weights, making the cross attention more flexible and expressive.

In addition to the more high-level intuition of moving relevant information from the encoder to the decoder, cross-attention can be viewed as establishing, or predicting, correspondences between input tokens and output tokens as they are generated. For example, in the context of machine translation, Bahdanau et al. (2014) find that target language phrases in the output tend to attend to the corresponding phrases in the source language. While cross-attention can highlight correspondences, it does not have a direct probabilistic or linguistic interpretation. In particular, it is problematic to view attention maps in general as a form of alignments in the sense of traditional alignment methods (Brown et al., 1990; Och and Ney, 2003). Traditional alignments are posterior

distributions that condition on the generated tokens. Cross-attention is not a posterior distribution because it takes place *before* the relevant tokens are generated. The interpretation and the role of cross-attention within the model become more varied and complicated with multi-headed attention. For instance, some cross-attention heads can focus on frequent but uninformative tokens (Ferrando et al., 2022), e.g. punctuation or an end-of-sequence token, as a 'no-op' (Clark et al., 2019) when they do not find the information that they are looking for. In addition, certain cross-attention heads seem to have overall very little importance. For example, Michel et al. (2019) show that 30% of cross attention heads in a particular machine translation model can be pruned without materially affecting the performance. Finally, models can be trained to adapt their attention values such that the true correspondences cannot be read off anymore while performing nearly identically (Pruthi et al., 2020).

Encoder-decoder architectures such as Transformers are highly flexible models. In addition to processing natural language, they have also been very successful in other areas, including computer vision (e.g. Liu et al., 2021b). However, this flexibility of Transformers and related encoder-decoder architectures means that they have relatively weak inductive biases, and it also comes at the price of control. For instance, it is hard to enforce constraints that an encoder-decoder model has to respect certain correspondences between input and output. As a result, generated outputs can be detached from the input, i.e. ignore important parts of the input or contain parts that may be likely to occur on their own but do not correspond to anything in the input. This includes **hallucinations** where the model generates a plausible sounding translation (Lee et al., 2018), summary (Cao et al., 2018), or a textual description of an image (Li et al., 2023c) whose content is not supported by the input.

Encoder-Decoder Pretraining Pretraining has been an absolutely crucial driver behind recent progress in NLP. In contrast to language models that are pretrained to predict the next word, encoder-decoder models condition on an input string bidirectionally and produce an output string. Hence, next-word prediction is an inefficient way of pretraining for such models. Instead, encoder-decoder models are pretrained with denoising objective, similar to masked language modeling employed by BERT (Devlin et al., 2019). Raffel et al. (2020) describe the following denoising procedure for the T5 model where spans of text are replaced by special masking tokens and the model is tasked with reconstructing all the corrupted spans:

Original Text	Thank you for inviting me to your party last week
Corrupted Input	Thank you [X] me to your party [Y] week
Desired Output	[X] for inviting [Y] last [Z]

Table 2.2: Example of T5 pretraining objective. [Z] marks the end of the desired output.

Important pretrained encoder-decoder model for English are BART (Lewis et al., 2020b), T5 (Raffel et al., 2020) and T0 (Sanh et al., 2022). There are also multilingual variants of these models, namely mBART (Liu et al., 2020), mT5 (Xue et al., 2021) and ByT5 (Xue et al., 2022) which uses bytes as tokens.

Wang et al. (2022) compare architectures and pretraining objectives for the purpose of zero-shot generalization. They find that decoder-only language models perform best immediately after pretraining. However, when models can be fine-tuned on instruction tuning data, encoder-decoder models pretrained with span denoising perform better.

2.2 Semantic Parsing

Semantic parsing is the NLP task of mapping a sentence to a formal meaning representation (also called logical form) that precisely captures important aspects of its meaning. Semantic parsing falls into two categories: *executable* and *broad-coverage* semantic parsing.

Executable semantic parsing aims to parse sentences into meaning representations that can be executed like a piece of code. A typical use case is translating questions about a database ('Which flights go from London to Bangkok on August 14th?') into a formal language to query databases (e.g. SQL or domain-specific query languages). The queries can then be executed on the database, and provide the answer that can be then passed back to the user.

As a result, executable semantic parsing has many practical applications, such as answering users' questions about databases or interacting with their phone, e.g. to manage a calendar. Executable semantic parsing has similarities with code generation (see Zan et al. (2023); Jiang et al. (2024) for recent surveys), and code generation methods can be used for semantic parsing. In both cases, a user expresses an intention in natural language, and a model is tasked with producing an appropriate executable response. While code generation typically targets helping programming novices or assisting experts with simpler algorithmic tasks, semantic parsing usually aims to enable users with no programming skills at all to access information, interact with their phone or with web services.

Broad-coverage semantic parsing aims to map sentences from *any* domain into a formal representation that captures important semantic aspects of the sentence. For instance, these representations may be used to formally represent who did what to whom or capture conditions under which a sentence is true. Broad-coverage meaning representations are of interest in formal semantics (Montague, 1970; Kamp and Reyle, 2013) but have also been used in NLP applications in the past. For example, Reddy et al. (2016) use a broad-coverage semantic representation as a backbone for executable semantic parsing, and meaning representations in the AMR formalism (Abstract Meaning Representation, Banarescu et al., 2013) have been used as inputs to information extraction (Li et al., 2015; Hsu et al., 2023) and summarization (Hardy and Vlachos, 2018) systems. We refer to Wein and Opitz (2024) for a recent survey of applications of AMR. However, with the success of large pretrained models, broad-coverage meaning representations have become less popular for practical NLP applications in recent years.

In this thesis, we consider both broad-coverage and executable semantic parsing. We provide examples with different meaning representation formalisms used throughout the thesis in Table 2.3.

Problem Formulation

From the perspective of machine learning, we view semantic parsing as a supervised sequence-to-sequence problem: we assume access to an independent and identically distributed (iid) training set $\mathcal{D}_{\text{train}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ containing pairs (x, y) of a sentence x, e.g. a question or an instruction, and a corresponding meaning representa-

Dataset		**Example**
Geoquery	Sentence	what state bordering Nevada has the largest population?
	FunQL	`answer( largest_one( population_1( state( next_to_2( stateid( 'nevada' ) ) ) ) ) )`
ATIS	Sentence	what's the cheapest of the dallas to atlanta flights after 2 pm?
	FunQL	`answer( argmin_fare( _flight( intersection( _>_departure_time_2( time( 1400 ) ), _from_2( city_name(dallas) ), _to_2( city_name(atlanta) ) ) ) ) )`
Okapi	Sentence	What powerpoint files do I have in my drive that were created by Marin three hours ago and were last updated by Clarita?
	Okapi-API	`GET drive.root.children FILTER file.createdBy = Marin FILTER file.createdDateTime = three hours ago FILTER file.lastModifiedBy = Clarita FILTER file.fileType = ppt`
COGS/ SLOG	Sentence	Lincoln painted the dog in a crate.
	LF	`*dog(`x_3`); paint.agent(`x_1`, Lincoln)` $\wedge$ `paint.theme(`x_1`,` x_3`)` $\wedge$ `dog.nmod.in(`x_3`,` x_6`)` $\wedge$ `crate(`x_6`)`
	Var-free LF	`paint( agent= Lincoln, theme= *dog( nmod.in= crate ) )`

Table 2.3: Instances with sentences and meaning representations from semantic parsing corpora used in this thesis. Geoquery (Zelle and Mooney, 1996), ATIS (Dahl et al., 1994) and Okapi (Hosseini et al., 2021) are instances of executable semantic parsing, whereas COGS (Kim and Linzen, 2020) and SLOG (Li et al., 2023a) target broad-coverage semantic parsing. COGS and SLOG both come in two different kinds of logical forms (LFs), a variable-free one and one with variables based on predicate logic in a Neo-Davidsonian style (Davidson, 1967; Parsons, 1990).

tion y. We assume that each sentence can be translated into its logical form without requiring conversational context and that all executable logical forms refer to the same database or environment, i.e. that there is no need to condition on a database schema. We may also assume access to a context-free grammar G that defines which logical forms are syntactically well-formed.

While a training dataset for semantic parsing consists of pairs of sentences and logical forms, in general, there is a *many-to-many* relationship between sentences and their meaning representations. In particular, there are paraphrases with diverging syntactic structure that nevertheless map to the same meaning representation. For example, in Table 2.3, the following sentence would have the same FunQL representation as the example shown for ATIS: 'I would like to fly as cheaply as possible to Atlanta from Dallas after 14:00'. In contrast to the sentence in Table 2.3, it is not a question and it expresses the intention of flying with a verb phrase rather than a noun phrase.

In the other direction, there are also usually many equivalent meaning representations for the same sentence. For example, the ordering of conjuncts in the ATIS example and in the variable-based logical form in Table 2.3 has no impact on the execution of the meaning representation or the truth conditions of the logical form. An-

other example of equivalence is taking the maximum of a list. Query languages such as SQL have a predicate for taking the maximum, but one can equivalently also sort the list in descending order and return the first element. Inconsistent annotation across the dataset, such as unsystematic ordering of conjuncts or alternating between ways of taking the maximum, can negatively affect the accuracy of a semantic parser (e.g. Guo et al., 2020).

Throughout the thesis, we assume that meaning representations are represented as sequences. Behind the sequential representation, meaning representations are often trees or directed graphs. This is often indicated with parentheses to encode mother-daughter relationships in trees (e.g. in FunQL in Table 2.3). Graphs are often linearized into a sequential representation with variables. For example, the COGS LF in Table 2.3 uses the variable x_3 in three places, `*dog(`x_3`)`, `paint.theme(`x_1`,` x_3`)` and in `dog.nmod.in(`x_3`,` x_6`)`. We can interpret this as a linearized representation of a graph with a node x_3 that has label `*dog` and which connects to the nodes x_1 and x_6 via two edges labeled `paint.theme` and `dog.nmod.in`, respectively. Graphs are the most general way and sometimes the most natural way to think of meaning representations. However, treating meaning representations as sequences is in practice often much more convenient because we can use more general-purpose sequence-to-sequence models. It also does not *require* having access to the grammar G and reduces the amount of dataset-specific pre-processing needed, e.g. to parse sequential representations into graphs. For sequence-based approaches, variables can pose modeling challenges because variables are arbitrary, i.e. a systematic renaming does not change the semantics of the logical form. In addition, the number of variables is in principle unbounded, and more variables might be needed for parsing sentences in the test set in comparison to sentences in the training set. For these reasons, variable-free representations are often preferred, although the variable-free representations used in practice tend to be less expressive, see e.g. Wu et al. (2023a).

Apart from variables, some other factors play into how well a semantic parser can predict meaning representations. Notably, this includes how consistent the correspondence between syntactic structure and logical form is. Guo et al. (2020) compare how the choice of meaning representation formalism affects the accuracy of semantic parsers. They compare four meaning representation formalisms (FunQL, λ-calculus, Prolog, SQL) and found FunQL to be performing best, followed by λ-calculus and SQL performing worst. Hence, in this thesis, we focus on the formalisms in Table 2.3 that tend to be more amenable to semantic parsing.[1]

From Grammar-Based to Sequence-to-Sequence Parsers

Grammar-based Semantic Parsers Traditionally, semantic parsing has been seen through the lens of linguistics, which places a strong emphasis on how the meaning of

[1]It may seem undesirable for a semantic parser to struggle with one meaning representation more than with another. However, as the following thought experiment shows, we can construct a nightmarishly complex encoding of a meaning representation, which shows that this is ultimately inevitable: Take one of the meaning representations in Table 2.3 and encrypt it with a strong encryption algorithm. The encryption is a bijection (if the key is known) and hence equivalent to the original meaning representation. However, by design, it will disrupt systematic correspondences between input and output and will make it virtually impossible to predict the correct (encrypted) meaning representation.

a sentence is derived along its syntactic structure and the notion that grammars provide rules of semantic interpretation (Montague, 1970). Hence, early semantic parsers were grammar-based. For example, the SHRDLU system (Winograd, 1971) contained a semantic parser using grammar rules to enable a user to interact with an agent in a blockworld environment using language. However, rule-based approaches turned out to be very difficult to scale to more complex environments and struggle with ambiguity and the flexibility of language in actual use.

With the rise of statistical methods in NLP, grammar rules were assigned probabilities or weights that were learned from data. Zettlemoyer and Collins (2005, 2007) proposed to use Combinatorial Categorial Grammar (CCG Steedman, 1996, 2000) for executable semantic parsing. By design, CCG's syntactic analyses can be mapped directly to terms of λ-calculus, making it highly suitable for semantic parsing. Zettlemoyer and Collins (2005, 2007) assigned weights to derivations with a log-linear model, and phrased semantic parsing as finding the highest scoring derivation for a sentence. The final logical form can be read off from the derivation. Wong and Mooney (2006) proposed learning a weighted synchronous context-free grammar (Lewis and Stearns, 1968; Chiang, 2007), which derives a string in English and its executable logical form in lockstep. That is, the grammar explicitly captures correspondences between parts of the input and parts of the output.

Challenges for Grammar-Based Semantic Parsers An important technical challenge for these grammar-based semantic parsers is that the parser estimates a distribution over *derivations* but only the result of the derivation, i.e. the input string and the logical form, is annotated. Since it would be too expensive to manually annotate the derivations, they are treated as latent variables that are induced with algorithms such as Expectation Maximization (Dempster et al., 1977). A crucial sub-problem of identifying the derivations is to find correspondences between individual words and parts of the logical form, and making them explicit with alignments. To that end, some approaches have used alignment models such as the IBM models (Brown et al., 1990) that originated in statistical machine translation. In addition, sometimes correspondences were directly provided in the form of hand-crafted (seed) lexicons. The methods we propose in Part I also make use of explicit correspondences and alignments, and we will encounter some related technical challenges with inducing alignments again.

Another challenge for traditional grammar-based parsers is the adaptation to new semantic formalisms. For example, a CCG derivation can be mapped easily to a term in λ-calculus but taking such a system and adjusting it to map derivations to database queries in domain-specific languages such as SPARQL or FunQL is a non-trivial problem that again often requires machine learning methods (Reddy et al., 2014).

Encoder-Decoder Parsers With the rise of Deep Learning in NLP, sequence-to-sequence-style encoder-decoder models have become extremely popular for machine translation (Bahdanau et al., 2014; Vaswani et al., 2017) and soon after for semantic parsing. Sequence-to-sequence models have tremendous advantages over earlier statistical methods. This includes transfer learning and much less manual involvement for researchers and practitioners, as there is no need for feature engineering or hand-crafted seed lexicons to kick-start EM-style algorithms. They are also much more

flexible, use simpler standard gradient-based learning algorithms, and can be easily applied to new semantic formalisms, often without any adaptation.

The first sequence-to-sequence models used for semantic parsing were trained from scratch. Because sequence-to-sequence architectures provide relatively weak inductive biases when randomly initialized, they have to learn a lot more about how language works from the semantic parsing training sets and are quite data hungry. This has prompted work on data augmentation with grammars (Jia and Liang, 2016) and architectures that take into account that logical forms are often trees (Dong and Lapata, 2016).

With large-scale pretraining, sequence-to-sequence models have become more sample efficient with respect to task-specific training data and perform very well on standard benchmarks (Rongali et al., 2020; Bevilacqua et al., 2021). This is achieved with only little customization for semantic parsing, such as constrained decoding to ensure well-formed logical forms (Scholak et al., 2021). However, as we will discuss in Section 2.3, standard sequence-to-sequence models are not as successful as one might expect for *systematic generalization*, which tests if models generalize according to linguistic principles that grammar-based semantic parsers have built-in by design.

Evaluation Metrics for Semantic Parsing

Accuracy is perhaps the most common evaluation metric for semantic parsing. Accuracy measures the proportion of instances in the test set for which the parser produces the correct output. There are several ways of measuring *correctness* that have been used in the literature. *Exact match* uses a string match between the gold meaning representation and the predicted meaning representation that considers a prediction correct if it uses the same tokens as the gold standard in the same order. However, this can be overly strict because it does not take into account that meaning representations might differ in their string representations but are semantically equivalent.

One way to overcome this for executable semantic parsing is to execute the query and test if it produces the desired output (Liang et al., 2011), known as *execution accuracy*. However, this can be overly lenient. For example, in a domain in which it is natural to ask 'how many …?', it could be the case that the answer to such questions is 4 in 90% of cases. Hence, testing if execution of a predicted meaning representation yields 4 as a result would provide little insight and can be overly optimistic about the true performance in general.

As a middle ground between these two extremes, one can test correctness by checking if the predicted meaning representation is semantically equivalent to the gold meaning representation. Depending on the meaning representation, checking equivalence in general can be very hard, so it is often approximated, focusing on common cases of equivalence, such as accounting for different orders of conjuncts (Dong and Lapata, 2016).

2.3 Systematic, Compositional and Structural Generalization

Despite its complexity, language is a system that follows fundamental principles, such as recursion and compositionality. Since recursion permits a phrase of a certain type to be embedded into a larger phrase of the same type, we can construct an arbitrary number of arbitrarily long sentences. In order to understand what each of these sentences means, we cannot possibly memorize their meaning, and a procedure or principle is required to understand new sentences we have not encountered before. This is addressed by the principle of compositionality, which says that the meaning of a complex expression is determined by the meaning of its parts and the way they are syntactically combined (e.g. Partee, 1984).[2] Compositionality thus plays an important role in linguistics (Montague, 1970) but also in cognitive science (Fodor, 1975; Piantadosi et al., 2016) and artificial intelligence (Lake et al., 2017).

Compositionality is also highly relevant for semantic parsing and a linguistically appropriate semantic parser should be able to *generalize compositionally*, i.e. it should use the principle of compositionality to generalize robustly to new sentences. More generally, NLP models should be capable of *systematic generalization* in line with linguistic principles, e.g. recursion, to be appropriate models of aspects of language. Traditional grammar-based systems, e.g. for semantic parsing (Section 2.2), have the notions of compositionality and recursion built-in, so they generalize in this way by design.

However, to the initial surprise of some, a range of evaluations have shown that many neural sequence-to-sequence models do not generalize in line with linguistic principles despite their success on standard benchmarks with standard evaluation methods (random splits). This has been shown with specific behavioral testing, particularly for semantic parsing (Finegan-Dollak et al., 2018; Lake and Baroni, 2018; Kim and Linzen, 2020; Li et al., 2023a) and machine translation (Li et al., 2021; Zheng and Lapata, 2023).

In this section, we discuss the evaluation methods that have been introduced for testing systematic generalization and organize them into different categories depending on what kind of generalization they measure. Finally, we discuss the implications of applying these evaluations to pretrained models.

Testing for Systematic Generalization

Standard evaluation methods randomly split a dataset into a training and test set. The model is trained on the training set and then evaluated on the test set. This ubiquitous setup is unsuitable for evaluating systematic generalization because it does not test if models use principles such as compositionality to make their predictions. Instead, it tests if a model can make the right predictions on examples that look fairly similar to the data it was trained on. This is because randomly splitting the data results in an

[2] As usual in language, there are exceptions. Idioms are prime examples, such as 'kicking the bucket', which has a compositional, literal meaning and a non-compositional, non-literal meaning that refers to dying. Compositionality is a foundational but arguably not the *only* factor at play.

in-distribution evaluation protocol where training and test data come from the *same* underlying distribution. In-distribution evaluations overly focus on frequent cases, which are close to the training data and hence relatively easy. Therefore, this evaluation protocol is not sensitive to the challenging generalizations at the tail of the distribution. In practical scenarios, it is rarely the case that a model is trained on the same distribution as it is later applied to.

Finegan-Dollak et al. (2018) present a cautionary tale how in-distribution evaluation can make a misleading impression about generalization: they show that random splits suggest that a naive slot-filling model performs well for semantic parsing. However, their slot-filling model is extremely limited in its generalization capabilities by design and cannot use the principle of compositionality: it categorizes the sentence into a fixed number of templatic 'intents', each with a fixed number of slots that need to be filled. For example, a question about a flight from A to B with airline C is seen as a one intent (Template 1) and a question about a flight from A to B at time D is seen as a completely different one (Template 2). Without Template 3 for questions about flights from A to B with airline C at time D, the slot-filling model can never answer such a question – in contrast to a model that uses the principle of compositionality.

To test if a model uses principles such as compositionality, the evaluation setup needs to be adjusted such that a compositional strategy can solve the task readily, but a non-compositional strategy (such as slot-filling) will perform poorly. Finegan-Dollak et al. (2018) suggest template splits, i.e. all instances of a template are either put into the training set or the test set. For example, questions relating to Templates 1 and 2 could go into the training set, whereas questions related to Template 3 would go into the test set. Hence, this evaluation setup rewards generalization to templates that were not encountered in the training data.

This is an instance of out-of-distribution generalization because the training data and the test data no longer come from the same distribution. Formally, the training data consists of inputs x and outputs y sampled from $P_{\text{train}}(x)P(y|x)$ and the test data is sampled from $P_{\text{test}}(x)P(y|x)$ with $P_{\text{train}}(x) \neq P_{\text{test}}(x)$ but $P(y|x)$, e.g. the 'oracle' semantic parser, is the same during training and testing. Evaluation setups like this test whether a model is robust to distribution shifts. The distribution shift we are using for evaluating systematic generalization is called a covariate shift and is studied broadly in Machine Learning (Liu et al., 2023) and NLP, e.g. from the perspective of domain adaptation (Ramponi and Plank, 2020). In contrast to the evaluation of systematic generalization, domain adaptation assumes access to unlabeled data from P_{test}. Out-of-distribution evaluation can also be viewed from the perspective of causal world models, and Richens and Everitt (2024) prove that models whose error is bounded for distribution shifts have an approximation of the true causal model of the task.

The initial observations that standard neural models struggle with systematic generalizations (Finegan-Dollak et al., 2018; Lake and Baroni, 2018) have since sparked an interest in improving systematic generalization and extending out-of-distribution benchmarks for systematic generalization (see Hupkes et al. (2023) for a taxonomy of generalization research). To establish terminology, we provide an overview of different kinds of systematic generalization and how they are evaluated with out-of-distribution test sets. We give a visual overview how these relate to each other in Fig. 2.1.

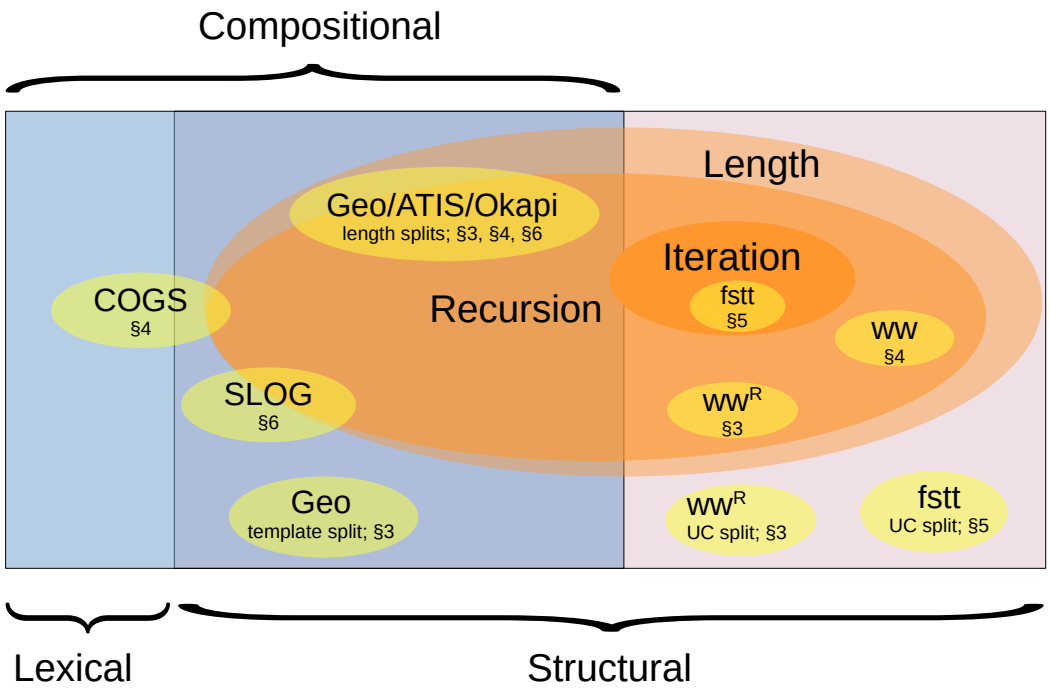


Figure 2.1: Venn diagrams of different kinds of *systematic generalization*: compositional, lexical and structural. Iteration generalization is a simple form of recursion generalization, which itself is a kind of structural generalization. We show datasets used in this thesis in yellow that test for the different kinds of systematic generalization.

Compositional Generalization refers to the ability to use the principle of compositionality to interpret the meaning of unseen expressions in a way that cannot be attributed to reliance on surface statistics. Hence, we view compositional generalization as a generalization at the syntax-semantics interface. As we detail next, out-of-distribution evaluation can be performed in a variety of ways, including the aforementioned template splits of Finegan-Dollak et al. (2018), and depends on the specific sub-category of compositional generalization.

Lexical Generalization is a kind of compositional generalization that refers to being able to correctly interpret a *word* in new contexts in which it has not been seen before, typically from a single occurrence of that word in training. Lexical generalization tests if a model can identify the syntactic category of a word (e.g. as a noun), form an association with its meaning, and interpret it in contexts in which other words of the same category have been seen during training. We provide two examples in Table 2.4.

Lake and Baroni (2018) proposed this out-of-distribution evaluation with their SCAN benchmark that requires a model to translate a command for a simple robot into a sequence of actions. Building on this, Kim and Linzen (2020) introduce COGS, a compositional generalization benchmark for broad-coverage semantic parsing with a strong focus on lexical generalization. COGS tests a model for 18 different forms of lexical generalization. In order to precisely control the distribution shift between training data and test data, the benchmark was synthetically generated by a grammar. While this often leads to semantically anomalous sentences, the sentences are gram-

Generalization	Training	Test
	SCAN (Lake and Baroni, 2018)	
'jump' in isolation → 'jump' in new context	walk ↦ WALK ⊕ walk twice ↦ WALK WALK ⊕ **jump** ↦ JUMP	**jump** twice ↦ JUMP JUMP
	COGS (Kim and Linzen, 2020)	
Subject (common noun) → Object	A **hedgehog** ate the cake. ↦ *cake(x_4); **hedgehog**(x_1) ∧ eat.agent(x_2, x_1) ∧ eat.theme(x_2, x_4)	The baby liked the **hedgehog**. ↦ *baby(x_1); ***hedgehog**(x_4); like.agent(x_2,x_1) ∧ like.theme(x_2,x_4)

Table 2.4: Examples for **lexical generalization**. The lexical item highlighted in bold appears only once in the training data, i.e. in one specific context (SCAN: in isolation, COGS: as a subject) and a model needs to generalize to usages in new contexts. ⊕ denotes that several structures are observed in the training data and they are all needed to generalize to the target structure.

matical and a compositional strategy should succeed on this benchmark.

Structural Generalization is the ability to use structural principles of language for generalization beyond the training distribution. In particular, this includes correct generalization to instances consisting of parts or having syntactic features that are known but whose *combination* has not been encountered during training. This is also known as *systematicity* (Hupkes et al., 2020) and we abbreviate such generalization scenarios with UC for unseen combinations.

Table 2.5 shows several examples. Making such generalizations requires the insight (i) what parts the input and output are composed of (ii) how parts in the input correspond to parts in the output and (iii) robustly composing the known parts in a novel way.

In contrast to compositional generalization, structural generalization is not constrained to the syntax-semantics interface and includes generalization that requires the application of structural principles on any symbol sequence. In Table 2.5 we show an example for mirroring a string ($w \mapsto ww^R$), where we test if the model decomposes the sub-string xyz into its constituent symbols and applies the same rule to them even if x, y and z only appear together in the training data.

Structural generalization tends to be harder than lexical generalization and the SLOG benchmark (Li et al., 2023a) extends COGS, focusing on structural generalization. SLOG is also synthetically generated to be able to test generalization very precisely. For evaluation on natural data, the aforementioned template splits (Finegan-Dollak et al., 2018) are often used but splits based on more sophisticated measures have been proposed. This includes splits based on Maximum Compound Divergence (MCD) (Keysers et al., 2020; Shaw et al., 2021). In order to test models on unseen combinations ('compounds') of predicates and entities, MCD partitions the data into training and test sets so as to maximize a *divergence* between the empirical training/test distributions of compounds. For example, if two predicates A and B appear rarely together in the training data, they are more likely to appear together in the test data. At

Generalization	Training	Test
Geoquery (template split) (Finegan-Dollak et al., 2018; Herzig and Berant, 2021)		
New Template	How many rivers does Wyoming have? ↦ answer(count(river(loc_2(stateid ('wyoming'))))) ⊕ What states border Colorado ? ↦ answer(state(next_to_2(stateid('colorado'))))	How many rivers run through the states bordering Colorado ? ↦ answer(count(river(traverse_2(state(next_to_2(stateid('colorado')))))))
New Template	What is the population density of the state with the smallest area? ↦ answer(density_1(smallest_one(area_1(state (all)))))	What is the state with the lowest population density ? ↦ answer(smallest_one(density_1(state(all))))
SLOG (Li et al., 2023a)		
PP in direct object NPs → PP in subject NPs → RC in indirect objects	Noah ate **the cake on the plate**. Noah saw **the cat that froze**.	**The cake on the table** burned. Max gave a fish to **a cat that ran**.
$w \mapsto ww^R$ (UC evaluation, Section 3.5)		
Unseen symbols after x; x is always followed by yz during training	i**xyz**bc ↦ i**xyz**bccb**zyx**i	b**x**c ↦ c**x**bb**x**c
Deletion of leading zeros (simplification of fstt evaluation, Section 5.3)		
Unseen combination of 2 with leading zeros	0012 ↦ 12 ⊕ 2201 ↦ 2201 ⊕ 1012 ↦ 1012	**002**1 ↦ 21

Table 2.5: Examples for **structural generalization**. We omit the meaning representations of SLOG for a compact presentation of the examples.

the same time, MCD splits ensure that the evaluation is fair and that all predicates or entities appear approximately equally often overall, both in the training data and the test data. MCD splits are implemented with a greedy procedure that adds one instance at a time to either the training or the test data.

Recursion Generalization is a specific form of structural generalization that requires use of the principle of recursion, and evaluates if a model 'understands' recursion. In particular, it refers to being able to generalize from examples with shallow recursion to examples with deeper recursion. We present examples in Table 2.6. A split is created by setting a maximum recursion depth, and everything below the cut-off is used as training data. Instances with depths slightly beyond the maximum training depth may be used as validation data. The recursion depth can be measured in the input and output simultaneously or primarily in the output, such as in the example for ATIS, where conjuncts are added by recursion according to the grammar of the logical form. For the ATIS example, deeper recursion in the logical form tends to correspond to deeper syntactic recursion in the input sentence but the levels of recursion in input and output do not correspond to each other as neatly as e.g. in COGS.

Generalization	Training	Test
COGS (Kim and Linzen, 2020) and SLOG (Li et al., 2023a)		
PP recursion max depth 4 → depth 5-12	Ava saw the ball **in** the bottle **on** the table.	Ava saw the cat **in** the box **on** the mat **on** the bed **on** the floor **in** the room.
Tail CP recursion max depth 4 → depth 5-12	Ava believed **that** Emma said **that** a fish froze.	Ava said **that** Emma liked **that** Max believed **that** Noah found **that** Liam saw **that** the cat slept.
Center embedding max depth 4 → depth 5-12	Eva saw the cat **that** the horse **that** the dog liked chased.	Ava held the dress **that** a store **that** a girl **that** a boy **that** a cat **that** a man drew saw loved knew sold.
ATIS (Dahl et al., 1994; Guo et al., 2020)		
3 conjuncts → 4 or more conjuncts	give me the flights for american airline from dallas to denver ↦ `answer( _flight( intersection ( airline_2(airline_code( aa)), _from_2( city_name (dallas)), _to_2(city_name(denver)) )))`	list flights from atlanta to boston leaving between 6pm and 10pm on august eighth ↦ `answer( _flight ( intersection ( _>_departure_time_2(time(1800)), _<_departure_time_2(time(2200)), _from_2(city_name(atlanta)), _to_2(city_name(boston)), _day_number_2(day_number(8)), _month_2(month(august)) )))`
$w \mapsto ww$ (Length split, Section 4.5)		
Max length 10 → Length 11 - 20	`icckh` ↦ `icckhicckh`	`bcgaieibikhgcaj` ↦ `bcgaieibikhgcajbcgaieibikhgcaj`
$w \mapsto ww^R$ (Length split, Section 3.5)		
Max length 10 → Length 11 - 20	`fafdk` ↦ `fafdkkdfaf`	`kfggjdhfeac` ↦ `kfggjdhfeaccaefhdjggfk`

Table 2.6: Examples for **recursion generalization**. We omit the meaning representations of COGS/SLOG for a compact presentation of the examples.

Iteration Generalization is a form of recursion generalization where the recursion takes a particularly simple form that can be implemented by Finite State Transducers (see Section 5.1 for a short overview). This *excludes* any form of center embedding, both on the source side and on the target side, such as matching parentheses of unlimited depth, which is common in tree-based semantic representation (see Table 2.3). We present an example in Table 2.7.

Generalization cases	Training	Generalization
Thousands separator (simplification of fstt evaluation, Section 5.3)		
3 insertions of separator → 4 or more insertions	`14812` ↦ `14,812` ⊕ `63721739260` ↦ `63,721,739,260`	`9263762670269316` ↦ `9,263,762,670,269,316`

Table 2.7: Example for **iteration generalization**.

Length Generalization refers to the ability to make correct predictions for input/output pairs (x, y) where x (or y) are longer than the inputs (or outputs) encountered during

training. This includes recursion generalization and iteration generalization; it also includes instances where input or output are longer than the training examples because they consist of a combination of long phrases that were not observed together during training, but do not involve recursion. Length generalization is easy to test for by simply partitioning examples into training and test sets based on their lengths (length splits; Lake and Baroni (2018)).

Evaluations of length generalization that do not evaluate recursion or iteration generalization are particularly of note for models whose parameters are tied to positions. In particular, this concerns Transformers encoders, which are equivariant to permutations and need explicit positional embeddings to distinguish words in different positions. Models with absolute position embeddings struggle with length generalization because the model has not been exposed to positions at the end of long inputs during training. Such findings have prompted research focusing on generalization and positional embeddings (Ruoss et al., 2023).

Systematic Generalization and Pretrained Models

The evaluation methods and splits we have presented in this section rely on a strict separation between training and test distribution, and test models on unfamiliar, out-of-distribution inputs of different kinds. Here, we discuss how this relates to using pretrained models that have been exposed to vast amounts of data and aim to clarify the role of out-of-distribution generalization for different kinds of pretrained models.

Pretraining with self-supervision Kim et al. (2022) have voiced concern that pretraining might circumvent the careful control between training and test distribution when testing for lexical generalization. For example, in the COGS benchmark, the word 'hedgehog' is only seen in a single context as a subject in the training set but pretrained models have been exposed to many sentences containing 'hedgehog' in a variety of contexts. This most likely includes the use as syntactic objects, which is the generalization that COGS tests for. Kim et al. (2022) argue that using pretrained models addresses a different research question, namely "whether models can adapt to a specific finetuning task under distribution shift of certain lexical items" but does not evaluate models for "[true] systematic generalization that does not depend on specific choice of lexical items".

The argument made by Kim et al. (2022) can, in principle, be extended to structural generalization. However, certain kinds of structural generalization likely have to be excluded from this argument because these phenomena are so rare that a model is unlikely to have been exposed to them during pretraining. The prime example for this is center embedding with recursion depth of 4 or more being (so far) unattested in natural language corpora (Karlsson, 2007). This also holds for other types of recursion, such as PP recursion for sufficient depths.

Pretrained models were likely exposed to sentences with syntactic phenomena that are supposed to be withheld from the training data of benchmarks such as COGS. However, for models pretrained on *text-only* datasets such as RoBERTa (Liu et al. (2019), used in Chapter 4), there is no supervision for corresponding outputs, e.g. logical forms. Instead, the pretraining relies on self-supervised objectives, such as reconstructing corrupted inputs or predicting the next words given a prefix. This provides

no direct evidence for the targeted generalizations and applies to all pretrained models used in this thesis since they were not pretrained on logical forms or code. However, pretraining may provide indirect evidence that the withheld phenomena should not be treated as special cases by the model. In comparison to lexical generalization, this indirect evidence seems to be weaker, which is reflected in pretrained models struggling more with structural generalization than with lexical generalization (Yao and Koller, 2022).

Multitask and Instruction-Tuned Models In addition to pretraining with simple self-supervised objectives, current LLMs are also instruction-tuned and undergo reinforcement learning to adjust responses to user preferences (Ouyang et al., 2022). The instruction tuning data can also contain common benchmark datasets (Touvron et al., 2023b). In all of these stages, models receive non-trivial supervision for outputs, notably including code. LLMs are effectively trained to be highly capable multi-task models that can be applied to new tasks. Hence, even when a specific task for which we want to evaluate systematic generalization was not observed during any training stage, the task might be within the *task distribution* that the model was trained on. This is a practically relevant setup in its own right (see Section 5.3.3 where we touch on a simple and controlled version of this). However, in the context of multi-task models that can follow instructions, we think that systematic generalization for tasks outside of the *task distribution* is the more challenging and interesting setup. So far, there have been limited evaluations that target systematic generalization that is specifically designed to be outside of the task distribution for LLMs. Designing such benchmarks is challenging because (i) the training data of LLMs is usually a well-guarded secret (although see OLMo Team et al. (2024)), and (ii) even if we had access to the data, it is not obvious how to determine if a specific task is within the task distribution. Consequently, this involves guesswork as to what a model might have seen during training. For instance, Dziri et al. (2023) show that GPT-3 and GPT-4 models struggle with systematic generalization for algorithmic problems (logic puzzles, dynamic programming, multiplication of large integers), which are conceivably outside the distribution of tasks the model was trained for.

As noted by Kim et al. (2022), the implications of using pretrained models depend on the perspective and research question. From the perspective of cognitive science, we contend that experiments with pretrained models are difficult to interpret because pretrained models have been exposed to much more linguistic data than a human native speaker.

Ultimately, what we care about in this thesis is the ability of a model to systematically generalize beyond the training distribution for a wide range of tasks that are of practical and theoretical interest. For a multi-task model, this includes systematic generalization for tasks outside the distribution over tasks it was trained on. Pretraining makes systematic generalization less challenging, but has no bearing on how useful it is to be able to make those generalizations.

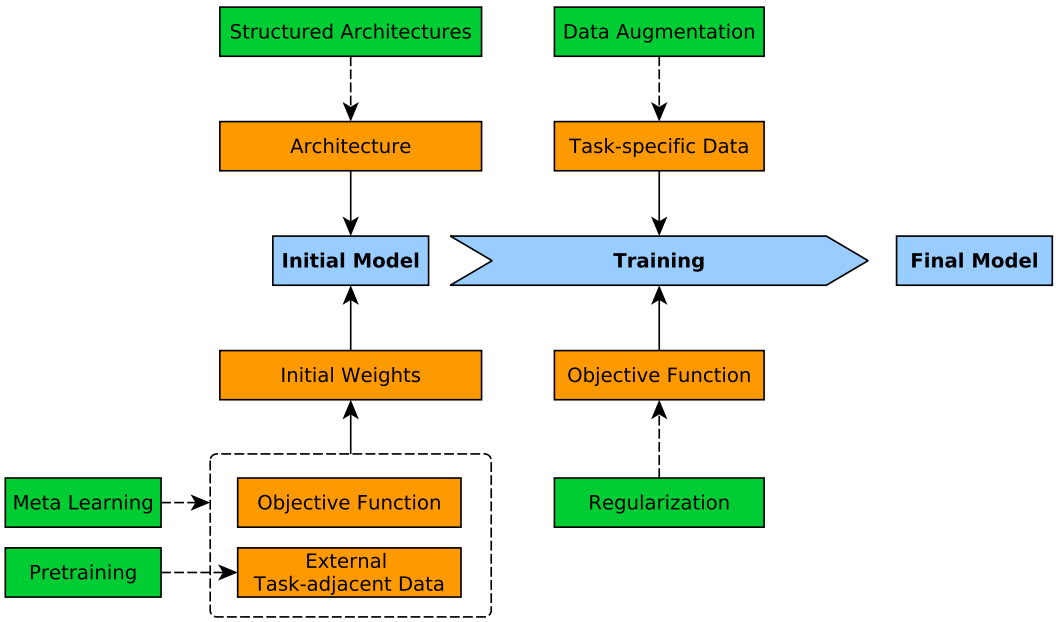


Figure 2.2: Sources of inductive bias that influence which solution a model converges to. Each aspect can be intervened on to influence the solution of the optimization procedure and how the model generalizes. In this section, we give an overview of related work and which aspects they target. If this document is viewed with a supported PDF reader, then hovering over the aspects depicted in green will show references for corresponding approaches.

2.4 Methods for Improving Systematic Generalization

In this section, we are going to give an overview of existing approaches for improving systematic generalization of sequence-to-sequence models with a focus on structural generalization and semantic parsing. As a reference point, we consider a setup in which a standard architecture, such as a Transformer, is randomly initialized and trained on the task-specific training data to maximize the log likelihood of the correct output. This reference point tends to perform poorly in terms of systematic generalization and produce outputs that are detached from the input. This is because there is nothing in that pipeline that provides a sufficient **inductive bias** for linguistic principles. In Fig. 2.2 we outline different aspects that influence which solution a model converges to, and give an overview in which aspects previous work has deviated from this point of reference to improve systematic generalization. In practice, many approaches combine interventions to multiple aspects, e.g. pretraining and data augmentation, to achieve the best performance. In this section, we go over each aspect and review relevant methods.

Structured Architectures encapsulate the required inductive bias for systematic generalization in the way they are wired or provide a decomposition of the task into simpler sub-tasks. Structured architectures tend to explicitly model correspondences between the input and output, and we will focus on these in this section. Such architectures

have also been proposed before the interest in systematic generalization with the objective of making models more sample-efficient by providing an appropriate inductive bias (e.g. Lyu and Titov, 2018; Groschwitz et al., 2018; Wu and Cotterell, 2019; Lyu et al., 2021a). Other changes to neural architectures that have been proposed to improve systematic generalization aim to disentangle representations (Zheng and Lapata, 2022) or share parameters across Transformer layers to be able to capture recursion up to a fixed depth (Csordás et al., 2022).

Akyürek and Andreas (2021) considerably improve lexical generalization by learning an external lexicon that captures systematic correspondences between an input token and output tokens. Similar to a copy mechanism (See et al., 2017), their sequence-to-sequence model can use attention to point to a word in the input and generate its lexical translation.

Kim (2021) proposes a sequence-to-sequence architecture with a hierarchical inductive bias. It uses a neural network to parametrize a quasi-synchronous context-free grammar (QCFG, Smith and Eisner (2006)). This approach first parses the input into a constituency tree and then generates a derivation of the output conditioned on the tree using alignments between constituents in the input and the output. Since neither the parse trees of the input and output nor the alignments are annotated, Kim (2021) treats these as latent variables and maximizes the marginal likelihood of the output string using a combination of exact marginalization and REINFORCE (Williams, 1992). The resulting model is very flexible but computationally demanding, making it challenging to scale to long sequences and larger datasets.

Zheng and Lapata (2021) learn a semantic tagger that tags tokens in the input with tokens of the output. This is combined with a standard sequence-to-sequence model that receives the original tokens augmented with the semantic tags as input. Since the semantic tags are not annotated, they induce the tags with variants of the EM algorithm.

Wang et al. (2021a) design a sequence-to-sequence approach that models the correspondence between input and output as a two-step process: it first reorders the input sequence and then translates the reordered input monotonically, i.e. from left-to-right. They propose a reordering model with a hierarchical inductive bias and train the model end-to-end since there are no annotations for the correspondences. Cazzaro et al. (2023) similarly decompose semantic parsing into a monotonic left-to-right translation followed by a reordering step to arrive at the final logical form. Both steps are implemented with standard pretrained sequence-to-sequence models and they assume alignments as given, so they can construct training data for the model used in each step.

The approaches by Wang et al. (2021a) and Cazzaro et al. (2023) are conceptually closely related to the methods we propose in Part I, and we discuss similarities and differences in more detail in Sections 3.6 and 4.7. The most important technical differences are as follows: the methods we introduce in Part I can be applied to a broader range of tasks than the method of Wang et al. (2021a) because our approach captures a strictly larger class of correspondences while being more efficient without compromising on accuracy. In contrast to Cazzaro et al. (2023), we do not assume alignments as given and induce them during training since high-quality alignments are often not available in practice.

Several methods for improving compositional generalization for semantic pars-

ing also exploit that meaning representations are often trees or graphs rather than sequences and directly predict trees or graphs. Hence, these methods do not fall into the category of sequence-to-sequence models. We nevertheless give a brief overview since these methods excel at structural generalization. All of these methods have in common that they align nodes, subgraphs or sub-trees to tokens in the sentence. Predicting edges between nodes can then be framed as prediction relations between tokens and is handled with models and algorithms inspired by syntactic parsing. The methods mainly differ in (i) which units are aligned to tokens, e.g. individual nodes (Herzig and Berant, 2021; Petit and Corro, 2023), sub-trees or subgraphs (Groschwitz et al., 2018; Lyu and Titov, 2018; Lyu et al., 2021a; Weißenhorn et al., 2022; Jambor and Bahdanau, 2022; Petit et al., 2023), and (ii) what kind of structure is predicted on the sentence, e.g. a tree resembling phrase structure (Herzig and Berant, 2021; Liu et al., 2021a), a dependency tree (Weißenhorn et al., 2022; Petit and Corro, 2023) or a dependency graph (Lyu and Titov, 2018; Lyu et al., 2021a; Jambor and Bahdanau, 2022; Petit and Corro, 2023). While some approaches directly predict the meaning representation in this way (Jambor and Bahdanau, 2022; Petit and Corro, 2023), others predict trees that can be seen as terms of an algebra (Groschwitz et al., 2018; Weißenhorn et al., 2022; Liu et al., 2021a; Herzig and Berant, 2021). These terms can then be deterministically evaluated bottom-up, resulting in the final meaning representation. This algebraic approach directly relies on the principle of compositionality, similarly to grammar-based methods: it first predicts the compositional structure and then uses the principle of compositionality to derive the meaning of a complex expression based on the meaning of its immediate sub-expressions.

Pretraining intervenes on the initial weights of the model and encodes domain knowledge from large amounts of task-adjacent external data in the parameters. Pretraining is used across all of NLP and has also led to improvements for systematic generalization. Furrer et al. (2020) benchmark the effectiveness of T5 pretraining and several architectures proposed for compositional generalization (in particular lexical generalization) and find T5 pretraining to perform more reliably. Herzig et al. (2021) demonstrate that pretraining is particularly helpful when combined with pre-processing logical forms into shorter intermediate representations that are more amenable to semantic parsing.

Large decoder-only LLM models such as from the GPT family (Brown et al., 2020) have also shown success for compositional generalization, in particular with prompting and in-context examples. Drozdov et al. (2022) report close to perfect accuracy on the COGS benchmark (Kim and Linzen, 2020) when prompting Codex, a version of InstructGPT (Ouyang et al., 2022) also trained on code. This performance cannot be attributed to the parameters of the model alone because considerable domain knowledge is added to the prompts. In particular, they instruct the model to syntactically parse the input sentence using a hand-crafted dataset-specific syntactic annotation with manually annotated in-context examples. Qiu et al. (2022b) compare differently-sized T5 (Raffel et al., 2020) and PaLM models (Chowdhery et al., 2023) and found that scaling up the size of the model improves compositional generalization to a limited degree for standard fine-tuning techniques. Although there is a positive trend for scaling model size when using in-context learning and prompt tuning with PaLM, these approaches perform worse in general than fine-tuning much smaller models T5 models. This casts

some doubts about how much naively scaling up pretrained models and datasets can further improve systematic generalization.

For executable semantic parsing, Levy et al. (2023) show that the selection of in-context examples can have a big impact on the ability to generalize compositionally and propose strategies for identifying relevant and diverse in-context examples.

The apparent success of closed-source models on new splits of established benchmarks, as evaluated by e.g. Levy et al. (2023), should be interpreted with caution. The training and test data of established benchmarks is often available in online repositories such as GitHub, which were almost certainly scraped for pretraining for many current LLMs. For example, at the time of writing, an exact string match search for a sentence from the Geoquery corpus found over 160 files on GitHub that contain the sentence with various types of annotations, including gold-standard logical forms. Hence, an LLM achieving high accuracy on such instances could be explained by memorization rather than generalization.

Meta-Learning views the task(s) we care about as a sample from a *distribution over tasks* that share important properties. Meta-learning assumes that we have access to more tasks from this distribution and aims to identify their shared properties to learn new tasks in a sample-efficient and effective way. Model-agnostic meta-learning (MAML) (Finn et al., 2017) is a popular way of meta-learning and intervenes on the initial weights of the model before it is trained on task-specific data. More specifically, it optimizes the initial weights such that gradient descent on a small amount of data from a randomly chosen task leads to good accuracy. This is achieved by back-propagating through gradient descent into the initial weights. Since learning well from small amounts of data is a consequence of inductive bias, MAML can be seen as optimizing for a specific inductive bias. McCoy et al. (2020) use MAML to inject an inductive bias for syllable structure based on Optimality Theory (Prince and Smolensky, 1993) into an LSTM-based model. They use MAML with synthetically generated datasets, each representing a different synthetic 'language' and show improvements on learning from small amounts of data and out-of-distribution generalization, including generalization to longer inputs. Conklin et al. (2021) apply MAML to improve compositional generalization for semantic parsing, with a focus on lexical generalization. Instead of using a distribution over genuinely different semantic parsing tasks, they simulate different tasks by sub-sampling the task-specific training data and optimize the model to generalize to simulated out-of-distribution tasks.

Another approach to meta-learning encodes a small 'support' dataset of input/output pairs for a particular task using a neural network (e.g. Santoro et al., 2016; Mishra et al., 2018). The network also receives an unlabeled input and is trained to predict the corresponding output in accordance with the support dataset it received. This approach has been called black-box meta learning (Finn, 2018) and – in the context of synthetic data – prior-fitted networks (Müller et al., 2022). Lake (2019) and Lake and Baroni (2023) demonstrate that this approach can be used to improve lexical generalization.

Meta-learning faces hard computational challenges. MAML needs to compute second-order derivatives to backpropagate through gradient descent, which requires large amounts of memory and can be sensitive to hyperparameters (Antoniou et al., 2019). However, less memory-intensive first-order approximations of MAML have

been proposed (Finn et al., 2017; Nichol et al., 2018). Black-box meta-learning needs to process and store activations for an entire dataset for each individual prediction, which is very compute and memory-intensive, making it already very difficult to scale to hundreds of training examples.

The methods we propose in Part II can be viewed as related to black-box meta-learning but our approach is vastly more efficient. This is because we exploit the fact that each task from the distribution is generated synthetically from a small program. Instead of encoding the examples generated by the program, we encode a compact *description* of the program.

Data Augmentation changes the task-specific training set. It 'augments' the training data with additional, generated training data that exhibit characteristics of the distribution we want the model to generalize to. There are roughly two approaches to generating more training data, those based on rules (Andreas, 2020; Qiu et al., 2022a; Yang et al., 2022; Akyurek and Andreas, 2023) and those based on neural network models (Wang et al., 2021b; Li et al., 2023d; Yao and Koller, 2024; Cazzaro et al., 2024).

Andreas (2020) presents GECA, a simple but 'good enough' rule-based method for data augmentation that is based on the idea of finding reusable, possibly discontinuous fragments that systematically link input and output. When two fragments x and y appear in similar environments, i.e. they are surrounded by the same words, GECA views these fragments as interchangeable. Hence, when fragment x appears in a new environment, a new instance can be synthesized by replacing x with y. Despite its simplifying assumptions about compositionality, it performs well in practice.

Qiu et al. (2022a) takes this further and learn a recursive grammar that simultaneously derives an input and an output sequence using a QCFG (Smith and Eisner, 2006). They follow the Minimum Description Length principle and greedily search for a grammar that both explains the data well and at the same time uses only a small number of highly reusable rules.

Wang et al. (2021b) propose a model-based approach to generate more training data. For semantic parsing, we often have access to a grammar of the logical forms that can ensure syntactic and basic semantic well-formedness. Wang et al. (2021b) propose to use such grammars to generate more logical forms. However, the generated logical forms are not paired with sentences. Hence, they train a model to translate logical forms to sentences on the original training set and then back-translate (Sennrich et al., 2016) the generated logical forms.

Data augmentation faces several challenges. In general, there is a trade-off between augmented data being sufficiently novel to be useful and being sufficiently conservative to avoid introducing annotation errors. Navigating this trade-off could be particularly challenging when the training data itself is noisy, in particular for rule-based systems where a small number of instances can lead to wide-ranging generalizations. Recursion also poses a challenge for data augmentation since any generated additional training data is finite and has a finite maximal recursion depth. To our knowledge, it has not been shown that augmentation enables substantial generalization beyond the recursion depth of the augmented data. This is in contrast to some approaches that target the architecture of the model, which can – in principle – capture recursion to arbitrary depth.

Regularization modifies the objective that a model is optimized for to influence *how* the model solves the task. For semantic parsing, Yin et al. (2021) induce alignments between spans in the sentence and spans in the logical form, and supervise the cross-attention of the model to respect these alignments. Yin et al. (2023) propose to regularize a model to represent the same token similarly across different syntactic contexts using contrastive learning. In addition, they encourage consistency of model predictions across different perturbations by dropout.

Abnar et al. (2020) explore 'transferring' inductive biases from one model to another using knowledge distillation, e.g. from a CNN to an MLP, and demonstrate that this can improve out-of-distribution generalization of the student model if the teacher model has the right inductive bias for the task.

Overall, regularization has been a comparatively less popular approach for improving systematic generalization. A potential reason for this could be a tendency of the optimization procedure to find a way to 'hack' the regularization loss, i.e. to reduce the loss without moving the strategy employed by the model in the direction that was intended (e.g. Skalse et al., 2022; Ferreira et al., 2025).

Part I

Architectures with Structural Inductive Biases

Chapter 3

Structured Reordering and Fertility

This chapter proposes a first approach towards Goal 1 outlined in the introduction and aims to improve structural generalization with a particular focus on semantic parsing. Since commonly used sequence-to-sequence models lack the inductive biases that enable structural generalization (see Section 2.3), we design a new sequence-to-sequence model with an architecture that provides an appropriate structural inductive bias. The main idea is to conceive of the output as being the result of edit operations applied to the input. This makes correspondences between input and output explicit and drives the inductive bias. We make these edit operations differentiable and turn them into flexible neural network layers that can be composed and trained end-to-end. This chapter is based on Lindemann et al. (2023a).

Consider the example in Fig. 3.1. Arguably, any model that produces the given semantic parse from the input in a generalizable way has to capture the correspondence between fragments of the input and fragments of the output, at least implicitly. This is challenging because of structural mismatches between input and output. For example, the fragment contributed by "flights" is discontinuous and intertwined with the rest of the semantic parse.

Grammar-based models such as those based on synchronous context-free grammars (SCFGs) (Lewis and Stearns, 1968; Chiang, 2007) *explicitly* capture the compositional process behind the data and therefore perform very well in compositional generalization setups. They can also model common structural mismatches like the one shown in the example. However, in contrast to sequence-to-sequence models, grammar-based models are rigid and brittle and thus do not scale well to new tasks.

Taking inspiration from grammar-based models, we also *explicitly* capture correspondences between fragments of the input and fragments of the output and present an end-to-end differentiable neural model that is both flexible and generalizes well compositionally. Our model consists of two structural layers: a phrase reordering layer originally introduced by Wang et al. (2021a) and a fertility layer, new in this work, which creates zero or more *copies* of the representation of any input token. Differentiability is ensured by using the expected value of each step. We show how to compute the expectation of the fertility step with dynamic programming and compose these layers to achieve strong generalization.

We use a simple non-autoregressive decoder and translate each token after the fertility and reordering layers into one output token conditioned only on the input sen-

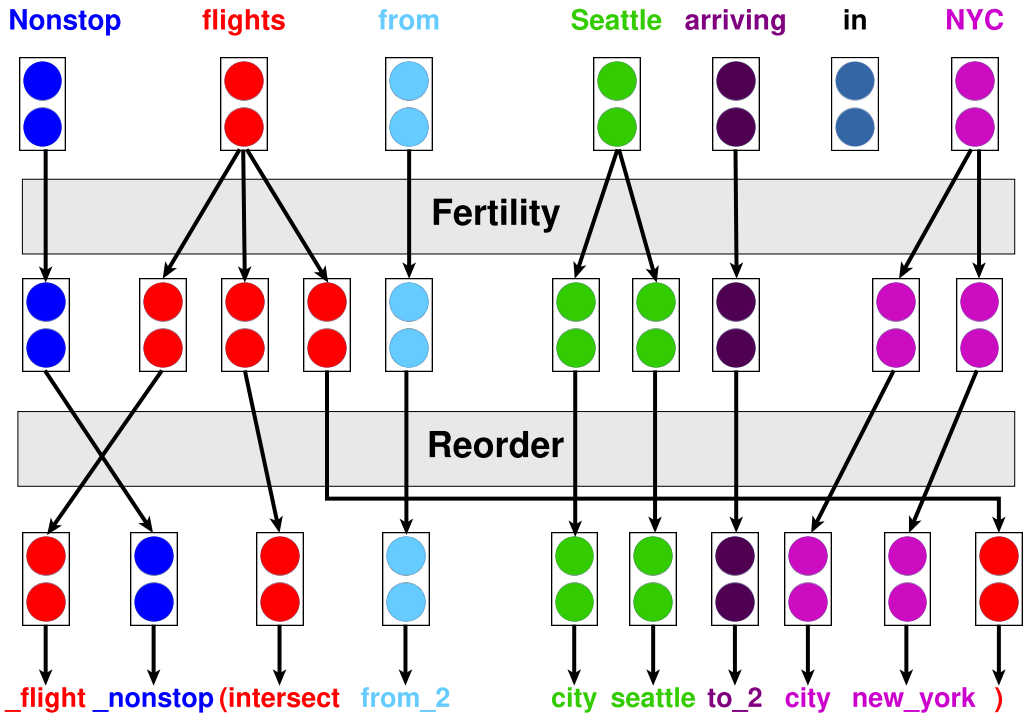


Figure 3.1: We model structural sequence-to-sequence tasks as the composition of differentiable fertility and phrase reordering layers. The model is trained end-to-end without direct supervision of the two structural layers. Output tokens generated by the same input token do not have to be consecutive.

tence. We found this to lead to better generalization than a version of our model using an LSTM-based decoder, and it also outperforms a Transformer trained from scratch. The simple decoder also makes it easy to integrate grammar constraints to ensure well-formed outputs in semantic parsing.

Many sequence-to-sequence models tend to predict the end of the sequence too early on instances that are longer than seen in training (Newman et al., 2020). Our model overcomes this by explicitly predicting the length of the output sequence as a sum of fertilities.

We first demonstrate the expressive capacity and inductive bias of our model on synthetic data. We then evaluate on three English semantic parsing datasets (Geoquery, ATIS, Okapi) and low-resource syntactic transformation tasks (Lyu et al., 2021b) that require the model to rephrase sentences, e.g. from active to passive voice or to topicalize adjectives or verbs. We outperform contemporary sequence-to-sequence models both with and without pretraining in structural generalization setups, particularly when the model has to generalize to longer examples. Our model also compares favorably with existing approaches that target structural generalization.
In short, the main contributions of this chapter are:

- an efficient differentiable fertility layer;
- a flexible end-to-end differentiable model that composes two structural operations (fertility and reordering) and achieves strong performance in structural generalization tasks.

3.1 Background

Structured Attention

We often want to model the relationship between input x of length n and output y of length l by means of a latent variable $\mathbf{Z}$. In particular, we assume that $\mathbf{Z} \in \mathcal{Z} \subseteq \mathbb{B}^{n \times l}$ is a boolean alignment matrix. We want to predict y from a representation produced by a function $g(\mathbf{Z}, x)$. In this case, the marginal distribution

$$P_\theta(y|x) = \mathbb{E}_{P_\theta(\mathbf{Z}|x)} P_\theta(y|g(\mathbf{Z}, x)) \tag{3.1}$$

can be intractable to compute because $\mathcal{Z}$ often has exponential size in x. In some important cases, however, we can formulate a similar but tractable model by using *structured attention* (Kim et al., 2017), which 'pushes' the expectation inside the model:

$$P_\theta(y|x) \approx P_\theta(y|g(\tilde{\mathbf{Z}}, x))$$

where $\tilde{\mathbf{Z}} = \mathbb{E}_{P_\theta(\mathbf{Z}|x)} \mathbf{Z}$ is now a 'soft' rather than a boolean matrix. Note that $\tilde{\mathbf{Z}}_{ij} = P_\theta(\mathbf{Z}_{ij} = 1|x)$ is a marginal probability that often can be efficiently computed with a dynamic program if $P_\theta(\mathbf{Z}|x)$ is factorizable. We will use such an approach for our model.

Marginal Permutations

In this section, we briefly review the method of Wang et al. (2021a) that we use in our model.

Wang et al. (2021a) build on bracketing transduction grammars (Wu, 1997) and show how to compute a distribution over separable permutations (Bose et al., 1998) with a hierarchical inductive bias. A permutation is separable if and only if it can be represented as a permutation tree. Some permutations are not separable. The internal nodes of a permutation tree are labeled as $\wedge$ or $\mathbb{A}$ and are interpreted as operations: $\wedge$ concatenates the values it receives from its left child with the value from its right child, whereas $\mathbb{A}$ concatenates them in reverse order. For example, the permutation tree $t = (\mathbb{A}\,(\wedge\, a\ b)\,(\mathbb{A}\, c\ d))$ represents the permutation $abcd \rightarrow dcab$. For some permutations, there are multiple trees that implement them, e.g. we can reverse $abcd$ to $dcba$ both with $(\mathbb{A}\,(\mathbb{A}\, a\ b)\,(\mathbb{A}\, c\ d))$ as well as with $(\mathbb{A}\, a\,(\mathbb{A}\,(\mathbb{A}\, b\ c)\, d))$.

We use the notation $\mathbf{R}_t(i, j) = 1$ to indicate that the permutation described by tree t maps position i to position j, otherwise $\mathbf{R}_t(i, j) = 0$.

Wang et al. (2021a) show how to compute the expected permutation matrix $\tilde{\mathbf{R}}_{i,j} = \mathbb{E}_{P_\theta(t|x)} \mathbf{R}_t(i, j)$ in polynomial time with a CYK-style algorithm if $P_\theta(t|x)$ factors according to the CYK chart. Crucially, the expected permutation can also be interpreted as a distribution over alignments, where $\tilde{\mathbf{R}}_{i,j}$ is the *marginal probability* that position i aligns to position j. We will use this alignment distribution as a building block in our model.

3.2 Overview of the Approach

In this section, we give a general overview of our approach and defer the details to Sections 3.3 and 3.4.

Conceptually, we want to model the transduction from input x of length n into output y of length l as the composition of two edit operations. First, we apply a *fertility* step, in which we decide for each token what its fertility is, i.e. how many *copies* we make of it. Assigning fertility of 0 corresponds to deleting the token. This step yields an *intermediate* sequence of tokens. We then reorder them using permutation trees (see Section 3.1). In the last step, we individually translate these tokens into the output tokens. Fig. 3.1 shows an example where the fertility step and reordering apply to vector representations of tokens.

The fertility step and the reordering can be represented as boolean matrices $\mathbf{F} \in \mathbb{B}^{n \times l}$ and $\mathbf{R} \in \mathbb{B}^{l \times l}$, respectively. These matrices denote alignments between the sequences before and after the operation. For example, $\mathbf{F}_{i,j} = 1$ means that input token i aligns to intermediate token j, i.e. j is one of the copies of i.

With this conceptualization, we would ideally use the following probabilistic model $P_\theta(y|x)$:

$$P_\theta(y|x) = \mathbb{E}_{\underbrace{P_\theta(\mathbf{F}|x)}_{\text{Fertility}}}\left[\mathbb{E}_{\underbrace{P_\theta(\mathbf{R}|x,\mathbf{F})}_{\text{Reordering}}} \underbrace{P_\theta(y|x,\mathbf{F},\mathbf{R})}_{\text{Decoder}}\right]$$

At training time, the true fertility values are unknown, but we observe the length l of y, so we condition on it:

$$P_\theta(y|x) = P_\theta(l|x) \cdot \mathbb{E}_{P_\theta(\mathbf{F},\mathbf{R}|x,l)} P_\theta(y|x,\mathbf{F},\mathbf{R}) \tag{3.2}$$

where $P_\theta(l|x)$ can be computed with dynamic programming relying on the fertility model, as we explain in the next section.

Computing the marginal likelihood and the gradients is intractable. Instead of computing the likelihood exactly, one could sample and use a score function estimator (Williams, 1992), but the resulting gradient estimates have high variance.

Instead, we use structured attention as discussed in Section 3.1 and 'push' the expectations inside the model:

$$\mathbb{E}_{P_\theta(\mathbf{F},\mathbf{R}|x,l)} P_\theta(\mathbf{y}|\mathbf{x},\mathbf{F},\mathbf{R}) \approx P_\theta(y|x,\tilde{\mathbf{F}},\tilde{\mathbf{R}}) \tag{3.3}$$

with $\tilde{\mathbf{F}} = \mathbb{E}_{P_\theta(\mathbf{F}|x,l)}\mathbf{F}$ and $\tilde{\mathbf{R}} = \mathbb{E}_{P_\theta(\mathbf{R}|x,\tilde{\mathbf{F}})}\mathbf{R}$. $\tilde{\mathbf{F}}$ and $\tilde{\mathbf{R}}$ now represent 'soft', differentiable versions of fertility and reordering. They approach their discrete counterparts as $P_\theta(\mathbf{F}|x,l)$ and $P_\theta(\mathbf{R}|\tilde{\mathbf{F}},x)$ become 'peakier', which tends to happen over the course of training. We can view $(\tilde{\mathbf{F}}\tilde{\mathbf{R}})_{i,j} = \sum_k \tilde{\mathbf{F}}_{i,k}\tilde{\mathbf{R}}_{k,j}$ as a probability of aligning i to j in the composition of the operations.

A tendency to memorize larger chunks of the output might contribute to poor compositional generalization (Hupkes et al., 2020). In order to avoid this, we use a simple decoder that generates each output token independently. A first attempt might look like this:

$$P_\theta(y|x,\tilde{\mathbf{F}},\tilde{\mathbf{R}}) = \prod_{i=1}^{l} \sum_{j,k} P_\theta(y_i|x_j)\tilde{\mathbf{F}}_{j,k}\tilde{\mathbf{R}}_{k,i} \tag{3.4}$$

where the summation over j and k marginalizes over all possible alignments to output position i.

Distinguishing copies The independence assumptions in Eq. (3.4) imply that $P_\theta(y_i|x_j)$ will have the same distribution for *all* copies of x_j, i.e. the model cannot express a preference to translate the first copy of *Seattle* to `city` and the second copy to `seattle` in Fig. 3.1. To enable this, we distinguish different copies of an input token.

We do this by defining $\mathbf{F}$ not as a matrix but as a tensor $\mathbf{F} \in \mathbb{B}^{n \times l \times d}$ where d is some fixed maximum fertility value. Let $\mathbf{F}_{i,j,u} = 1$ iff intermediate token j is the u-th copy of input token i. For example, in Fig. 3.1, $\mathbf{F}_{4,6,1} = 1$ and $\mathbf{F}_{4,7,2} = 1$ as intermediate tokens 6 and 7 are the first and second copy of the input token *Seattle*. With this definition of $\mathbf{F}$ (and accordingly defined $\tilde{\mathbf{F}}$), we can define a stronger decoder:

$$P_\theta(y|x, \tilde{\mathbf{F}}, \tilde{\mathbf{R}}) = \prod_{i=1}^{l} \sum_{j,k,u} P_\theta(y_i|x_j, u) \tilde{\mathbf{F}}_{j,k,u} \tilde{\mathbf{R}}_{k,i}$$

where we now additionally marginalize over which copy of the input sequence we are translating.

3.3 Fertility and Alignment

In this section we describe how to compute $\tilde{\mathbf{F}} = \mathbb{E}_{P_\theta(\mathbf{F}|\mathbf{x},l)}\mathbf{F}$, i.e. the expected alignment that results from the fertility step given that the intermediate token sequence will have length l.

In the previous section, we looked at the fertility step mostly from the perspective of an alignment between the input tokens and the intermediate tokens. However, the fertility step is *parameterized* as assigning a fertility value $\mathbf{f}_i \in 0, \ldots, d \in \mathbb{N}$ to every token x_i. We now show how to compute $\tilde{\mathbf{F}}$ efficiently as a function of the distribution over fertility values $P_\theta(\mathbf{f}|\mathbf{x})$.

We denote the alignment that follows from $\mathbf{f}$ as $\mathbf{F}(\mathbf{f})$, i.e. $\mathbf{F}(\mathbf{f})_{i,j,u} = 1$ iff intermediate token j is the u-th copy of token i. $\tilde{\mathbf{F}}$ can be expressed as:

$$\tilde{\mathbf{F}} = \mathbb{E}_{P_\theta(\mathbf{F}(\mathbf{f})|\mathbf{x},l)}\mathbf{F}(\mathbf{f}) \tag{3.5}$$

where we assume that the fertilities are independent of each other, conditioned on $\mathbf{x}$:

$$P_\theta(\mathbf{F}(\mathbf{f})|\mathbf{x}) = P_\theta(\mathbf{f}|\mathbf{x}) = \prod_{i=1}^{n} P_\theta(\mathbf{f}_i|\mathbf{x})$$

Note that conditioning on the output length l in Eq. (3.5) introduces inter-dependencies between the fertility values. In order to compute $\tilde{\mathbf{F}}_{i,j,u}$, we need to marginalize over all possible assignments to the fertility vector $\mathbf{f}$ which satisfy $\mathbf{F}(\mathbf{f})_{i,j,u} = 1$. We do this with a dynamic programming algorithm that is similar to the forward/backward algorithm for HMMs (Baum, 1972).

Computing marginals We characterize the situations where $\mathbf{F}(\mathbf{f})_{i,j,u} = 1$ as the integer solutions to a set of equations (see also Fig. 3.2). First, in order for intermediate token j to be the u-th copy of i, there should be $j-u$ intermediate tokens generated by input tokens preceding i:

$$\mathbf{f}_1 + \ldots + \mathbf{f}_{i-1} = j - u \tag{3.6}$$

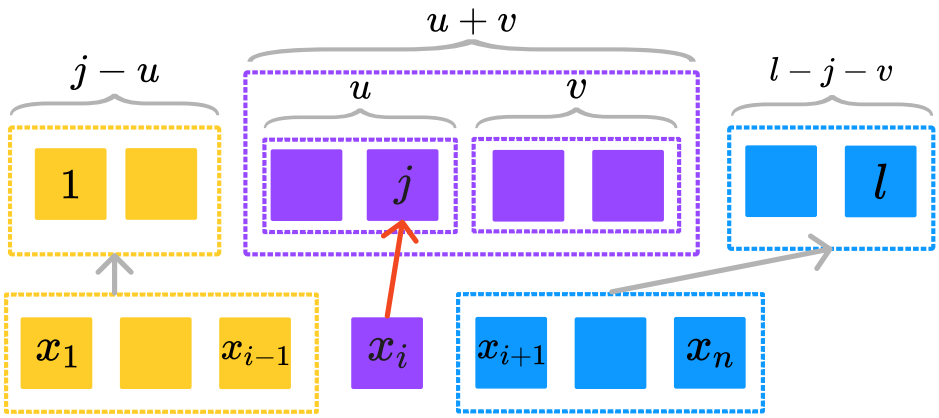


Figure 3.2: Efficiently computing the marginal probability that the u-th copy of i is at position j. We partition the intermediate sequence into four parts, and marginalize over all possible ways of choosing v.

Second, the fertility $\mathbf{f}_i$ has to be at least u. We capture this by requiring

$$\mathbf{f}_i = u + v \tag{3.7}$$

with $v \geq 0$. Finally, the input tokens following i have to contribute the remaining $l-j-v$ intermediate tokens:

$$\mathbf{f}_{i+1} + \ldots + \mathbf{f}_n = l - j - v \tag{3.8}$$

We then compute the probability of creating a sequence of length l, where the j-th intermediate token is the u-th copy of i, by marginalizing over v (dropping $\mathbf{x}$ for readability):

$$\begin{aligned} P_\theta(\mathbf{F}(\mathbf{f})_{i,j,u} = 1, \mathbf{f}_0 + \ldots + \mathbf{f}_{n+1} = l) = \sum_{v=0}^{\min(l-j,d-u)} & P_\theta(\mathbf{f}_0 + \ldots + \mathbf{f}_{i-1} = j-u) \\ & \times P_\theta(\mathbf{f}_i = u+v) P_\theta(\mathbf{f}_{i+1} + \ldots + \mathbf{f}_{n+1} = l-j-v) \end{aligned} \tag{3.9}$$

We handle the boundary cases with $i = 1, i = n$ by adding dummy variables $\mathbf{f}_0$ and $\mathbf{f}_{n+1}$ and setting $P_\theta(\mathbf{f}_0 = 0) = P_\theta(\mathbf{f}_{n+1} = 0) = 1$.

In order to compute Eq. (3.9), we also compute 'forward' probabilities $P_\theta(\mathbf{f}_0 + \ldots + \mathbf{f}_i = h)$ and 'backward' probabilities $P_\theta(\mathbf{f}_i + \ldots + \mathbf{f}_{n+1} = h)$ for all i, h. This can be done recursively:

$$P_\theta(\mathbf{f}_0 + \ldots + \mathbf{f}_i = h) = \sum_{r=0}^{d} P_\theta(\mathbf{f}_i = r) P_\theta(\mathbf{f}_0 + \ldots + \mathbf{f}_{i-1} = h - r)$$

Finally, $\tilde{\mathbf{F}}_{i,j,u}$ was defined in Eq. (3.5) by conditioning on the length l, so:

$$\tilde{\mathbf{F}}_{i,j,u} = \frac{P_\theta(\mathbf{F}(\mathbf{f})_{i,j,u} = 1, \mathbf{f}_0 + \ldots + \mathbf{f}_{n+1} = l)}{P_\theta(\mathbf{f}_0 + \ldots + \mathbf{f}_{n+1} = l)}$$

Run time The run time of the fertility step is dominated by computing Eq. (3.9). Note that v has a range of at most d, thus, we can compute the probabilities for all i, j, u in $O(n \cdot l \cdot d^2)$ time. Eq. (3.9) can be computed in parallel for each i, j, u.

3.4 Composing Fertility and Reordering

We will now describe in detail how $P_\theta(\mathbf{F}|x,l)$ and $P_\theta(\mathbf{R}|x,\tilde{\mathbf{F}})$ are defined and how the fertility and reordering layers are composed.

Fertility Layer The fertility layer takes a desired output length l and a sequence of vectors $\mathbf{x}_1,\ldots,\mathbf{x}_n$ as input (GloVe embeddings (Pennington et al., 2014) in our case) and returns a marginal alignment $\tilde{\mathbf{F}}$ (see Section 3.3) and a sequence of vectors $\mathbf{h}_1,\ldots,\mathbf{h}_l$ for use as input to the next layer. We compute the distribution over fertilities by first encoding $\mathbf{x}_1,\ldots,\mathbf{x_n}$ with a bidirectional LSTM, yielding a sequence of hidden states $\mathbf{h}^f_1,\ldots,\mathbf{h}^f_n$. We then model $P_\theta(f_i|\mathbf{x}) = \text{softmax}_\tau(\text{MLP}^f(\mathbf{h}^f_i))$, where τ is the temperature parameter of the softmax.

We use $\tilde{\mathbf{F}}$ (Eq. (3.5)) as a form of structured attention to compute the input to the reordering layer:

$$\mathbf{h}_j = \sum_{i,u} \tilde{\mathbf{F}}_{i,j,u}(\mathbf{x}_i + \mathbf{w}_u)$$

where $\mathbf{w}_u$ is a learned embedding indicating that j is a u-th copy of some token. Intuitively, $\mathbf{h}_j$ for an intermediate token j represents the corresponding token in the input sequence and also indicates *which* copy of that token it is.

Reordering Layer Given the output $\mathbf{h}_1,\ldots,\mathbf{h}_l$ from the fertility layer, the reordering layer computes the alignment distribution $\tilde{\mathbf{R}}$ as the expected permutation following Wang et al. (2021a). This procedure involves populating a CYK-style chart with scores. We first run a bidirectional LSTM with a skip connection over $\mathbf{h}$ and compute a contextualized representation of the tokens after the fertility step:

$$\mathbf{h}^r_i = [\text{LSTM}^r(\mathbf{h}_{\leq i}); \text{LSTM}^r(\mathbf{h}_{\geq i})] + \mathbf{h}_i$$

Based on these representations, we compute scores for the chart following Stern et al. (2017).

Decoder Our decoder factors as follows (see Section 3.2):

$$P_\theta(y|x,\tilde{\mathbf{R}},\tilde{\mathbf{F}}) = \prod_{i=1}^{l} \underbrace{\sum_{j,k,u} P_\theta(y_i|x_j,u)\tilde{\mathbf{F}}_{j,k,u}\tilde{\mathbf{R}}_{k,i}}_{P_\theta(y_i|x,\tilde{\mathbf{R}},\tilde{\mathbf{F}})}$$

$P_\theta(y_i|x_j,u)$ conditions only on the original input token and on the index indicating which copy of this token we are translating. For this reason, we contextualize the *input* with a bidirectional LSTM with a skip connection:

$$\mathbf{h}'_j = \rho[\text{LSTM}^d(\mathbf{x}_{\leq j}); \text{LSTM}^d(\mathbf{x}_{\geq j})] + \mathbf{x}_j \tag{3.10}$$

with ρ as hyperparameter.

We experiment with three versions of the decoder. In (i), we parameterize $P_\theta(y_i|x_j,u)$ as $P_\theta(y_i|x_j,u) = \text{softmax}(\mathbf{W}_u\text{MLP}(\mathbf{h}'_j))$. In (ii), we additionally use a copy mechanism

(Gu et al., 2016). When there is an ambiguity whether an output token might be copied from the input or generated from the vocabulary, we always assume the former for simplicity. In (iii), we use an autoregressive variant where we encode $y_{<i}$ with an LSTM, defining $P_\theta(y_i|y_{<i},u) = \text{softmax}(\mathbf{W}_u\text{MLP}(\mathbf{h}'_j + \text{LSTM}(y_{<i})))$.

Training As mentioned in Section 3.2, at training time we condition on the observed length l of y: $P_\theta(y|x) = P_\theta(l|x)P_\theta(y|x,\tilde{\mathbf{F}},\tilde{\mathbf{R}})$ with $\tilde{\mathbf{F}}$ conditioned on l. We use a weighted version of the log likelihood as the objective function:

$$\sum_{(x,y)\in\mathcal{D}, l=|y|} \lambda_1 \log P_\theta(l|x) + \log P_\theta(y|x,l)$$

with $\mathcal{D}$ being the training set of input/output pairs and l being the length of y.

For the semantic parsing tasks, we found it necessary to give our model a reasonable starting point in terms of alignments. We encourage it to respect high-confidence automatic alignments during the first m training epochs by adding the following term to our objective function:

$$\lambda_2 \sum_{(i,j)\in\mathcal{A}} \log \sum_{k,u} \tilde{\mathbf{F}}_{i,k,u}\tilde{\mathbf{R}}_{k,j}$$

where $\mathcal{A}$ is a set of high-confidence alignments with a posterior probability of at least χ according to an IBM-1 alignment model (Brown et al., 1993). Initializing an alignment model with alignments from a simpler model was a common strategy in statistical machine translation (Och and Ney, 2003; Liang et al., 2006).

Inference In order to make predictions with a trained model, we want to compute the most likely output y given the input x. It is convenient to treat the length as a discrete variable and use the same algorithm for computing $\tilde{\mathbf{F}}$ as derived for training. We therefore search for $\arg\max_y P_\theta(y|x) = \arg\max_l P_\theta(l|x)P_\theta(y^*(l)|x,l)$ with $y^*(l)$ being the most likely output of length l, i.e. $y^*(l) = \arg\max_y P_\theta(y|x,l)$. Recall that in versions (i) and (ii) of the decoder, each output position is conditionally independent given x, so we can easily find $y^*(l)$ for any l exactly:

$$y^*(l)_i = \arg\max_{y_i} P_\theta(y_i|x,\tilde{\mathbf{R}},\tilde{\mathbf{F}})$$

with $\tilde{\mathbf{F}} = \mathbb{E}_{P_\theta(\mathbf{F}|\mathbf{x},l)}\mathbf{F}$ conditioned on l. For version (iii) of the decoder, we use greedy search instead. It would be too costly to compute $y^*(l)$ for all l, so we explore only the top k most likely lengths.

Grammar-based decoding In executable semantic parsing, we want to produce only well-formed outputs, which can be characterized by a context-free grammar G. In practical applications, this grammar is needed to execute the query, so there is relatively little extra engineering effort in using it for decoding. We search for

$$y^*(l) = \arg\max_{y\in L(G)} P_\theta(y|x,l)$$

Because of the simple decoder, we can do this *exactly* by applying a modified version of Viterbi CYK. Unlike in parsing, the string is not observed because it is exactly what we are looking for. Therefore, we fill all entries from i to i in the chart C with $C_{A,i,i} = \max_{A \to a \in G} P_\theta(y_i = a | x, \tilde{\mathbf{R}}, \tilde{\mathbf{F}})$. We then continue with the normal Viterbi CYK with weights of 1 on all other rules. Effectively, this adaptation of CYK pairs each string $y \in L(G)$ with a derivation that has the weight $P_\theta(y|x,l)$ and in which *only* the choice of terminal symbols determines the weight. If a string y has multiple derivations according to G, they all have the same weight. Hence, finding the *derivation* with the highest weight using CYK coincides with finding the most likely *string* according to $P_\theta(y|x,l)$. Note that we do not want to marginalize over all derivations of a string because we care about $P_\theta(y|x,l)$ rather than a distribution defined by G.

3.5 Evaluation

Synthetic Data

In order to probe the expressive capacity and inductive bias of our model, we first evaluate on a mirroring task $T = \{(w, ww^R) | w \in \Sigma^*\}$, e.g. $abc \to abccba$. The challenge is that the length of the dependency between output tokens grows with the length of the example. In our experiments, models are trained on examples with input lengths 3 to 9, and tested in two setups. In the first setup (Length), the model has to generalize to examples with lengths 11 to 20; we use examples with length 10 as validation data. In the second setup (unseen combination, UC), the model only sees the symbols `x`, `y` and `z` grouped together as `xyz` on the input side at training time. The model is tested on examples that contain `x`, `y` or `z` adjacent to other symbols. See Appendix B.1 for further details on the setup.

We compare our model without copying (F→R) with a variant that first applies the reordering and then the fertility step (R→F) and autoregressive variants of the two (AR F→R and AR R→F). As baselines, we also compare with an LSTM-based seq2seq model with attention and a Transformer with relative positional embeddings, which was previously shown by Csordás et al. (2021) to perform well at compositional generalization. AR R→F has similarities to Wang et al. (2021a) who first reorder and then use an autoregressive decoder with monotonic attention.

Results Table 3.1 shows mean accuracy across 5 random initializations. The accuracy of the relative Transformer and the LSTM-based seq2seq model drops sharply for longer inputs. In contrast, F→R and AR F→R generalize perfectly even to much longer examples. In the UC setup, F→R outperforms the rest by a wide margin. Interestingly, all autoregressive models struggle in this setup, including AR F→R which obtained perfect accuracy in the Length setup. This is consistent with the hypothesis that autoregressive models tend to memorize entire chunks (Hupkes et al., 2020).

Expressivity of F→R and R→F For this task, an input token corresponds to two output tokens that may be arbitrarily far apart from each other. F→R can learn this alignment because this can be captured by a separable permutation (see Section 3.1) following the fertility step, duplicating every input token. In contrast, R→F and AR

Model	Accuracy					
		Length				UC
	dev	11	12	13	14 - 20	test
Transformer	82.0	3.0	0.0	0.0	0.0	42.0
LSTM	**100.0**	94.3	67.9	6.9	0.0	0.1
R→F	0.0	0.0	0.0	0.0	0.0	1.3
AR R→F	90.0	85.7	89.8	87.5	80.5	1.2
F→R	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**79.9**
AR F→R	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	32.1

Table 3.1: Exact match accuracy on the **mirroring** task.

R→F cannot represent the correct alignment directly because the fertility step is applied only after the reordering, leading to alignments between an input token and a *contiguous span* in the output. However, in theory, they can represent this alignment *implicitly* through the LSTM (Eq. (3.10)). The evaluation shows that this does not work reliably in practice: we find that R→F gets stuck in bad local minima and fails completely on the task. While AR R→F performs well in the Length setup, it is weak in the UC setup.

Geoquery

Geoquery (Zelle and Mooney, 1996) is a standard dataset for semantic parsing and has recently been used to evaluate to what extent semantic parsers are capable of generalizing to (i) structurally unseen queries (template split), and (ii) structurally unseen long examples. We follow the setup of Herzig and Berant (2021), using the variable-free FunQL representation (Kate et al., 2005), a copy mechanism, and evaluate with execution accuracy.

Results Table 3.2 shows the results on the different splits. We report means and standard deviations of 5 random initializations. Our method performs well across the different splits, and in particular on the length split that evaluates a challenging form of compositional generalization. As an ablation, we remove the grammar-based decoding. We notice a considerable drop in accuracy, but it still outperforms the baselines. The drop in accuracy is slightly bigger out-of-distribution than in-distribution.

In line with the experiments on synthetic data, AR F→R and the approach of Wang et al. (2021a) drastically lose accuracy when going from in-distribution to the compositional generalization setups. This provides further evidence that a strong decoder can hinder compositional generalization.

In contrast to the experiments on synthetic data, R→F and F→R perform comparably. Manual inspection of the data shows that good alignments on this dataset can be obtained even with the stronger assumption on possible alignments made by R→F.

Model	iid	Template	Length
Seq2Seq (HB)	78.5	46.0	24.3
Seq2Derivation † (HB)	72.1	54.0	24.6
BART-base (HB)	87.1	67.0	19.3
Span (HB) †	78.9	65.9	41.4
Span + lexicon (HB) †	86.1	82.2	63.6
Liu et al. (2021a)	-	**84.1**	-
Wang et al. (2021a)	75.2*	43.2*	-
R→F †	**89.1**±1.0	80.4±1.2	68.6±1.4
R→F	83.5±0.7	73.0±1.6	49.2±5.1
F→R †	88.6±3.3	79.9±2.5	**68.8**±5.2
F→R	80.7±2.3	68.7±4.3	53.4±5.9
AR F→R	81.1±1.0	52.8±3.3	37.6±1.8

Table 3.2: Execution accuracy on different splits of **Geoquery**. HB is Herzig and Berant (2021) and † refers to systems that enforce well-formedness of the output. * Wang et al. (2021a) use exact match accuracy and anonymize named entities instead of copying.

Model	iid	Length
LSTM seq2seq†	76.52±1.66	4.95±2.16
LSTM seq2seq	75.98±1.30	4.95±2.16
Rel. Transformer	75.76±1.43	1.15±1.41
BART-base	**86.96**±1.26	19.03±4.57
F→R †	74.15±1.35	**35.41**±4.09
F→R	68.26±1.53	29.91±2.91

Table 3.3: Accuracy on different splits of **ATIS**.

ATIS

ATIS (Dahl et al., 1994) is a semantic parsing datasets for flight bookings. In comparison to Geoquery, the queries tend to be longer and the word order is more flexible. We use the variable-free FunQL notation as annotated by Guo et al. (2020). Apart from the original iid test split, we create a length split: Semantic parses with fewer than 4 conjuncts form the training set, parses with exactly 4 conjuncts form the development set and the test set contains instances with more than 4 conjuncts. Details on the split and preprocessing are in Appendix B.1 and run times are in Table B.8.

We compare our model with finetuned BART-base (Lewis et al., 2020b), an LSTM-based seq2seq model with attention, and the relative Transformer (Csordás et al., 2021). We also run a version of the LSTM-based model with a large beam of size 50 and filter out instances that are not well-formed; the resulting outputs are well-formed at least 99.7% of the time.

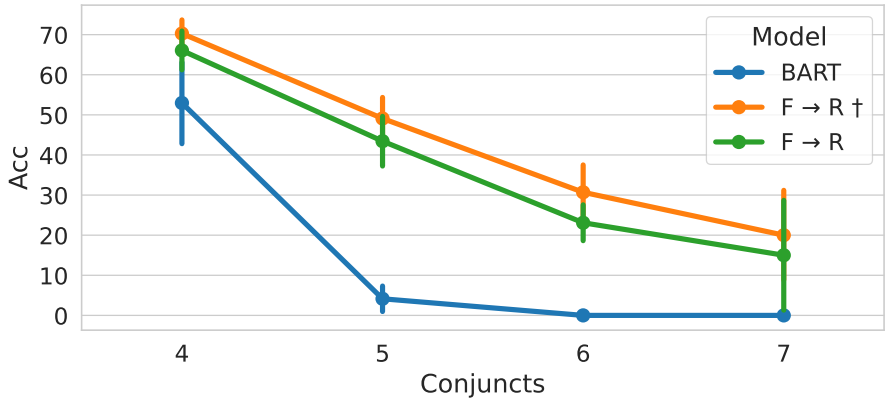


Figure 3.3: Accuracy on the document domain of **Okapi** by number of parameters in the gold logical form.

Model	Calendar	Document	Email
BART-base	36.7±3.0	2.7±2.1	20.5±9.8
F→R †	**69.5**±13.9	**42.4**±5.7	**55.6**±2.7
F→R	57.2±19.9	36.1±5.6	43.9±3.8

Table 3.4: Accuracy on length splits by domain on **Okapi**.

Results Table 3.3 shows mean accuracy and standard deviations of 5 random initializations. While our approach does not reach the same accuracy as the baselines on the iid split, it outperforms them on the compositional length split by a margin of more than 16 points. Without grammar-based decoding, we again observe a noticeable loss in accuracy, but we still substantially outperform BART on the length split. Constraining the output to be grammatical does not appear as beneficial for the LSTM baseline.

Okapi

Hosseini et al. (2021) introduce a semantic parsing dataset for evaluating compositional generalization on three domains (document, calendar, email). Since template splits were not found to be challenging, we focus on generalizing to longer examples. Models are trained on short examples with up to 3 'conjuncts' (such as filtering based on an attribute or ordering results) and are tested on examples with more conjuncts (at least 4 for the calendar and email domain and at least 5 for the document domain). The splits are described in Appendix B.1. In contrast to the other datasets we consider, Okapi is noisy because it was collected with crowd workers. This presents an additional challenge.

Results Table 3.4 shows the results of our model with copying from 5 random initializations. F→R outperforms fine-tuned BART-base by a large margin, in particular on the challenging split of the document domain.

Transfer Type	Approach	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L	CIDEr
Active to passive	GPT-2 (Lyu et al., 2021b)	47.6	32.9	23.8	18.9	21.6	46.4	1.820
	Seq2seq (Kim, 2021)	83.8	73.5	67.3	59.8	46.7	77.1	5.941
	Neural QCFG (Kim, 2021)	83.6	77.1	71.3	66.2	49.9	80.3	6.410
	Human (Lyu et al., 2021b)	93.1	88.1	83.5	79.5	58.7	90.5	8.603
	F→R	**92.1**±1.3	**85.4**±1.5	**79.7**±1.7	**74.6**±1.9	**55.9**±0.5	**86.0**±1.1	**7.610**±0.130
Adj. emphasis	GPT-2 (Lyu et al., 2021b)	26.3	7.9	2.8	0.0	11.2	18.8	0.386
	Seq2seq (Kim, 2021)	50.5	29.6	18.4	11.9	24.2	51.4	1.839
	Neural QCFG (Kim, 2021)	67.6	50.6	39.3	31.6	37.3	68.3	3.424
	Human	83.4	75.3	67.9	66.1	52.2	81.1	6.796
	F→R	**78.3**±1.4	**61.9**±0.7	**49.9**±1.3	**40.5**±1.4	**43.0**±1.0	**69.1**±0.6	**4.268**±0.170
Verb emphasis	GPT-2 (Lyu et al., 2021b)	30.9	17.0	9.5	4.1	14.0	29.2	0.593
	Seq2seq (Kim, 2021)	52.6	38.9	29.4	21.4	29.4	46.4	2.346
	Neural QCFG (Kim, 2021)	66.4	51.2	40.7	31.9	37.0	**58.9**	3.227
	Human (Lyu et al., 2021b)	64.9	56.9	49.3	42.1	43.3	69.3	5.668
	F→R	**68.4**±0.6	**52.7**±0.6	**41.6**±0.7	**32.8**±0.6	**37.4**±0.4	**58.9**±0.4	**3.498**±0.121

Table 3.5: Results on the **syntactic transformation** tasks from Lyu et al. (2021b). All models except for GPT-2 use copying.

Fig. 3.3 shows the accuracy on the development and test sets of the document domain as a function of the number of conjuncts in the gold logical form. BART performs relatively well when applied to examples with one more conjunct than seen in the training set, but then its performance drops sharply. F→R is more accurate and its accuracy also drops much slower with the number of conjuncts. We notice different failure modes for the two models: on the test set, BART deviates by 5.4 tokens on average from the gold length, whereas F→R deviates only by 1.0 tokens on average. This is in line with the observations of Newman et al. (2020) that seq2seq models systematically predict the end of sequence token too early on long out-of-distribution examples. Our results suggest that predicting length as a sum of fertilities is more robust towards this shift in distribution.

Syntactic Transformation Tasks

In addition to being important for compositional generalization, structural inductive biases can help when only little data is available. We evaluate our model in such a scenario on the syntactic transformation ('style transfer') tasks of Lyu et al. (2021b). A model is given an English sentence and has to reformulate it to conform with a certain 'style'. We focus on the tasks identified as challenging by Lyu et al. (2021b): *active to passive* (2462 training examples), *adjective emphasis* (627 examples) and *verb emphasis* (1081 examples).

For the emphasis tasks, the word to be emphasized is provided in the input. Following Kim (2021), to incorporate it, we add a special learned embedding vector to the embedding of that token.

Results Table 3.5 shows the results comparing our F→R model to previous work based on three random initializations. Our method performs as well or better than the baselines on all syntactic transformation tasks for all metrics. The improvement compared to prior work is strongest for the active-to-passive task and weakest for the verb emphasis task, where our model ties with Kim (2021) in terms of ROUGE-L.

3.6 Related Work

Fertility The concept of fertility was introduced by Brown et al. (1990) for statistical machine translation to capture that a word in one language is often consistently translated to a certain number of words in another language. Tu et al. (2016) and Malaviya et al. (2018) incorporate fertility into the attention mechanism of sequence-to-sequence models. Cohn et al. (2016); Gu et al. (2016) use heuristic supervision for training their fertility models. In contrast to prior work, we learn an explicit fertility component jointly with the rest of our model.

Monotonic alignment Related to fertility is the concept of monotonic alignment, i.e. an alignment a that maps output positions to input positions such that for any two output positions $i < j$, $a(i) \leq a(j)$. The alignment produced by our fertility layer is monotonic. Monotonic alignments are usually modeled by an HMM-like model that places the monotonicity constraint on the transition matrix (Yu et al., 2016; Wu and Cotterell, 2019), leading to a run time of $O(|x|^2|y|)$ with x being the input string and y the output string. Raffel et al. (2017) parameterize the alignment using a series of Bernoulli trials and obtain a training run time of $O(|x||y|)$. Our approach also has $O(|x||y|)$ run time.

Compositional generalization There is a growing body of work on improving the ability of neural models to generalize compositionally, in particular in semantic parsing. We provide a broad overview in Section 2.4 and focus here on the most closely related works.

We use the reordering mechanism of Wang et al. (2021a) who design a sequence-to-sequence model that first reorders phrases and then uses a monotonic attention mechanism on top. Our approach differs from theirs in several important aspects: (i) we use fertility instead of monotonic attention, which parameterizes alignments differently; (ii) we apply the fertility step first and *then* reorder the phrases, so our model can directly create alignments where output tokens aligned to the same input token do not have to be consecutive; (iii) we predict the length as the sum of the fertilities and not with an end-of-sequence token; (iv) they use an LSTM-based decoder network whereas we found that a simpler decoder can generalize better.

Concurrently[1] with this work, Cazzaro et al. (2023) propose an approach that first monotonically translates the input and then reorders the translation. They phrase each of those steps with standard sequence-to-sequence models or as tagging, i.e. each input token is tagged with a single output token or a special null token based on its contextualized representation. They show improvements for compositional generalization on a semantic parsing version of the SCAN benchmark (Lake and Baroni, 2018; Herzig and Berant, 2021) and on Geoquery in multiple languages. Using standard pretrained models in both stages makes the model expressive but requires annotated training data for the intermediate sequence, i.e. after the monotonic translation and before reordering. To create these annotations, they assume alignments between input and output as given, either from an external alignment model or gold alignments. This is in contrast to the method we present here, which induces alignments as part of training and

[1] A surprising 4 days of difference between arxiv preprints.

is end-to-end differentiable. They do not constrain the reordering model to output a permutation of its input.

3.7 Conclusion

In this chapter, we have presented a first approach towards Goal 1 and proposed a flexible end-to-end differentiable sequence-to-sequence model with strong structural generalization capabilities, in particular for semantic parsing. It predicts the output sequence from the input by composing a fertility layer with a reordering layer. This explicitly captures correspondences between input and output. Our evaluation confirms that this provides an inductive bias that enables structural generalization, in particular when generalizing to longer examples or deeper recursion in the logical forms. Our approach also proves effective for syntactic transformation tasks such as emphasizing adjectives when only small amounts of data are available.

The efficient fertility layer introduced in this work may be useful in other scenarios beyond structural generalization, e.g. in non-autoregressive machine translation, or to improve the efficiency of large models on long documents by compressing the document using fertilities restricted to 0 or 1. Future work could also investigate other structured layers that make edit operations differentiable (e.g. Soulos et al., 2024) and the best ways of composing and training them.

Limitations In some situations, we found the model presented here to be sensitive to hyperparameters. In particular, for Geoquery, a considerable fraction of the data does not require a reordering, i.e. it leaves the output from the fertility layer unchanged. In this case, the reordering model has a risk of overgeneralization, leaving *all* sequences unchanged, which is exceedingly hard to recover from (see also Section 4.7 for a discussion about the prominent status of the identity permutation in the reordering model). Here, we addressed this issue heuristically by hyperparameter selection, in particular using a smaller learning rate of the reordering model to avoid overgeneralization early during training.

While the fertility layer presented in this chapter is efficient, it is not entirely straightforward to implement for GPUs. For this reason, we move the data to RAM and perform the computation on the CPU using numba (Lam et al., 2015) with a manually defined backward pass. The computational bottleneck of the model is the high run time complexity of the reordering layer (see Table B.8 for run times in our experiments). For moderately long output sequences (e.g. more than 50 tokens), the reordering approach taken here becomes impractically slow and memory-demanding. In addition, while the structured reordering step can represent many permutations of practical interest, we observed some cases where our model could not produce the correct output because it could not represent the necessary permutation.

In the next chapter, we will address some of these limitations with a more flexible permutation model that can – in principle – predict any permutation and is also more efficient in practice, making such an approach applicable to larger datasets with longer instances.

Chapter 4

Multiset Tagging and Latent Permutations

In the previous chapter, we have presented a first approach towards Goal 1, the design of a sequence-to-sequence model with an inductive bias that enables structural generalization for semantic parsing. This chapter – based on Lindemann et al. (2023b) – refines the approach and addresses some limitations we identified in the previous chapter, mainly concerning the reordering model. The reordering model used in the previous chapter is restricted in its expressivity and can only predict a specific class of permutations (separable permutations), limiting its applicability to tasks and datasets that approximately conform to this restriction. In addition, it is computationally very expensive and becomes impractical for moderately long sequences. In this chapter, we propose a new permutation model that does not put any restrictions on which permutations can be predicted and empirically is also computationally cheaper. This makes the approach more broadly applicable, including to the COGS benchmark (Kim and Linzen, 2020).

In order for a model to generalize structurally in semantic parsing, it has to identify reusable 'fragments' and be able to systematically combine them in novel ways. One way to make a model sensitive to fragments is to make it rely on a tree that makes the compositional structure explicit. However, this complicates the training because these trees are usually not part of the training data. Trees can be induced as part of training, but this is computationally very demanding (Kim, 2021; Wang et al., 2021a). In principle, providing gold-standard trees directly is another option. In some cases, one can use domain knowledge and structural preprocessing to construct trees (Weißenhorn et al., 2022), but this is not always possible and manual annotation is slow and expensive.

In this chapter, we propose a two-step sequence-based approach with a structural inductive bias that does not rely on trees: the 'fragments' are multisets of output tokens that we predict for every input token in a first step. A second step then arranges the tokens we predicted in the previous step into a single sequence using a permutation model. In contrast to Chapter 3 and other prior work (Wu, 1997; Zhang and Gildea, 2007; Stanojević and Sima'an, 2015; Wang et al., 2021a), there are no hard constraints on the permutations that our model can predict. We show that this enables structural generalization on tasks that go beyond the class of synchronous context-free languages.

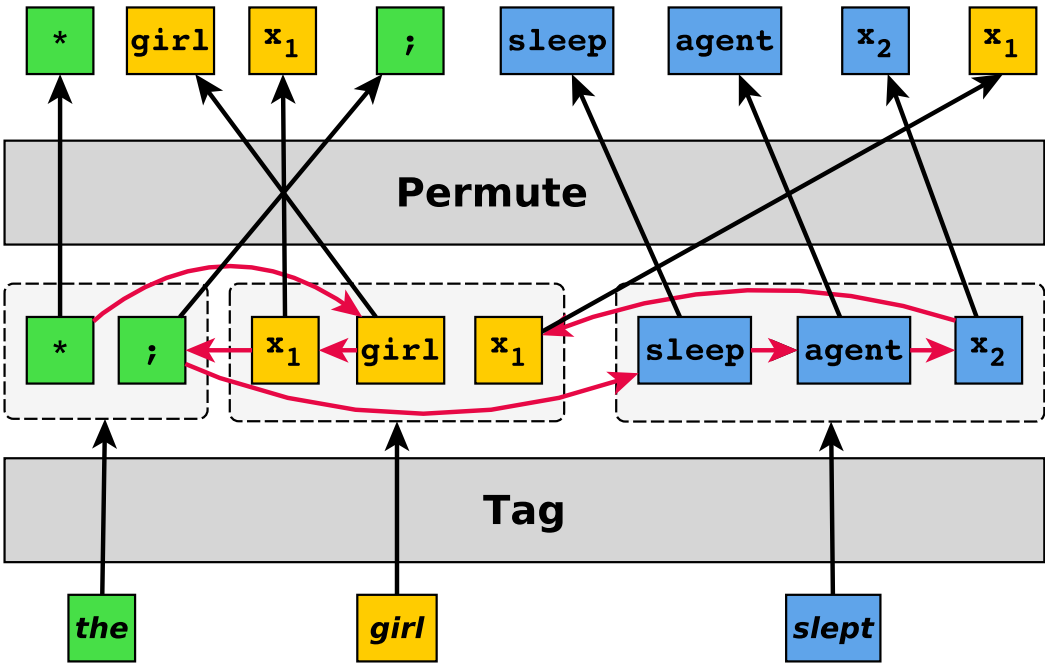


Figure 4.1: We model seq2seq tasks by first predicting a multiset (dashed boxes) for every input token, and then permuting the tokens to put them into order. We realize the permutation using the red edges: if there is an edge from token i to j, then i is the predecessor of j in the output. Every token is visited exactly once.

We overcome two key technical challenges in this chapter: Firstly, we do not have supervision for the correspondence between input tokens and output tokens. Therefore, we induce the correspondence during training. Secondly, predicting permutations without restrictions is computationally challenging. For this, we develop a differentiable, GPU-friendly algorithm.

On realistic semantic parsing tasks, our approach outperforms previous work on generalization to longer examples than seen at training. We also outperform all other non-tree models on the structural generalization tasks in semantic parsing on COGS (Kim and Linzen, 2020). For the first time, we also show that a model *without* an inductive bias towards trees can obtain high accuracy on generalization to deeper recursion on COGS.

In short, our main contributions in this chapter are:

- a flexible sequence-to-sequence model that performs well on structural generalization in semantic parsing without assuming that input and output are related to each other via a tree structure.
- a differentiable algorithm to parameterize and predict permutations without a priori restrictions on what permutations are possible.

4.1 Overview and Motivation

Our approach consists of two stages. In the **first stage** (multiset tagging), we predict a multiset $\mathbf{z}_i$ of tokens for every input token $\mathbf{x}_i$ from the contextualized representation of $\mathbf{x}_i$. This is motivated by the observation that input tokens often systematically correspond to a fragment of the output (like *slept* corresponding to `sleep` and `agent` and a variable in Fig. 4.1). Importantly, we expect this systematic relationship to be largely invariant to phrases being used in new contexts or deeper recursion. We refer to the elements of the multisets as *multiset tokens*.

In the **second stage** (permutation), we order the multiset tokens to arrive at a sequential output. Conceptually, we do this by going from left to right over the *output* and determining which multiset token to put in every position. Consider the example in Fig. 4.1. For the first output position, we simply select a multiset token (`*` in the example). All subsequent tokens are put into position by 'jumping' from the token that was last placed into the output to a new multiset token. In Fig. 4.1, we jump from `*` to `girl` (shown by the outgoing red edge from `*`). This indicates that `girl` is the successor of `*` in the output and hence the *second* output token. From `girl` we jump to one of the $\mathtt{x}_1$ tokens to determine what the *third* output token is and so on. Since we predict a permutation, we must visit every multiset token exactly once in this process.

The jumps are inspired by reordering in phrase-based machine translation (Koehn et al., 2007) and methods from dependency parsing, where directly modeling bi-lexical relationships on hidden states has proven successful (Kiperwasser and Goldberg, 2016). Note also that *any* permutation can be represented with jumps. In contrast, prior work (Wang et al., 2021a; Lindemann et al., 2023a) has put strong restrictions on the possible permutations that can be predicted. Our approach is more flexible and empirically it also scales better to longer inputs, which opens up new applications and datasets.

Setup We assume we are given a dataset $\mathcal{D} = \{(x^1, y^1), \ldots\}$ of input utterances x and target sequences y. If we had gold alignments, it would be straightforward to train our model. Since we do not have this supervision, we have to discover during training which tokens of the output y belong into which multiset $\mathbf{z}_i$. We describe the model and the training objective of the multiset tagging in Section 4.2. After the model is trained, we can annotate our training set with the most likely $\mathbf{z}$, and then train the permutation model.

For predicting a permutation, we associate a score with each possible jump and search for the highest-scoring sequence of jumps. We ensure that the jumps correspond to a permutation by means of constraints, which results in a combinatorial optimization problem. The flexibility of our model and its parametrization come with the challenge that this problem is NP-hard. We approximate it with a regularized linear program which also ensures differentiability. Our permutation model and its training are described in Section 4.3. In Section 4.4, we discuss how to solve the regularized linear program and how to backpropagate through the solution.

4.2 Learning Multisets

For the multiset tagging, we need to train a model that predicts the multisets $\mathbf{z}_1, \ldots, \mathbf{z}_n$ of tokens by conditioning on the input. We represent a multiset $\mathbf{z}_i$ as an integer-valued vector that contains for every vocabulary item v the multiplicity of v in $\mathbf{z}_i$, i.e. $\mathbf{z}_{i,v} = k$ means that input token i contributes k occurrences of output type v. If v is not present in multiset $\mathbf{z}_i$, then $\mathbf{z}_{i,v} = 0$. For example, in Fig. 4.1, $\mathbf{z}_{3,\texttt{sleep}} = 1$ and $\mathbf{z}_{2,\texttt{x}_1} = 2$. As discussed in Section 4.1, we do not have supervision for $\mathbf{z}_1, \ldots, \mathbf{z}_n$ and treat them as latent variables. To allow for efficient exact training, we assume that all $\mathbf{z}_{i,v}$ are independent of each other conditioned on the input:

$$P_\theta(\mathbf{z} \mid x) = \prod_{i,v} P_\theta(\mathbf{z}_{i,v} \mid x) \tag{4.1}$$

where v ranges over the *entire* vocabulary.

Parametrization We parameterize $P_\theta(\mathbf{z}_{i,v} \mid x)$ as follows. We first embed the input x using the static word embeddings of RoBERTa (Liu et al., 2019), yielding a sequence of vectors $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_n$. We then pass $\mathbf{x}$ as usual through the pretrained RoBERTa encoder, where $\textsc{encoder}(\mathbf{x})$ is the output of the final layer. To obtain the hidden representation $\mathbf{h}_i$, we use a skip connection and add the static word embedding to the output of the encoder:

$$\mathbf{h}_i = \textsc{encoder}(\mathbf{x})_i + \mathbf{x}_i \tag{4.2}$$

We then pass $\mathbf{h}_i$ through a feedforward network obtaining $\tilde{\mathbf{h}}_i = \textsc{ff}(\mathbf{h}_i)$ and define a distribution over the multiplicity of v in the multiset $\mathbf{z}_i$:

$$P_\theta(\mathbf{z}_{i,v} = k | x_i) = \frac{\exp\left(\tilde{\mathbf{h}}_i^T \mathbf{w}^{v,k} + b^{v,k}\right)}{\sum_l \exp\left(\tilde{\mathbf{h}}_i^T \mathbf{w}^{v,l} + b^{v,l}\right)} \tag{4.3}$$

where the weights $\mathbf{w}$ and biases b are specific to v and the multiplicity k. In contrast to standard sequence-to-sequence models, this softmax is not normalized over the vocabulary but over the multiplicity, and we have distinct distributions for every vocabulary item v. Despite the independence assumptions in Eq. (4.1), the model can still be strong because $\mathbf{h}_i$ takes the entire input x into account.

Training The probability of generating the overall multiset $\mathbf{m}$ as the union of all $\mathbf{z}_i$ is the probability that for every vocabulary item v, the total number of occurrences of v across all input positions i sums to $\mathbf{m}_v$:

$$P_\theta(\mathbf{m}|x) = \prod_v P_\theta(\mathbf{z}_{1,v} + \ldots + \mathbf{z}_{n,v} = \mathbf{m}_v | x)$$

This can be computed recursively:

$$P_\theta(\mathbf{z}_{1,v} + \ldots + \mathbf{z}_{n,v} = \mathbf{m}_v \mid x) = \sum_k P_\theta(\mathbf{z}_{1,v} = k|x) P_\theta(\mathbf{z}_{2,v} + \ldots \mathbf{z}_{n,v} = \mathbf{m}_v - k|x)$$

Let $\mathbf{m}(y)$ be the multiset of tokens in the gold sequence y. We train our model with gradient ascent to maximize the marginal log-likelihood of $\mathbf{m}(y)$:

$$\sum_{(x,y)\in\mathcal{D}} \log P_\theta(\mathbf{m}(y) \mid x) \tag{4.4}$$

Like in Chapter 3, we found it helpful to initialize the training of our model with high-confidence alignments from an IBM-1 model (Brown et al., 1993) (see Appendix C.2 for details).

Preparation for permutation The scoring function of the permutation model expects a sequence as input. There is no a priori obvious order for the elements within the individual multisets $\mathbf{z}_i$. We handle this by imposing a canonical order $\textsc{Order}(\mathbf{z}_i)$ on the elements of $\mathbf{z}_i$ by sorting the multiset tokens by their id in the vocabulary. They are then concatenated to form the input $z' = \textsc{Order}(\mathbf{z}_1)\ldots\textsc{Order}(\mathbf{z}_n)$ to the permutation model.

4.3 Relaxed Permutations

After predicting a multiset for every input token and arranging the elements within each multiset to form a sequence z', we predict a permutation of z'.

We represent a permutation as a matrix $\mathbf{V}$ that contains exactly one 1 in every row and column and zeros otherwise. We write $\mathbf{V}_{i,j} = 1$ if position i in z' is mapped to position j in the output y. Let $\mathcal{P}$ be the set of all permutation matrices.

Now we formalize the parametrization of permutations as discussed in Section 4.1. We associate a score predicted by our neural network with each permutation $\mathbf{V}$ and search for the permutation with the highest score. The score of a permutation decomposes into a sum of scores for binary 'features' of $\mathbf{V}$. We use two types of features.

The first type of feature is active if the permutation $\mathbf{V}$ maps input position i to output position j (i.e. $\mathbf{V}_{ij} = 1$). We associate this feature with the score $\mathbf{s}_{i\mapsto j}$ and use these scores only to model what the first and the last token in the output should be. That is, $\mathbf{s}_{i\mapsto 1}$ models the preference to map position i in the input to the first position in the output, and analogously $\mathbf{s}_{i\mapsto n}$ models the preference to put i into the last position in the output. For all output positions j that are neither the first nor the last position, we simply set $\mathbf{s}_{i\mapsto j} = 0$.

The second type of feature models the jumps we introduced in Section 4.1. We introduce a feature that is 1 iff $\mathbf{V}$ contains a jump from k to i, and associate this with a score $\mathbf{s}_{k\curvearrowright i}$. In order for there to be a jump from k to i, the permutation $\mathbf{V}$ must map input i to some output position j ($\mathbf{V}_{ij} = 1$) *and* it must also map input position k to output position $j-1$ ($\mathbf{V}_{k,j-1} = 1$). Hence, the product $\mathbf{V}_{k,j-1}\mathbf{V}_{ij}$ is 1 iff there is a jump from k to i at output position j. Based on this, the sum $\sum_j \mathbf{V}_{k,j-1}\mathbf{V}_{ij}$ equals 1 if there is *any* output position j at which there is a jump from k to i and 0 otherwise. This constitutes the second type of feature.

Multiplying the features with their respective scores, we want to find the highest-

scoring permutation under the following overall scoring function:

$$\arg\max_{\mathbf{V}\in\mathcal{P}} \sum_{i,j} \mathbf{V}_{ij}\mathbf{s}_{i\mapsto j} + \sum_{i,k} \mathbf{s}_{k\curvearrowright i}\left(\sum_{j} \mathbf{V}_{k,j-1}\mathbf{V}_{ij}\right) \tag{4.5}$$

Let $\mathbf{V}^*(\mathbf{s})$ be the solution to Eq. (4.5) as a function of the scores. Unfortunately, $\mathbf{V}^*(\mathbf{s})$ does not have sensible derivatives because $\mathcal{P}$ is discrete. This makes $\mathbf{V}^*(\mathbf{s})$ unsuitable as a neural network layer. In addition, Eq. (4.5) is NP-hard (see Appendix C.1.1).

We now formulate an optimization problem that approximates Eq. (4.5) and which has useful derivatives. Firstly, we relax the permutation matrix $\mathbf{V}$ to a bistochastic matrix $\mathbf{U}$, i.e. $\mathbf{U}$ has non-negative elements and every row and every column each sum to 1. Secondly, note that Eq. (4.5) contains quadratic terms. As we will discuss in the next section, our solver assumes a linear objective, so we replace $\mathbf{V}_{k,j-1}\mathbf{V}_{ij}$ with an auxiliary variable $\mathbf{W}_{ijk}$. The variable $\mathbf{W}_{ijk}$ is designed to take the value 1 if and only if $\mathbf{U}_{i,j} = 1$ *and* $\mathbf{U}_{k,j-1} = 1$. We achieve this by coupling $\mathbf{W}$ and $\mathbf{U}$ using constraints. Then, the optimization problem becomes:

$$\arg\max_{\mathbf{U},\mathbf{W}} \sum_{i,j} \mathbf{U}_{ij}\mathbf{s}_{i\mapsto j} + \sum_{i,j,k} \mathbf{W}_{ijk}\mathbf{s}_{k\curvearrowright i} \tag{4.6}$$

$$\text{subject to } \sum_{i} \mathbf{U}_{ij} = 1 \qquad \forall j \tag{4.6a}$$

$$\sum_{j} \mathbf{U}_{ij} = 1 \qquad \forall i \tag{4.6b}$$

$$\sum_{k} \mathbf{W}_{ijk} = \mathbf{U}_{ij} \qquad \forall j > 1, i \tag{4.6c}$$

$$\sum_{i} \mathbf{W}_{ijk} = \mathbf{U}_{k(j-1)} \; \forall j > 1, k \tag{4.6d}$$

$$\mathbf{U}, \mathbf{W} \geq 0 \tag{4.6e}$$

Finally, in combination with the linear objective, the argmax operation still causes the solution $\mathbf{U}^*(\mathbf{s})$ of Eq. (4.6) as a function of s to have no useful derivatives. This is because an infinitesimal change in s has no effect on the solution $\mathbf{U}^*(\mathbf{s})$ for almost all s. This is a consequence of the fundamental theorem of linear programming, which implies that a solution is attained at a vertex of the feasible region.

To address this, we add an entropy regularization term $\tau(H(\mathbf{U}) + H(\mathbf{W}))$ to the objective Eq. (4.6), where $H(\mathbf{U}) = -\sum_{ij} \mathbf{U}_{ij}(\log \mathbf{U}_{ij} - 1)$, and $\tau > 0$ determines the strength of the regularization. The entropy regularization 'smooths' the solution $\mathbf{U}^*(\mathbf{s})$ in an analogous way to softmax being a smoothed version of argmax. The parameter τ behaves analogously to the softmax temperature: the smaller τ, the sharper $\mathbf{U}^*(\mathbf{s})$ will be. We discuss how to solve the regularized linear program in Section 4.4.

Predicting permutations At test time, we want to find the highest scoring permutation, i.e. we want to solve Eq. (4.5). We approximate this by using $\mathbf{U}^*(\mathbf{s})$ instead, the solution to the entropy regularized version of Eq. (4.6). Despite using a low temperature τ, there is no guarantee that $\mathbf{U}^*(\mathbf{s})$ can be trivially rounded to a permutation matrix. Therefore, we solve the linear assignment problem with $\mathbf{U}^*(\mathbf{s})$ as scores using

the Hungarian Algorithm (Kuhn, 1955). The linear assignment problem asks for the permutation matrix $\mathbf{V}$ that maximizes $\sum_{ij} \mathbf{V}_{ij}\mathbf{U}^*(\mathbf{s})_{ij}$.

Parametrization

We now describe how we parameterize the scores s to permute the tokens into the right order. We first encode the original input utterance x like in Eq. (4.2) to obtain a hidden representation $\mathbf{h}_i$ for input token x_i; we do not share parameters with the multiset tagging. Let a be the function that maps $a(i) \mapsto j$ if the token in position i in z' came from the multiset that was generated by token x_j. For example, in Fig. 4.1, $a(6) = 3$ since `sleep` was predicted from input token *slept*. We then define the hidden representation $\mathbf{h}'_i$ as the concatenation of $\mathbf{h}_{a(i)}$ and an embedding of z'_i:

$$\mathbf{h}'_i = \left[\mathbf{h}_{a(i)}; \text{EMBED}(z'_i)\right] \tag{4.7}$$

Note that tokens that appear more than once in the same multiset have the same representation $\mathbf{h}'_i$. In order to distinguish them, we concatenate another embedding to $\mathbf{h}'_i$: if the token z'_i is the k-th instance of its type in its multiset, we concatenate an embedding for k to $\mathbf{h}'_i$. For example, in Fig. 4.1, $z'_5 =$ `'x1'` and it is the second instance of `'x1'` in its multiset, so we use the embedding for 2.
We parameterize the scores for starting the output with token i as

$$\mathbf{s}_{i \mapsto 1} = \mathbf{w}_{\text{start}}^T \text{FF}_{\text{start}}(\mathbf{h}'_i)$$

and analogously for ending it with token i:

$$\mathbf{s}_{i \mapsto n} = \mathbf{w}_{\text{end}}^T \text{FF}_{\text{end}}(\mathbf{h}'_i)$$

We set $\mathbf{s}_{i \mapsto j} = 0$ for all other i, j.

We parameterize the jump scores $\mathbf{s}_{k \curvearrowright i}$ using Geometric Attention (Csordás et al., 2022) from $\mathbf{h}'_k$ to $\mathbf{h}'_i$. Intuitively, Geometric Attention favours selecting the 'matching' element $\mathbf{h}'_i$ that is closest to $\mathbf{h}'_k$ in terms of distance $|i-k|$ in the string. We refer to Csordás et al. (2022) for details.

Learning Permutations

We now turn to training the permutation model. At training time, we have access to the gold output $\mathbf{y}$ and a sequence $\mathbf{z}'$ from the output of the multiset tagging (see the end of Section 4.2). We note that whenever $\mathbf{y}$ (or $\mathbf{z}'$) contains one vocabulary item at least twice, there are multiple permutations that can be applied to $\mathbf{z}'$ to yield $\mathbf{y}$. Many of these permutations will give the right result for the wrong reasons and the permutation that is desirable for generalization is latent. For example, consider Fig. 4.2. The token `agent` is followed by the entity who performs the action, whereas `theme` is followed by the one affected by the action. The permutation indicated by dashed arrows generalizes poorly to a sentence like *Emma knew that James slid* since *slid* will introduce a `theme` role rather than an `agent` role (as the sliding is happening to James). Thus, this permutation would then lead to the incorrect output `know theme`

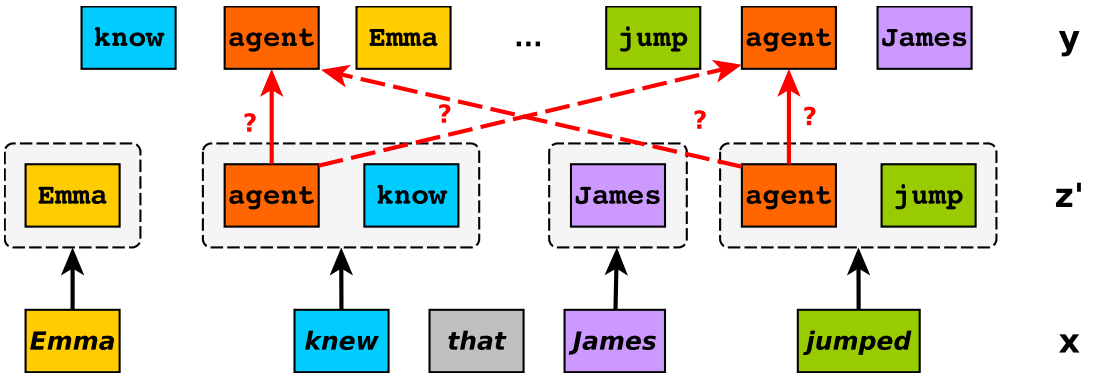


Figure 4.2: Two possible permutations (represented with red arrows) of z' that yield the same result y. Only the permutation marked with solid arrows corresponds to the right linguistic relation that will generalize well.

`Emma ... slide agent James`, in which Emma is affected by the knowing event and James is the one who slides something.

In order to train the permutation model in this setting, we use a method that is similar to EM for structured prediction.[1] During training, the model output $\mathbf{U}^*(s)$ and $\mathbf{W}^*(s)$ often represents a soft permutation that does *not* permute $\mathbf{z}'$ into $\mathbf{y}$. Our goal is to push the model output into the space of (soft) permutations that lead to $\mathbf{y}$. More formally, let $\mathbf{Q} \in Q(\mathbf{y}, \mathbf{z}')$ be the set of bistochastic matrices such that $\mathbf{Q}_{i,j} = 0$ iff $\mathbf{z}'_i \neq \mathbf{y}_j$. That is, any permutation included in $Q(\mathbf{y}, \mathbf{z}')$ leads to the gold output $\mathbf{y}$ when applied to $\mathbf{z}'$.

First, we project the current prediction $\mathbf{U}^*(s)$ and $\mathbf{W}^*(s)$ into $Q(\mathbf{y}, \mathbf{z}')$ using the KL divergence as a measure of distance (E-step):

$$\hat{\mathbf{U}}, \hat{\mathbf{W}} = \underset{\mathbf{U} \in Q(\mathbf{y}, \mathbf{z}'), \mathbf{W}}{\arg\min} \; \mathrm{KL}(\mathbf{U} \,||\, \mathbf{U}^*(s)) + \mathrm{KL}(\mathbf{W} \,||\, \mathbf{W}^*(s)) \tag{4.8}$$

subject to Eq. (4.6a) to Eq. (4.6e)

Similar to the M-step of EM, we then treat $\hat{\mathbf{U}}$ and $\hat{\mathbf{W}}$ as soft gold labels and train our model to minimize the KL divergence between labels and the predicted soft permutation:

$$\mathrm{KL}(\hat{\mathbf{U}} \,||\, \mathbf{U}^*(s)) + \mathrm{KL}(\hat{\mathbf{W}} \,||\, \mathbf{W}^*(s))$$

As usual with neural networks, we minimize this objective with gradient descent wrt the model parameters.

In order to implement this training procedure, we need to be able to compute the E-step, i.e. solve Eq. (4.8). Fortunately, this optimization problem can be framed as an optimization problem that looks much like the entropy-regularized version of Eq. (4.6) as we will show in the next section with Eq. (4.10). Hence, we will be able to use the same algorithm from the next section for both optimization problems. There is one difference of note, however, between the E-step (Eq. (4.8)) and Eq. (4.6). Our E-step has

[1] It can also be derived as a lower bound to the marginal log-likelihood $\log P_\theta(\mathbf{y}|\mathbf{x}, \mathbf{z}')$ (see Appendix C.1.3), and be seen as a form of posterior regularization (Ganchev et al., 2010)

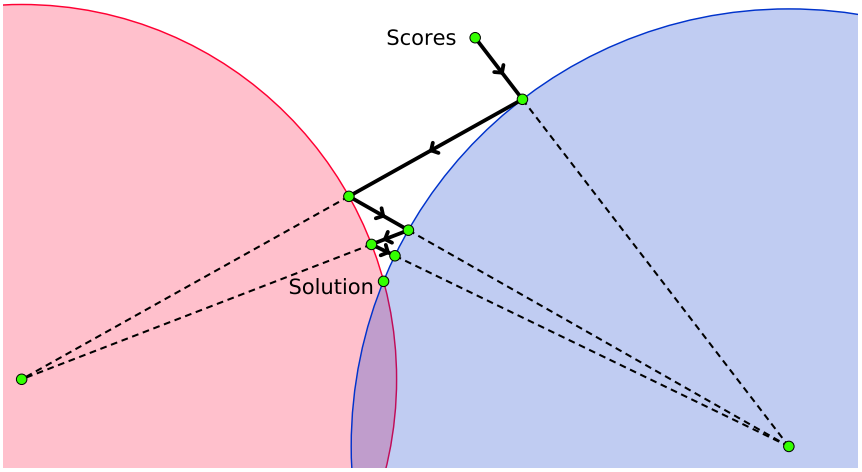


Figure 4.3: Geometric intuition of Bregman's method of alternating projections with two constraints C_0 (blue), C_1 (red). The solution is the point within the feasible region ($C_0 \cap C_1$) that is closest (e.g. in the sense of KL-divergence) to the scores. The algorithm starts with the scores and alternatingly projects onto the two constraints until convergence.

an additional constraint on the feasible region, namely $\mathbf{U} \in Q(\mathbf{y}, \mathbf{z}')$. This constraint rules out certain correspondences. We can approximately enforce this constraint by applying a mask to $\mathbf{U}^*(s)$ such that any forbidden correspondence receives a negative score with high magnitude. As a consequence of this masking, the solution $\hat{\mathbf{U}}$ has a value close to 0 for such forbidden correspondences.

4.4 Inference for Relaxed Permutations

Now we describe how to solve the entropy regularized form of Eq. (4.6) and how to backpropagate through it. This section may be skipped on the first reading as it is not required to understand the experiments; we note that the resulting algorithm (Algorithm 1) is conceptually relatively simple. Before describing our method, we explain the general principle.

Bregman's Method

Bregman's method (Bregman, 1967) is a method for constrained convex optimization. In particular, it can be used to solve problems of the form

$$\mathbf{x}^* = \underset{\mathbf{x} \in C_0 \cap C_1 \ldots \cap C_n, \mathbf{x} \geq \mathbf{0}}{\arg\max} \; \mathbf{s}^T \mathbf{x} + \underbrace{\tau H(\mathbf{x})}_{\text{Regularizer}} \tag{4.9}$$

where $C_0, \ldots, C_n$ are linear equality constraints, $H(\mathbf{x}) = -\sum_i \mathbf{x}_i (\log \mathbf{x}_i - 1)$ is a form of entropy regularization, and τ determines the strength of the regularization. Note that our parameterization of permutations (Eq. (4.6)) has this form.

It is easier to explain Bregman's method if we look at this optimization problem from an equivalent geometric perspective. We can also think of Eq. (4.9) as a projec-

tion onto the set $C = C_0 \cap C_1 \ldots \cap C_n$ using a generalized KL-divergence $\mathrm{KL}(\mathbf{x} \mid \mathbf{y}) = \sum_i \mathbf{x}_i(\log \frac{\mathbf{x}_i}{\mathbf{y}_i} - 1)$ as a measure of distance.[2]

$$\begin{aligned} &\arg\max_{\mathbf{x} \in C} \mathbf{s}^T \mathbf{x} - \tau \sum_i \mathbf{x}_i(\log \mathbf{x}_i - 1) \\ =&\arg\min_{\mathbf{x} \in C} \tau \sum_i \mathbf{x}_i(\log \mathbf{x}_i - \frac{\mathbf{s}_i}{\tau} - 1) \\ =&\arg\min_{\mathbf{x} \in C} \mathrm{KL}(\mathbf{x} \mid\mid \exp(\frac{\mathbf{s}}{\tau})) \end{aligned} \tag{4.10}$$

Bregman's method is a simple iterative process for solving such optimization problems, and takes a divide-and-conquer approach. Instead of projecting onto all constraints $C = C_0 \cap C_1 \ldots \cap C_n$, it projects onto one constraint at a time, analogous to the method of Projections onto Convex Sets (POCS, von Neumann (1951); Bauschke and Borwein (1996)). We start with the scores $\mathbf{s}$ and then cyclically iterate over the constraints and project the current estimate $\mathbf{x}^i$ onto the chosen constraint until convergence (see Fig. 4.3 for an illustration):

$$\begin{aligned} \mathbf{x}^0 &= \exp \frac{\mathbf{s}}{\tau} \\ \mathbf{x}^{i+1} &= \operatorname*{arg\,min}_{\mathbf{x} \in C_{i \bmod (n+1)}} \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{x}^i) \end{aligned} \tag{4.11}$$

In order to apply Bregman's method, we need to be able to compute the KL projection $\arg\min_{\mathbf{x} \in C_i} \mathrm{KL}(\mathbf{x} \mid \mathbf{x}^i)$ for *all* $C_0, \ldots, C_n$ in closed-form. We discuss how to do this for Eq. (4.6) in the next section.

As an example for Bregman's method, consider a problem of the form Eq. (4.9) with a single linear constraint $C^\Delta = \{\mathbf{x} \mid \sum_i \mathbf{x}_i = 1\}$. In this case, Bregman's method coincides with the softmax function and only requires a single iteration. This is because the KL projection $\mathbf{x}^* = \arg\min_{\mathbf{x} \in C^\Delta} \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{y})$ for $\mathbf{y} > \mathbf{0}$ has the closed-form solution $\mathbf{x}^*_i = \frac{\mathbf{y}_i}{\sum_i \mathbf{y}_i}$. Sinkhorn's algorithm (Sinkhorn, 1964) of alternatingly normalizing rows and columns of a matrix is another example of Bregman's method.

If we have a closed-form expression for a KL projection (such as normalizing a vector), we can use automatic differentiation to backpropagate through it. To backpropagate through the entire solver, we apply automatic differentiation to the composition of all projection steps.

Bregman's Method for Eq. (4.6)

In order to apply Bregman's method to solve the entropy regularized version of Eq. (4.6), we need to decompose the constraints into sets which we can efficiently project onto. We choose the following three sets here: (i), containing Eq. (4.6a) and Eq. (4.6c), and (ii), containing Eq. (4.6a) and Eq. (4.6d), and finally (iii), containing only Eq. (4.6b). We now need to establish what the KL projections are for our chosen sets. For (iii), the projection is simple:

[2]Technically, the generalized KL divergence additionally has the term $\sum_i \mathbf{y}_i$. We are justified in leaving this out since we compute the minimizer wrt $\mathbf{x}$, which renders this additional term constant.

Algorithm 1 Bregman's method for Eq. (4.6) with entropic regularization

function BREGMAN(s, τ)
  $\mathbf{U}_{ij} = \exp(\tau^{-1} s_{i \mapsto j})$
  $\mathbf{W}_{ijk} = \exp(\tau^{-1} \mathbf{s}_{k \curvearrowright i})$
  while within budget and not converged **do**
    $\mathbf{U}, \mathbf{W}$ = KL project($\mathbf{U}, \mathbf{W}$; 4.6a,4.6c) with Prop. 2
    $\mathbf{U}, \mathbf{W}$ = KL project($\mathbf{U}, \mathbf{W}$; 4.6a,4.6d) with Prop. 2
    $\mathbf{U}$ = KL project ($\mathbf{U}$, 4.6b) with Prop. 1
  return $\mathbf{U}, \mathbf{W}$

Proposition 1. (Benamou et al. (2015), Prop. 1) For $\mathbf{A}, \mathbf{m} > 0$, the KL projection $\arg\min_{\mathbf{U}} \mathrm{KL}(\mathbf{U} \mid\mid \mathbf{A})$ subject to $\sum_j \mathbf{U}_{ij} = \mathbf{m}_i$ is given by $\mathbf{U}^*_{ij} = \mathbf{m}_i \frac{\mathbf{A}_{ij}}{\sum_{j'} \mathbf{A}_{ij'}}$.

Let us now turn to (i) and (ii). The constraints Eq. (4.6d) and Eq. (4.6c) are structurally essentially the same, meaning that we can project onto (ii) in basically the same manner as onto (i). We project onto (i), with the following proposition:

Proposition 2. For $\mathbf{A}, \mathbf{B} > \mathbf{0}$, the KL projection

$$\begin{aligned} \underset{\mathbf{U},\mathbf{W}}{\arg\min} \quad & \mathrm{KL}(\mathbf{U} \mid\mid \mathbf{A}) + \mathrm{KL}(\mathbf{W} \mid\mid \mathbf{B}) \\ \text{subject to} \quad & \sum_i \mathbf{U}_{ij} = 1 \quad \forall j \\ & \sum_k \mathbf{W}_{ijk} = \mathbf{U}_{ij} \ \forall j, i \end{aligned} \tag{4.12}$$

is given by:

$$\mathbf{T}_{ij} = \sqrt{\mathbf{A}_{ij} \sum_k \mathbf{B}_{ijk}}, \qquad \mathbf{U}^*_{ij} = \frac{\mathbf{T}_{ij}}{\sum_{i'} \mathbf{T}_{i'j}}, \qquad \mathbf{W}^*_{ijk} = \mathbf{U}^*_{ij} \frac{\mathbf{B}_{ijk}}{\sum_{k'} \mathbf{B}_{ijk'}}$$

The proof can be found in Appendix C.1.2.

We note that the algorithm is easy to implement for GPUs, so that projections can be done in parallel for all i and j, and that batching is trivial. In practice, we implement all projections in log space for numerical stability. We visually demonstrate how the algorithm operates for a simple example in Fig. 4.4.

4.5 Evaluation

Doubling Task

Our permutation model is very expressive and is not limited to synchronous context-free languages. This is in contrast to the formalisms that other approaches rely on, such as Wang et al. (2021a). To evaluate if our model can structurally generalize beyond the synchronous context-free languages in practice, we consider the function

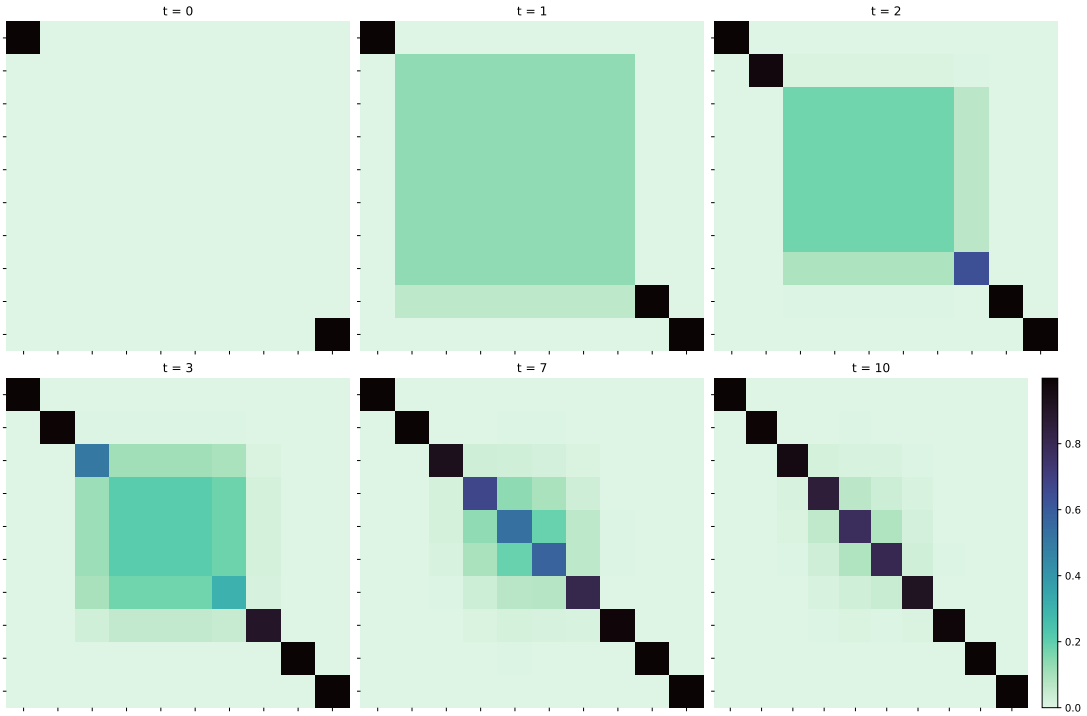


Figure 4.4: Development of intermediate values of $\mathbf{U}$ over the course of Bregman's method (Algorithm 1). Each time step corresponds to one pass through the loop. In this example, the input to the algorithm are scores $s_{i \mapsto j}$ that map the first element in the input to the first element in the output, and the last element in the input to the last element in the output (see $t = 0$). High scores are assigned to jumps $\mathbf{s}_{k \curvearrowright k+1}$ from each position k to its right neighbor $k+1$. In each iteration, the algorithm propagates the information from the positions that already have a clear assignment, starting from both ends of the sequence.

$F = \{(w, ww) \mid w \in \Sigma^*\}$. This function is related to processing challenging natural language phenomena such as reduplication and cross-serial dependencies. We compare our model with an LSTM-based seq2seq model with attention and a Transformer in the style of Csordás et al. (2021) that uses a relative positional encoding. Since the input is a sequence of symbols rather than English, we replace RoBERTa with a bidirectional LSTM and use randomly initialized embeddings. The models are trained on inputs of lengths 5 to 10 and evaluated on longer examples. The results can be found in Fig. 4.5. All models get perfect or close to perfect accuracy on inputs of length 11, but accuracy quickly deteriorates for the LSTM and the Transformer. In contrast, our model extrapolates very well to longer sequences.

COGS

COGS is a synthetic benchmark for compositional generalization introduced by Kim and Linzen (2020). Models are tested for 21 different cases of generalization, 18 of which focus on using a lexical item in new contexts (**Lex**). There are 1000 instances per generalization case. Seq2seq models struggle in particular with the *structural* gen-

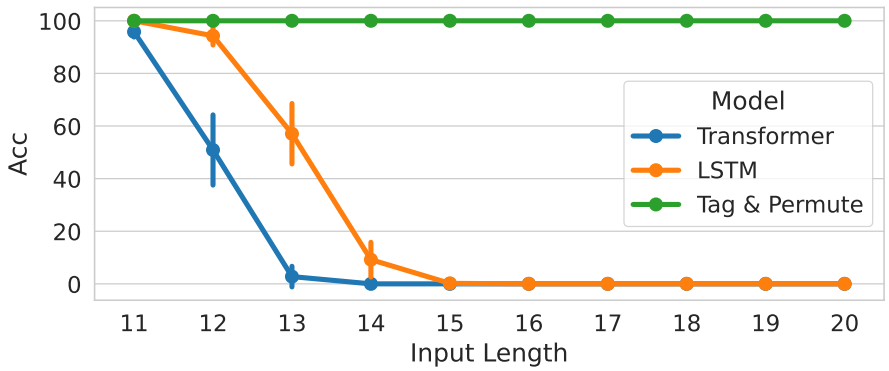


Figure 4.5: Accuracy by input length for doubling task.

eralization tasks (Yao and Koller, 2022), and we focus on those: (i) generalization to deeper **PP** recursion than seen during training ("Emma saw a hedgehog on a chair in the garden beside the tree..."), (ii) deeper **CP** recursion ("Olivia mentioned that James saw that Emma admired that the dog slept"), and (iii) PPs modifying subjects when PPs modified only objects in the training data (**Obj PP** $\rightarrow$ **Subj PP**).

We follow previous work and use a lexicon (Akyürek and Andreas, 2021) to map some input tokens to output tokens (see Appendix C.2 for details). We also use this mechanism to handle the variable symbols in COGS.

We report the means and standard deviations for 10 random seeds in Table 4.1. Our approach obtains high accuracy on CP and PP recursion but exact match accuracy is low for Obj PP $\rightarrow$ Subj PP. This is in part because our model sometimes predicts semantic representations for Obj PP $\rightarrow$ Subj PP that are equivalent to the gold standard but use a different order for the conjuncts. Therefore, we report accuracy that accounts for this in Table 4.2. In both tables, we also report the impact of using a simple copy mechanism instead of the more complex lexicon induction mechanism (-Lex). Our model outperforms all other non-tree-based models by a considerable margin.

Structural generalization without trees All previous methods that obtain high accuracy on recursion generalization on COGS use trees. Some approaches directly predict a tree over the input (Liu et al., 2021a; Weißenhorn et al., 2022), while others use derivations from a grammar for data augmentation (Qiu et al., 2022a) or decompose the input along a task-specific parse tree (Drozdov et al., 2022). Our results show that trees are not as important for compositional generalization as their success in the literature may suggest, and that weaker structural assumptions already reap some of the benefits.

Logical forms with variables COGS uses logical forms with variables, which were removed in conversion to variable-free formats for evaluation of some approaches (Zheng and Lapata, 2021; Qiu et al., 2022a; Drozdov et al., 2022). Wu et al. (2023b) have argued for keeping the variable symbols because they are important for some semantic distinctions; we keep the variables.

Model	Obj PP → Subj PP	CP	PP	Lex	Total
Qiu et al. (2022a)[◇♣]	-	-	-	-	**99**
Drozdov et al. (2022)[◇♣]	-	-	-	-	**99**
Akyürek and Andreas (2021)	0	0	1	96	82
Zheng and Lapata (2021)	0	14	14	98	85
Zheng and Lapata (2021)[◇]	0	25	35	**99**	88
T5-base[◇]	0	13	18	**99**	86
Tag & Permute [◇]	**9**$_{\pm 13}$	**79**$_{\pm 11}$	**85**$_{\pm 14}$	97$_{\pm 2}$	92$_{\pm 2}$
Tag & Permute [◇]- Lex	5$_{\pm 6}$	71$_{\pm 26}$	82$_{\pm 20}$	79$_{\pm 1}$	75$_{\pm 2}$

Table 4.1: Exact match accuracy on **COGS** by generalization type. Tag & Permute is the approach proposed here. [◇] refers to models using **pretrained** Transformers, and [♣] refers to models implicitly or explicitly using **trees**. The results with T5 were obtained by Zheng and Lapata (2021).

Model	Obj PP → Subj PP	CP	PP	Lex	Total
Lear[♣]	**93**	**100**	**99**	**99**	**99**
AM parser[◇♣]	72	**100**	97	76	78
Dangle	8	14	14	**99**	87
Tag & Permute [◇]	33$_{\pm 24}$	82$_{\pm 11}$	91$_{\pm 5}$	97$_{\pm 2}$	93$_{\pm 2}$
Tag & Permute [◇]- Lex	35$_{\pm 22}$	73$_{\pm 27}$	92$_{\pm 5}$	79$_{\pm 1}$	77$_{\pm 2}$

Table 4.2: Accuracy on **COGS** when the order of conjuncts is disregarded or established in post-processing. For Lear (Liu et al., 2021a) and the AM parser (Groschwitz et al., 2018; Weißenhorn et al., 2022), we report numbers of Weißenhorn et al. (2022).

ATIS

While COGS is a good benchmark for compositional generalization, the data is synthetic and does not contain many phenomena that are frequent in semantic parsing on real data, such as paraphrases that map to the same logical form. ATIS (Dahl et al., 1994) is a realistic English semantic parsing dataset with executable logical forms. We follow the setup of Chapter 3 and use the variable-free FunQL representation (Guo et al., 2020). Apart from the usual iid split, we evaluate on a length split, where a model is trained on examples with few conjuncts and has to generalize to longer logical forms with more conjuncts. For a fair comparison with previous work, we do not use the lexicon/copying. We also evaluate a version of our model without RoBERTa that uses a bidirectional LSTM and GloVe embeddings instead. This mirrors the model of Chapter 3.

Table 4.3 shows mean accuracy and standard deviations over 5 runs. Our model is competitive with non-pretrained models in-distribution, and outperforms all other models on the length generalization. The high standard deviation on the length split stems from an outlier run with 18% accuracy – the second worst-performing run

Model	iid	Length
LSTM seq2seq	75.98±1.30	4.95±2.16
Transformer	75.76±1.43	1.15±1.41
BART-base$^{\diamond}$	**86.96**±1.26	19.03±4.57
F→R ♣	68.26±1.53	29.91±2.91
F→R † ♣	74.15±1.35	35.41±4.09
Tag & Permute $^{\diamond}$	76.65±1.67	**41.39**±13.47
Tag & Permute	73.93±1.43	38.79±7.11

Table 4.3: Accuracy on **ATIS**. † indicates grammar-based decoding. F→R is the model from Chapter 3.

Model	Calendar	Doc	Email
BART-base$^{\diamond}$	36.7±3.0	0.6±0.3	20.5±9.8
F→R ♣	57.2±19.9	36.1±5.6	43.9±3.8
F→R † ♣	69.5±13.9	42.4±5.7	55.6±2.7
Tag & Permute $^{\diamond}$	**74.3**±3.5	**57.8**±5.5	**60.6**±4.8
Tag & Permute	65.6±2.8	41.4±4.9	47.6±4.5

Table 4.4: Accuracy on length splits by domain on **Okapi**.

achieved an accuracy of 44%. Even without pretraining, our model performs very well. In particular, without grammar-based decoding our model performs on par or outperforms F→R *with* grammar-based decoding.

The run time of F→R is dominated by the permutation model and it takes up to 12 hours to train on ATIS. Training the model presented here only takes around 2 hours for both stages.

Okapi

Finally, we consider the Okapi (Hosseini et al., 2021) semantic parsing dataset, in which an English utterance from one of three domains (Calendar, Document, Email) has to be mapped to an API request. We again follow the setup from Chapter 3 and evaluate on the corresponding length split, where a model has to generalize to longer logical forms. In contrast to all other datasets we consider, Okapi is quite noisy because it was collected with crowd workers. This presents a realistic additional challenge on top of the challenge of structural generalization.

The results of 5 runs can be found in Table 4.4. Our model outperforms both BART (Lewis et al., 2020a) and F→R. In the comparison without pretraining, our model also consistently achieves higher accuracy than the comparable version of F→R without grammar-based decoding.

In Fig. 4.6 we show the accuracy of our model on the document domain in comparison with previous work by number of conjuncts in the logical form. All models

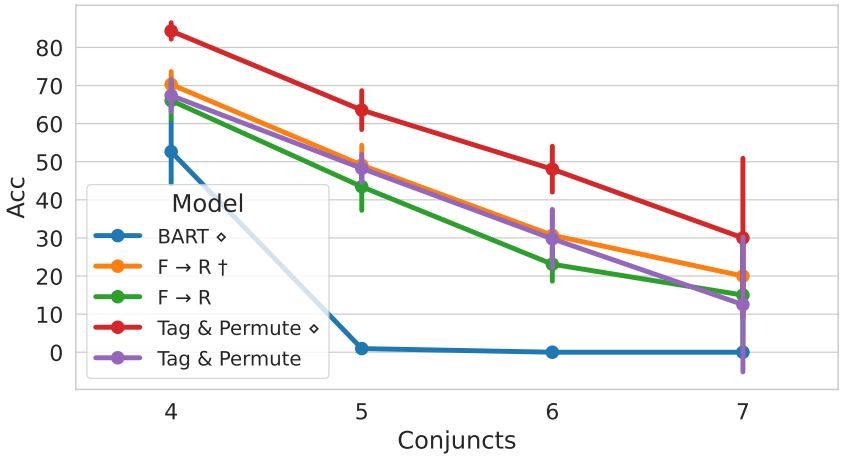


Figure 4.6: Accuracy on the document domain of **Okapi** by number of conjuncts in the gold logical form.

become worse with higher number of conjuncts, as expected due to the increasing complexity of the questions. Tag & Permute with pretraining outperforms all other models by a comfortable margin. Without pretraining and grammar-based decoding, Tag & Permute not only achieves very similar aggregate performance to the version of F→R *with* grammar-based decoding but also closely mirrors the accuracy profile across conjuncts.

4.6 Ablations and Error Distribution

Performance breakdown In order for our approach to be accurate, both the multiset tagging model and the permutation model have to be accurate. Table 4.5 explores which model acts as the bottleneck in terms of accuracy on ATIS and COGS. The answer depends on the dataset: for the synthetic COGS dataset, predicting the multisets correctly is easy except for O → S, and the model struggles more with getting the permutation right. In contrast, for ATIS, the vast majority of errors can be attributed to the first stage.

Permutation baseline A simpler approach for predicting a permutation of the output z' from the multiset tagging is to use a seq2seq model. In order to compare our approach to such a baseline, we concatenate the original input x with a separator token and z'. We then feed this as input to a BART-base model which is trained to predict the output sequence y. At inference time, we use beam search and enforce the output to be a permutation of the input. As detailed in Table 4.6, this approach works well in-distribution and it also shows a small improvement over finetuning BART directly on the length split of ATIS. However, it does not perform as well as our approach. On COGS, our model outperforms the permutation baseline by an even bigger margin. Unseen variable symbols could be a challenge for BART on COGS which might explain part of the gap in performance.

This approach towards predicting permutations is similar to that of Cazzaro et al.

		Freq	Seq	Seq/Freq
COGS	O → S	50±34	9±13	11±15
	CP	97±5	79±11	82±13
	PP	99±0	85±14	85±15
ATIS	iid	77.6±1.4	76.7±1.7	98.7±0.5
	Length	42.2±13.6	41.4±13.5	97.8±0.8

Table 4.5: Performance breakdown of the first and second stage. 'Freq' refers to accuracy measured on the predicted multiset; it measures performance of the first stage. 'Seq' measures the accuracy of both stages. 'Seq/Freq' is the percentage of correct predictions given that the multiset is predicted correctly.

		BART perm	Ours
COGS	O → S	16±8	33±24
	CP	17±3	82±11
	PP	16±5	91±5
ATIS	iid	77.1±1.6	76.6±1.7
	Length	23.8±9.4	41.4±13.5

Table 4.6: Performance of our permutation model in comparison to using BART for predicting the permutation. We ignore the order of conjuncts for evaluation.

(2023) except that they do not constrain the beam search to permutations. We found that not constraining the output to be a permutation worked worse in the compositional generalization setups we consider here.

4.7 Related Work

Tagging and Reordering As reviewed in Section 3.6, Cazzaro et al. (2023) propose a conceptually similar approach that first monotonically translates the input and then reorders the translation. They phrase each of those steps with standard sequence-to-sequence models or as tagging, i.e. each input token is tagged with a single output token or a special null token based on its contextualized representation. Similar to them, we train both stages separately in this chapter. While they rely on external alignments to construct training data for the first stage, we induce correspondences during training. In contrast to our approach, theirs does not explicitly model that there are in general multiple possible permutations which generate the correct output, but that they might do so for the wrong reason (see Fig. 4.2). For the benchmarks we evaluated on, we found that our permutation approach performed better than a version that uses BART for reordering with constrained decoding, which is closer to the approach of Cazzaro et al. (2023).

Comparison with Structured Reordering The permutation approach we propose in this chapter differs in several important aspects from the reordering approach with permutation trees of Wang et al. (2021a) that we use in Chapter 3.

The most apparent difference is the expressivity: the reordering with permutation trees admits many permutations of interest, and also excludes arrangements that are unattested in natural language (Stanojević and Steedman, 2021). Nevertheless, when dealing with semantic parsing, we have encountered situations where separable permutations are not expressive enough.[3] The reordering approach proposed in this chapter addresses this and can represent and predict any permutation. This comes at the price of not having a hierarchical inductive bias and making the inference problem NP-hard. However, our experiments show that this method performs well for structural generalization in practice. Despite the high computational complexity in the worst case, we found our permutation model to be faster and more memory-efficient in practice. This is because the structured reordering approach requires $O(n^5)$ memory and compute since it constructs a matrix of size $O(n^2)$ in the innermost loop of a CYK-style algorithm, which goes through $O(n^3)$ iterations.

Another difference is that the structured reordering approach reorders a sequence that already has an order imposed on it. In contrast, Algorithm 1 is equivariant to permutations of the scores, i.e. permuting the scores and applying Algorithm 1 gives the same result as applying the algorithm and then permuting the result. This makes our approach suitable for arranging sets or partially-ordered sets and leaves it to the modeler to take order into account for the parameterization of the jumps. The parameterization in this chapter takes the original into account to capture the different syntactic relations that words are in.

There is also a more subtle difference between the two approaches in terms of how the parameterization affects the distribution over permutations: In the structured reordering approach, for certain permutations, there are multiple trees that describe them, but other permutations can be described only by a single tree. This comes from the associativity of the concatenation (and of the reverse concatenation). For instance, any permutation tree that only uses concatenation results in the identity permutation. That is, for a sentence of n words, there are as many trees that will result in the identity permutation as there are binary trees with n leaves, which grows rapidly with n.

Just after the model is initialized, we have a roughly uniform distribution over permutation trees. The many-to-one relation between permutation trees and permutations results in a heavily skewed distribution over permutations and favors the identity permutation and a complete reversal. In principle, this is easy to address with a normal form that introduces a convention to represent (reverse) concatenation of three elements with a single tree rather than one of two possible trees (Shapiro and Stephens, 1991; Stanojević and Steedman, 2021). For the algorithm of Wang et al. (2021a), however, this would require additional book-keeping and would increase compute and memory usage by a factor of two. This is in contrast to the permutation approach presented in this chapter: Since there is exactly one sequence of jumps for every permutation, a uniform distribution over scores translates to uniform preference over permutations.

[3] This is *not* in conflict with Stanojević and Steedman (2021) because we consider the relationship between sentences and logical forms rather than sentences in different natural languages.

Predicting Permutations without Trees Mena et al. (2018) and Lyu and Titov (2018) use variational autoencoders based on the Sinkhorn algorithm to learn latent permutations. The Sinkhorn algorithm (Sinkhorn, 1964) is also an instance of Bregman's method and solves the entropy regularized version of Eq. (4.6) without the **W**-term. This parameterization is considerably weaker than ours since it cannot capture our notion of 'jumps', i.e. bilexical relationships between the multiset tokens.

Devatine et al. (2022) investigate sentence reordering methods. They use bigram scores, which results in a similar computational problem to ours. However, they deal with it by restricting the space of possible permutations to enable tractable dynamic programs with run time from $O(n^6)$ in the least restricted case down to $O(n)$ in the most restricted case. Eisner and Tromble (2006) propose local search methods for decoding permutations for machine translation.

Outside of NLP and Machine Learning, Kushinsky et al. (2019) have applied Bregman's method to the quadratic assignment problem, which Eq. (4.5) is a special case of. Since they solve a more general problem, using their approach for Eq. (4.6) would require $O(n^4)$ rather than $O(n^3)$ variables in the linear program.

Tree-based approaches Despite not using trees in our model, we briefly highlight two parallels to tree-based approaches, such as that of Groschwitz et al. (2018). The two stages in our approach loosely correspond to two stages in lexicalized frameworks such as CCG (Steedman, 2000) and the AM-algebra (Groschwitz et al., 2018). These approaches first apply supertagging (Bangalore and Joshi, 1999; Clark and Curran, 2007; Lewis and Steedman, 2014), which tags each token with the semantic contribution it makes, along with a formal characterization of its syntactic properties in the case of CCG. This corresponds to our multiset tagging, which produces output tokens that correspond to one input token. In contrast to CCG, our multiset tagging does not include information about syntactic properties.

In these tree-based approaches, supertagging is followed by a parsing step that predicts a tree with the supertags at the leaves. Our permutation model predicts a sequence of jumps, which does not form a tree. However, one might conjecture that the semantic contributions of two tokens that are in a syntactic relation, e.g. a subject and its verb, have a tendency to be adjacent in logical forms, such as in Fig. 4.1 with the adjacent variables x_2 (contributed by the verb 'sleep') and x_1 (contributed by 'the girl', the subject of 'sleep'). In such situations, the permutation model predicts a jump that coincides with a syntactic dependency.

Structural Inductive Biases without Trees Shortly after publication of the paper underlying this chapter, Petit et al. (2023) also find that a model without an inductive bias towards trees can handle deeper recursion, at least for the COGS benchmark (Kim and Linzen, 2020). Their model predicts a semantic graph (rather than a sequence) and draws its inductive bias in part from valency constraints, e.g. an AGENT edge can only exist between a word (or supertag, more precisely) that requires an outgoing AGENT edge and one that requires a corresponding *incoming* AGENT edge. This approach achieves perfect recursion generalization, but valency constraints are difficult to apply to NLP tasks with sequential outputs and beyond syntactic and semantic parsing.

4.8 Conclusion

In this chapter, we have presented a flexible new sequence-to-sequence model for semantic parsing. Our approach relies on a similar conceptualization as Chapter 3 and consists of two steps: We first tag each input token with a multiset of output tokens. Then we arrange those tokens into a sequence using a permutation model. We introduce a new method to predict and learn permutations based on a regularized linear program that does not restrict what permutations can be learned. This makes the approach more expressive than the model in Chapter 3 and also proves efficient enough to scale to larger datasets with longer sequences. The model we present has a strong ability for structural generalization on synthetic and natural semantic parsing datasets. We show for the first time that a structural inductive bias that is not based on trees can already achieve strong generalization to deeper recursion than seen during training.

An alternative approach one could have taken in this chapter would be to reuse the architecture with the fertility layer from Chapter 3, replace the structured reordering model with the new permutation layer presented in this chapter. Then the model could be trained end-to-end. We initially experimented with this but found the resulting model was quite hard to optimize and got stuck in local minima, where the fertility layer had converged to a sub-optimal solution, and the permutation layer could not correct this. This motivated breaking up the training process into multiset tagging and predicting permutations. Taking the approach of separately training the two stages as done in this chapter, but using the permutation model from Chapter 3 would have been difficult because the first stage could learn to predict multisets that cannot be permuted into the correct output with that model's constraints on possible permutations. The approach in this chapter does not face this issue since the permutation model can predict any permutation.

Limitations The conditional independence assumptions of our multiset tagging model are a limitation for the applicability of this model. For example, the independence assumptions are too strong to apply it to natural language generation tasks such as summarization. From a technical point of view, the independence assumptions are important to be able to induce the latent assignment of output tokens to multisets efficiently. Future work may design multiset tagging methods that make fewer independence assumptions.

To backpropagate through the prediction of permutations, we 'unroll' the loop in Algorithm 1 and use standard automatic differentiation. While simple and convenient, this approach needs to keep all intermediate results in memory. Future work could make memory consumption independent of the number of iterations and achieve the optimal requirement of $O(n^3)$ by using implicit differentiation (e.g. Blondel et al., 2022).

While our method for predicting permutations is comparatively fast, processing long sequences, e.g. with more than 100 tokens, remains somewhat slow. Possible future directions to speed this up could include a smarter way to iterate over the constraints, which has been used to increase the speed of the Sinkhorn algorithm (Altschuler et al., 2017). Local search or dual decomposition are also potential candidates for improving inference speed.

Regarding the importance of trees for compositional generalization, our model has no component that could provide an explicit inductive bias towards trees. However, one might counter that the pretrained RoBERTa model that we use as a component likely *implicitly* captures tree-like structures to some extent (Tenney et al., 2019). We cannot exclude this implicit notion of trees having an effect, but this counterargument does not hold as clearly for our experiments with an LSTM-based variant of our model without pretraining, which also performed well in terms of structural generalization.

Part II

Injecting Inductive Biases by Simulation

Chapter 5

Injecting an FST-like Inductive Bias into a Transformer

In the second part of the thesis and starting with this chapter, we propose a general and efficient framework for introducing inductive biases into standard encoder-decoder Transformers (Goal 2). This chapter is based on Lindemann et al. (2024a) and focuses on the inductive biases of Finite State Transducers (FSTs). We apply an analogous method to syntax-sensitive tasks in Chapter 6.

Inductive biases, i.e. the preferences and the knowledge a model brings to the task, enable a model to learn from small amounts of data and generalize systematically outside of the training distribution. While sequence-to-sequence models perform very well on in-distribution data, they usually lack structural inductive biases and consequently struggle with systematic generalization, such as unseen combinations of known substrings (Lake and Baroni, 2018; Keysers et al., 2020), extrapolation to longer inputs (Hupkes et al., 2020), and deeper recursion (Kim and Linzen, 2020).

Integrating structural inductive biases into sequence-to-sequence models is challenging. One popular approach is to develop specialized architectures (Dong and Lapata, 2016; Wu and Cotterell, 2019; Zheng and Lapata, 2021; Kim, 2021, inter alia) as we have done in Part I. However, specialized architectures make it difficult to precisely control and adjust the nature of the inductive bias when demands change, as the architecture would need to be modified and models re-trained. Recently, some works instead have tried to inject inductive biases into sequence-to-sequence models by pretraining on a well-chosen synthetic task (Krishna et al., 2021; Wu et al., 2021b, 2022; Goodale et al., 2025) or meta-learning on a distribution of synthetic tasks (McCoy et al., 2020; McCoy and Griffiths, 2023) using MAML (Finn et al., 2017). Here, the inductive bias can be controlled by the choice of the synthetic task. However, meta-learning with MAML scales poorly because it requires expensive second-order derivatives. While computationally cheaper versions of MAML exist (Finn et al., 2017; Nichol et al., 2018; Bansal et al., 2022), it is unclear if they perform well enough in these setups[1] and standard pretraining can be less effective than MAML (McCoy and Griffiths, 2023).

[1] Anecdotally, we experimented with meta-adapters (Bansal et al., 2022) as a computationally cheaper version of MAML, optionally also using a first-order approximation, but we found the training objective difficult to optimize in our setup.

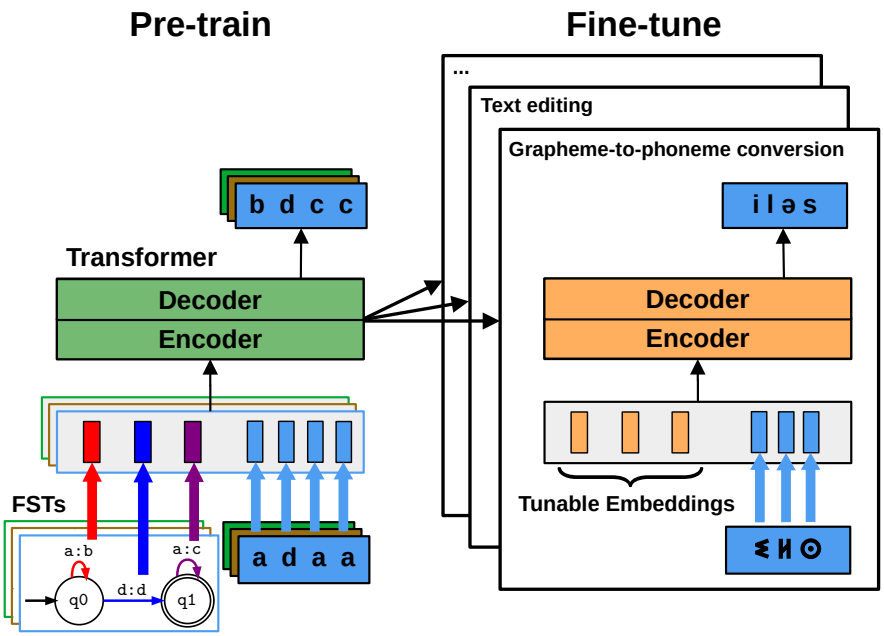


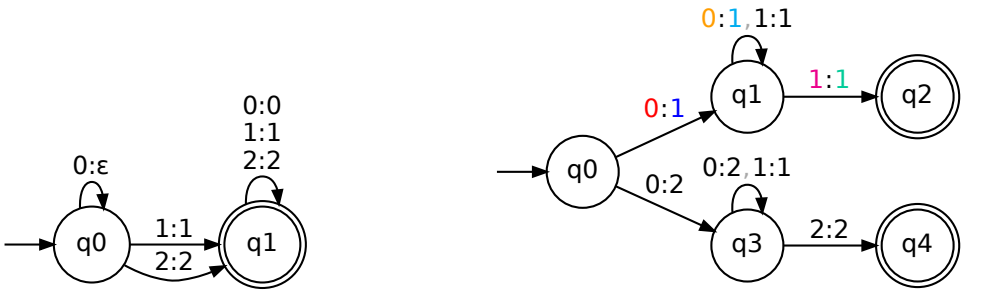
Figure 5.1: Left: Pretraining a Transformer to simulate automatically generated FSTs. Right: fine-tuning the Transformer and the prefix where the FST used to be on a downstream task by using only input/output pairs. Tunable parameters are represented in orange.

Here, we present a computationally inexpensive way of injecting a structural inductive bias into a pretrained Transformer. In this chapter, we focus specifically on introducing an inductive bias that is helpful for tasks that traditionally have been approached with Finite State Transducers (FSTs). We choose FSTs because they are formally well understood, are easy to generate automatically, and are one of the simplest computational devices that are useful in NLP applications. Hence, they provide a simple but nevertheless practical context that allows us to thoroughly test and analyze our method.

Our approach (SIP, for **S**imulation-**I**nduced **P**rior) is simple (see Fig. 5.1): given a representation of an FST and an input string, a Transformer is further pretrained to predict what the output of the FST is on the given input. We assume that FSTs are not specified for fine-tuning on downstream tasks, so we replace the FST with tunable embeddings and fine-tune the model solely on input/output pairs. Since we fine-tune all parameters, the model can deviate from FST-like behavior if needed.

Contributions We show that a model pretrained with SIP has an inductive bias that improves systematic generalization and few-shot learning for 'FST-like' downstream tasks. SIP not only improves systematic generalization on FST tasks similar to those seen during pretraining but also on ones that are structurally more complex. The same pretrained model also transfers well to natural data and achieves strong results on few-shot learning of text editing (e.g. Jane Doe $\rightarrow$ J. Doe) and grapheme-to-phoneme conversion.

Our probing experiments give insights into how the inductive bias is injected: SIP not only leads to the imitation of the input/output behavior of FSTs, but encourages



(a) A deterministic FST.

(b) A non-deterministic but functional FST.

Figure 5.2: Examples of *functional* FSTs. The FST in (a) deletes leading zeros. The FST in (b) replaces any `0` by a `1` if the last input symbol is a `1`. Conversely, if the last symbol is a `2`, any `0` is replaced by a `2`. The output can only be determined after the last input symbol.

dynamics to emerge that *simulate* transitions between FST states in the hidden representations. Fine-tuning can leverage these dynamics, providing the inductive bias, and learn representations that resemble those of ground truth FSTs.

5.1 Finite State Transducers

We briefly review Finite State Transducers (FSTs), which we use in our experiments. FSTs are closely related to Finite State Automata (FSAs). While an FSA describes a set of strings, an FST describes a *relation* between strings, i.e. a set of pairs (x, y), where x is an input and y is an output.

FSTs can be visualized as labeled directed graphs (see Fig. 5.2), where the nodes are called *states* and the edges are called *transitions*. Consider the path $\texttt{q0} \xrightarrow{\texttt{0:1}} \texttt{q1} \xrightarrow{\texttt{0:1}} \texttt{q1} \xrightarrow{\texttt{1:1}} \texttt{q2}$ in Fig. 5.2b. This path is called an *accepting path* because it starts in an *initial* state (indicated by an arrow 'from nowhere' pointing to the state), and it ends in a *final* state (indicated by double circles). An accepting path shows what an input can be mapped to. In this case, the path shows that the FST transduces the input `001` into the output `111`. We can read off which input an accepting path associates an output to by concatenating all the strings along the path occurring before ':'. The output can be determined by concatenating the strings after ':'. Hence, each transition $\xrightarrow{\sigma:\rho}$ can be thought of as 'replacing' σ by ρ. Inserting and deleting can be achieved by means of the empty string, written as ε. For example, Fig. 5.2a 'replaces' leading zeros by an empty string, effectively deleting them.

In general, an input can be paired with arbitrarily many different outputs. We call an FST f **functional** if every input x is paired with at most one output y, and use the notation $f(x)$ to refer to y. All FSTs we consider here are functional.

In this work, we investigate generalization across different sub-classes of FSTs, namely from the less expressive deterministic FSTs to non-deterministic FSTs. An FST is called **deterministic** if (i) it has a unique initial state, (ii) for all states q and input symbols σ there is at most one transition $q \xrightarrow{\sigma:\rho} q'$ and (iii) $\sigma \neq \varepsilon$. Intuitively, this means that in any state, for an input symbol σ there is at most one possible next state

and one possible output, and hence for any input string, there is at most one path that is compatible with it. Because of this, we can always infer a prefix of the output by looking only at a *prefix* of the input string and ignoring the rest. For example, consider the input prefix `001`. In the deterministic FST in Fig. 5.2a, we know that the output has to start with `1` because there is only one path that is compatible with `001`. In contrast, in the non-deterministic FST in Fig. 5.2b, three paths are compatible with `001` that have different outputs. In that case, we can only determine the output once we look at the last symbol of the input. In short, non-deterministic FSTs can take context to the right into account, but deterministic FSTs cannot.

5.2 Simulation-Induced Prior

Our approach largely follows the pretraining and fine-tuning paradigm. We first pretrain on synthetic FST tasks by giving the model a representation of an FST as a prefix and an input string (see Fig. 5.1). The training objective is to predict the output of the FST on the input string. Our research hypothesis is that training a model to predict the behavior of an FST incentivizes the model to acquire reusable dynamics that internally simulate FSTs. When fine-tuning the model using a tunable prefix instead of an encoding of an FST, these dynamics should be easy to leverage and provide a structural inductive bias for FST-like tasks.

Pretraining

During pretraining, the model is given a representation of an FST and a string in its domain and has to predict the output of that FST on the given input string. The input to the Transformer is a sequence of vectors from $\mathbb{R}^d$, which consists of a prefix that represents the FST f and a suffix comprised of the embeddings of the input string (see Fig. 5.1):

$$\underbrace{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k}_{\text{FST encoding}}, \quad \underbrace{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n}_{\text{Embeddings of input to FST}} \tag{5.1}$$

Each $\mathbf{h}_i$ encodes a transition $p \xrightarrow{\sigma:\rho} q$ as a vector:

$$\mathbf{h}_i = \mathbf{W}[\text{EMB}_{\text{State}}(p); \text{EMB}_{\text{State}}(q); \text{EMB}_{\text{Symbol}}(\sigma); \text{EMB}_{\text{Symbol}}(\rho); \text{EMB}_{\text{Final}}(e)]$$

where $[;]$ represents vector concatenation, e indicates if q is a final state, and W is linear layer that ensures that $\mathbf{h} \in \mathbb{R}^d$. All embeddings are simple look-up tables based on the id of the state or symbol. The initial state of the FST is always assigned the id 0, and positional embeddings are used as usual. The model is trained to maximize the log probability of the output $y = f(x)$ of the FST f.

Fine-tuning

After pretraining, we can apply our model to a downstream task and fine-tune it. We assume we do not have access to an FST for the downstream task, and therefore, we

replace the FST encoding with a sequence of tunable embeddings. That is, the input to the model is a sequence of vectors:

$$\underbrace{\mathbf{h'}_1, \mathbf{h'}_2, \ldots, \mathbf{h'}_k}_{\text{Tunable embeddings}}, \underbrace{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n}_{\text{Input}}$$

where $\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n$ are the embeddings of the input tokens, $\mathbf{h'}_i \in \mathbb{R}^d$ are the tunable embeddings and k is a hyperparameter. The embeddings $\mathbf{h'}_i$ are initialized to the average of the encoding of multiple FSTs from the pretraining phase. The most straightforward way to fine-tune is to only modify $\mathbf{h'}$ because we are looking for an FST-like task representation. This is similar to prompt tuning (Lester et al., 2021). However, this does not work well on tasks outside the pretraining distribution. Hence, we fine-tune the entire model, including the prefix, and use a higher learning rate for the prefix than for the rest of the model (see Appendix D.6).

Constructing Pretraining Data

To create our pretraining data, we sample 40,000 deterministic FSTs. For every FST, we sample 5 input/output pairs with input lengths up to 35. In total, this leads to 200,000 input/output pairs for training along with their FSTs. To describe the sampling procedure in more detail, we use an overall vocabulary $\mathcal{V}$ consisting of the printable ASCII tokens and the Unicode block for IPA symbols (used for transcribing speech). Seq2seq tasks in the wild usually do not use the whole space of this vocabulary, so for each task T we first uniformly sample the vocabulary size $|\mathcal{V}_T|$ between 5 and 25 and then uniformly select a subset $\mathcal{V}_T \subseteq \mathcal{V}$. Then, we uniformly sample the number of states $|Q_T|$ between 2 and 4, and the number of final states between 1 and $|Q_T|$. For every state q and every symbol $\sigma \in \mathcal{V}_T$ we introduce at most one outgoing transition to a state q', chosen uniformly at random. This ensures that the FST is deterministic. We then sample the output for the transition: either a symbol $\rho \in \mathcal{V}_T$ or ε. Finally, we minimize the number of states of the FST using OpenFST (Allauzen et al., 2007), and exclude those without cycles, as they express finite relations. See Appendix D.2.1 for details.

In practical applications of FSTs, in particular for text editing, one often wants to keep certain parts of the input unchanged. This can be achieved with a set of transitions of the form $q \xrightarrow{\sigma:\sigma} q'$ for all $\sigma \in \mathcal{V}_T$. Since it is very unlikely to sample such a set of transitions, we use a special symbol that acts as a shorthand for this. We use analogous shorthands for converting lower case to upper case and vice-versa. These shorthand symbols are also used when encoding the FST for pretraining (Eq. (5.1)).

5.3 Evaluating SIP's Inductive Bias

To understand the effects of our pretraining procedure on the inductive bias of the model and on the downstream performance, we first explore systematic generalization on a suite of synthetic FST tasks, which we call *fstt*. This allows us to precisely control the similarity between the pretraining and the downstream task.

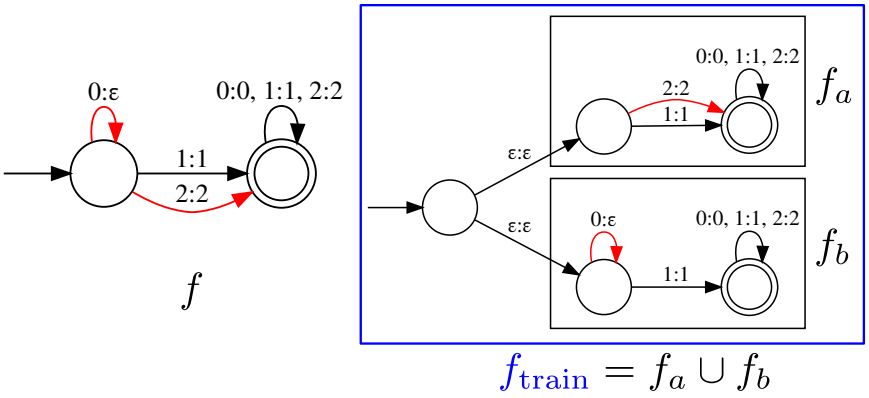


Figure 5.3: Constructing training data for evaluating unseen combinations of transitions. Based on the given FST f, we construct an FST f_{train} that withholds the *combination* of the two red transitions.

5.3.1 Evaluation Methodology with fstt

To evaluate the degree to which a model has an inductive bias towards FSTs, we now describe two methods for generating training and test data that reward a model for showing important aspects of FST-like systematic generalization.

Iteration Generalization Cycles are a characteristic feature of FSTs, and iteration generalization tests if a model learns that cycles can be traversed more often than seen during training. More specifically, given an FST, we generate training data that requires visiting any state only a few times (*iteration count* up to 3). In the test data, the model has to generalize to visiting states more often (iteration count at least 4). This is a case of length generalization (Lake and Baroni, 2018) but tailored specifically to FSTs.

Unseen Combinations of Transitions When an FST processes a string, the set of possible next transitions only depends on the current FST state; it does not matter how the current state was reached. Hence, a model with an inductive bias towards FSTs should also not be overly sensitive to how a state is reached, and correctly handle situations where a specific *combination* of transitions was unobserved during training. For example, consider the FST f on the left in Fig. 5.3, which deletes leading zeros from a number. Suppose that a model is trained on examples such as `0012`, `2201`, `1012` but no training example contains the combination of leading zeros followed by a `2`, which corresponds to using the combination of the two transitions highlighted in red. If the model has an inductive bias towards FSTs, it should generalize to this unseen combination and correctly handle inputs such as `0021`. UC is related to the MCD splits of Keysers et al. (2020), who also withhold combinations of seen elements to assess systematic generalization.

To generate appropriate training and test data for this, we sample a pair of adjacent transitions $\langle t_a, t_b \rangle$, such as the red transitions in Fig. 5.3, and ensure that no training example uses both transitions within the *same* string. We do this by creating a new FST f_{train} as follows (right side of Fig. 5.3): We create two copies f_a, f_b of the original FST f. In f_a, we remove the transition t_b; in f_b, we remove the transition t_a. Then

$f_{\text{train}} = f_a \cup f_b$, which can be constructed by introducing a new initial state with ε-transitions into the respective initial states of f_a and f_b. Hence, t_a or t_b may be used – but not both in the same string. Note that f_{train} still describes a partial function (rather than a relation) because any accepting path in f_a and any accepting path in f_b is also an accepting path in f. As a result, whenever f_a and f_b are both defined, they agree on the result $f_a(x) = f_b(x) = f(x)$. We generate test data by sampling examples from f for which f_{train} is *not* defined, which can be efficiently implemented with a complement construction.

To make the generalization setup more challenging, these steps can be applied to multiple pairs of adjacent transitions at the same time, i.e. to withhold $\langle t_a^1, t_b^1 \rangle, \ldots, \langle t_a^k, t_b^k \rangle$: We create the copy f_a and remove the transitions $t_b^1, \ldots, t_b^k$ from f_a and analogously remove $t_a^1, \ldots, t_a^k$ from f_b. See Appendix D.2.3 for details.

Relation to Occam's Razor Occam's razor states that one should prefer simpler hypotheses over more complex ones if they describe the observed data equally well. The training/test splits we present here can also be seen as testing if a model implements aspects of a version of Occam's razor for FSTs, i.e. if a model prefers 'simple' FSTs, i.e. those with a small number of transitions, over more complex ones.

To make this relation clearer, suppose one has already committed to FSTs as the right family of functions to describe the data generated from f_{train}. Both FSTs f and f_{train} perfectly predict the outputs on the training data but f_{train} is more complex and has a considerable amount of repeated structure. Hence, testing on the combination of withheld transitions can give us insight into whether the model has inferred a simpler explanation that is closer to f or a more complex one closer to f_{train} that makes an unnecessary distinction how a state was reached.

A similar argument can be made for iteration generalization: restricting the number of iterations that can be made through a cycle corresponds to constructing a new, larger FST that 'unrolls' the cycle for a fixed number of iterations; this makes the description of this new FST longer and hence more complex than the original one.

5.3.2 Setup and Baselines

To make a fair comparison, all models we experiment with in the main paper share the same architecture and are initialized from the same checkpoint before any additional pretraining, namely ByT5-small (Xue et al., 2022). ByT5 has 300 million parameters and was pretrained on the multilingual C4 corpus. It uses raw bytes as tokens, which enables full Unicode support and is a natural unit to consider for FST-like tasks such as text editing and grapheme-to-phoneme conversion. We report additional results with a T5-Base model in Appendix D.3.3, where we observe similar trends. All experiments use greedy decoding.

SIP-d4 This is a model using the method we propose in this work. We pretrain on the data generated in Section 5.2 (**d**eterministic FSTs, with up to **4** states) for 20 epochs. This model achieves an average sequence-level accuracy of 98% on predicting the output of an unseen FST from the training distribution. For fine-tuning, we use a

prefix of length 50 for all experiments in this paper. As an ablation, we also fine-tune the model without the prefix of tunable embeddings (-prefix).

Naive pretraining We use the same pretraining data as for SIP-d4 but omit the description of the FST and only train on input/output pairs. We pretrain for a single epoch as longer pretraining deteriorates performance on downstream tasks, potentially because of memorization.

Task embeddings (TE) TE is a simplified version of SIP. Instead of using an encoding of an FST in the prefix, this baseline uses 50 randomly initialized embeddings specific to each FST. The embeddings are learned from examples jointly with the rest of the model. Several works have used a single embedding to encode a domain/task in multi-task learning (Tsvetkov et al., 2016; Stymne et al., 2018; Zhang et al., 2022b). Using a shorter tunable prefix resulted in considerably worse performance in our setup. TE is fine-tuned analogously to SIP, i.e. with a prefix of tunable embeddings.

Set Wu et al. (2022) investigate the effectiveness of 18 simple synthetic pretraining tasks and found Set to perform best on average. The task is to deduplicate characters such that every type occurs only once, e.g. the input `dabacd` becomes `dabc`. This task can be represented by a deterministic FST, albeit a very large one with 2^n states for a vocabulary of size n. We sample 200,000 examples according to the procedure of Wu et al. (2022) to match our pretraining dataset size. We train for a single epoch, at which point it reaches 99.7% accuracy on unseen strings. Again, this performed better on downstream tasks than training for 20 epochs.

5.3.3 Systematic Generalization within the Pretraining Distribution

First, we want to establish to what degree the pretraining has conferred any inductive bias on the distribution it was pretrained on. In this set of experiments and the remainder of this section, models are fine-tuned and we provide no ground-truth FSTs.

Setup For each generalization setup, we generate 5 unseen FSTs with 4 states each using the same procedure as for the pretraining, ensuring they have not been seen in the pretraining. We fix the vocabulary size to its maximum value (25) in the pretraining data and only use printable ASCII characters in order to reduce variance across tasks. To evaluate UC, we withhold the combination of up to 20 pairs of transitions and generate 5000 training examples with lengths 3 to 15 and corresponding test data as described in Section 5.3.1. For iteration generalization, we generate training examples with a maximum iteration count of 3 and test on longer examples of length up to 30 with an iteration count of at least 4. Since the out-of-distribution performance of two checkpoints of the same model can vary significantly, we report averages on the test set of the last 10 epochs.

Results The results can be found in Table 5.1. On average, SIP-d4 achieves close to perfect accuracy (with one outlier on UC, skewing the mean). TE also shows a clear improvement over the other baselines, but SIP-d4 outperforms TE by a large margin.

	Iteration		UC			
	Acc↑	ED↓	Acc↑	$\tilde{\text{Acc}}$ ↑	ED↓	$\tilde{\text{ED}}$ ↓
ByT5	37.8	5.87	47.4	57.5	1.49	0.93
Naive	42.6	4.41	44.9	43.2	1.52	1.35
Set	44.4	4.58	43.6	42.0	1.47	1.31
TE	61.3	2.49	57.3	63.1	1.13	0.74
SIP-d4	**94.8**	**0.12**	**73.1**	**93.3**	**0.61**	**0.13**
-prefix	84.9	0.62	61.1	76.3	0.99	0.50

Table 5.1: Evaluating systematic generalization on fstt with 4 states. We report arithmetic means over 5 tasks. ED is edit distance. Due to an outlier task on UC, we additionally report the median as $\tilde{\text{Acc}}$ and $\tilde{\text{ED}}$.

Model	Pretrain Length	Test Length			
		40 to 70		90 to 110	
		Acc↑	ED↓	Acc↑	ED↓
ByT5	1024	29.3	15.6	1.4	55.4
Set	35	31.8	13.9	1.0	55.1
TE	35	35.2	9.4	0.1	50.8
SIP-d4	35	69.4	2.6	3.4	34.5
SIP-d4-long	110	**92.8**	**0.2**	**81.5**	**1.1**

Table 5.2: Average generalization ability across 5 FSTs with 4 states. Models were fine-tuned on inputs of length up to 15, and tested on inputs beyond the pretraining length cutoff of 35.

This suggests that SIP-d4 and TE, to a lesser extent, indeed have acquired a stronger inductive bias for FSTs than the other methods. Using SIP-d4 without the tunable prefix leads to a substantial drop in accuracy, highlighting its importance. We analyze the representations learned by SIP-d4 in the tunable prefix in Appendix D.4.1.

5.3.4 Longer Strings than during Pretraining

Does the inductive bias of SIP extend to longer strings than those seen during pretraining? To address this question, we test a more extreme version of length generalization. Similarly to before, we generate 5 FSTs with 4 states each. The training data contains strings with lengths up to 15 and the test data contains strings of length between 40 and 70, which is beyond the length cutoff of 35 used during pretraining of SIP-d4. We repeat this process and also generate a version with test strings of length between 90 and 110.

We report results in Table 5.2. ByT5 struggles with this generalization setup across the board. TE performs slightly better, in particular in terms of edit distance. SIP-d4 performs remarkably well on lengths 40 to 70 which are beyond the lengths seen during its pretraining. However, performance drops very sharply for inputs of length 90

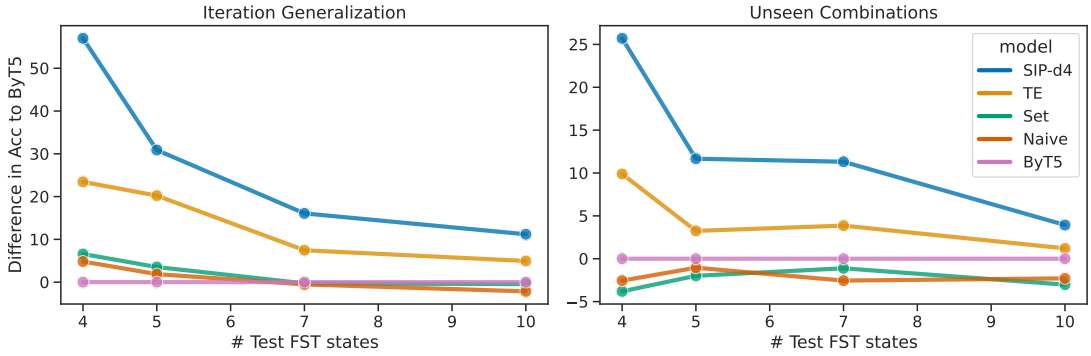


Figure 5.4: Evaluation on deterministic FST tasks with more states than seen in pretraining. We show the deviation in percentage points from ByT5.

to 110. We hypothesize that this happens because the relevant positional embeddings were not properly pretrained by SIP.[2] As another point of comparison and to test if this can be addressed with appropriate further pretraining, we continue pretraining SIP-d4 on 40,000 FSTs with strings of length up to 110 (SIP-d4-long). As expected, additional pretraining with longer strings consistently improves length generalization. Although accuracy maintains a much higher level in comparison to SIP-d4, there is still a considerable reduction in accuracy for the more challenging length generalization setup.

5.3.5 More Complex FSTs

Does the inductive bias introduced by SIP extend to more complex FST tasks than seen during pretraining? To investigate this, we use the same sampling methodology as for constructing the pretraining data but generate FSTs with more states. SIP-d4 was pretrained on FSTs with up to 4 states, and we evaluate on FST tasks with 5, 7 and 10 states.

We show in Fig. 5.4 how the individual models deviate from the accuracy of ByT5 as a function of the number of states in the test FST. SIP always performs best by a clear margin, regardless of the number of states in the FSTs. As we increase the number of states and move away from the pretraining distribution, SIP improves less over the baselines. We see a similar pattern for TE but with considerably smaller improvements over ByT5.

5.3.6 Non-Deterministic FSTs

As shown in the previous section, SIP still works well for more complex FST tasks than seen during pretraining. However, this evaluation focused on the favourable case where both pretraining and evaluation involve the same class of FSTs, namely deterministic FSTs. Deterministic FSTs can only take left context into account (see Section 5.1), which is a restrictive assumption. Here, we evaluate if the inductive bias conferred

[2] T5-style models use relative position encodings that add a bias term to the attention logits depending on the relative distance between tokens.

	Iteration		UC	
	Acc↑	ED↓	Acc↑	ED↓
ByT5	83.4	0.52	83.1	0.40
Naive	83.1	0.49	84.2	0.37
Set	82.3	0.52	83.7	0.37
TE	84.2	0.49	82.7	0.42
SIP-d4	87.8	0.32	90.0	0.24
SIP-d4+	88.2	0.30	90.5	0.22
SIP-nd7	**89.5**	**0.27**	**91.2**	**0.18**

Table 5.3: Evaluation on non-deterministic FSTs. We report averages over 5 tasks.

by SIP carries over to non-deterministic functional FSTs, i.e. those that can also take context to the *right* into account.

We automatically generate 5 non-deterministic FSTs with 21 states each (see Appendix D.2.2 for details) and report averages in Table 5.3. Despite the structural mismatch between pretraining and the downstream tasks, SIP-d4 shows clear improvements over the baselines. Interestingly, TE does not consistently outperform the other baselines, despite its stronger results on deterministic FSTs.

Our pretraining procedure does not hinge on using deterministic FSTs. This raises the question if we can achieve even better performance by adjusting the inductive bias. To investigate this, we further pretrain SIP-d4 on 40,000 non-deterministic FSTs with up to 7 states, which we call SIP-nd7. To control for the additional training data of SIP-nd7, we also further pretrain SIP-d4 with the same number of deterministic FSTs with the same characteristics as in Section 5.2 (SIP-d4+). The results in Table 5.3 show better performance of SIP-nd7, which supports the hypothesis that the inductive bias can be adjusted. SIP-d4+ shows a smaller improvement over SIP-d4. Based on 5 additional FSTs per setup to gain more statistical power, we found that the difference between SIP-nd7 and SIP-d4+ is statistically significant ($p \approx 0.017$, $n = 20$, paired approx. permutation test).

5.4 Transfer to Natural Data

In this section, we investigate to what degree the inductive bias from pretraining on synthetic data transfers to tasks with natural data that have been traditionally approached with finite-state methods.

Low-resource Grapheme-to-Phoneme Conversion

Grapheme-to-phoneme conversion is the task of converting a word as a sequence of symbols (for example, letters in the Latin alphabet) into a description of how this word is pronounced as letters in the IPA alphabet. For example, a possible pronunciation of 'explanation' is [ˌɛksplə'neɪʃən] . Grapheme-to-phoneme conversion can be part of text-to-speech pipelines and FSTs for this purpose are usually two or three magnitudes

	ban	cop	got	lao	syl	tel	tzm	Avg
Charsiu	68.3	7.8	67.0	35.1	47.6	73.3	18.6	45.4
ByT5	50.2	1.0	30.7	1.9	9.8	6.9	2.7	14.8
Set	53.9	2.2	58.2	5.8	28.2	27.7	6.4	26.1
TE	54.7	1.9	37.0	5.1	30.0	16.2	7.4	21.8
SIP-d4	**59.2**	**6.6**	56.5	**8.2**	**39.8**	**33.1**	**11.0**	**30.6**
-prefix	55.1	3.2	**63.9**	7.8	28.0	28.9	7.0	27.7

Table 5.4: Grapheme-to-phoneme conversion with 100 training examples. We show averages of 5 selections of training examples.

larger than the FSTs we constructed for pretraining. Because of this, it enables us to test how far beyond the pretraining distribution SIP remains helpful. We focus on learning from small amounts of data, for which a structural inductive bias towards FSTs should be particularly helpful. We evaluate on 7 low-resource languages from different language families that use their own scripts (Balinese, Coptic, Gothic, Lao, Sylheti, Telugu and Central Atlas Tamazight). We obtained the data from Wikipron (Lee et al., 2020).

As a soft upper bound, we compare with Charsiu (Zhu et al., 2022), which is a ByT5-small model that has been further pretrained on 7.2 million examples of grapheme-to-phoneme conversion across 100 languages. Although Charsiu was not exposed to the scripts of the languages we chose, in some cases, it has seen related languages whose scripts are encoded similarly in Unicode.[3]

We report accuracies in Table 5.4, and phoneme-error-rates in Appendix D.3.2; trends are identical. The original ByT5-small model performs worst on average despite being a strong model for grapheme-to-phoneme conversion in general (Xue et al., 2022). On average across the languages, SIP-d4 outperforms the other methods that pretrain on synthetic data as well as ByT5. The difference between SIP-d4 and Set is statistically significant ($p \approx 4 \times 10^{-4}$, paired approx. permutation test). Fine-tuning SIP-d4 without the tunable prefix leads to a drop in performance, except for Gothic. Charsiu performs very well on Telugu, potentially because of how it is encoded in Unicode and its large overlap in lexicon with Sanskrit (Staal, 1963), which is part of its training data.

Few-shot Text Editing

Learning simple text editing tasks (Jane Doe → J. Doe) from a handful of examples with a Transformer requires a strong structural inductive bias to overcome competing explanations of the data and hence provides a good benchmark for our approach. While

[3]For Brahmic scripts such as Devanagari (used e.g. for Hindi) and Telugu, Unicode encodes characters from different scripts in a similar way if they have structural (and historic) ties across the scripts (The Unicode Consortium, 2024). For instance, the Telugu character ఆ and the Devanagari character आ can both be read as [aː] and are encoded as byte sequences `E0 B0 86` and `E0 A4 86`, respectively, where the first two bytes essentially designate the script, and the last byte identifies the character within the script. Hence, knowledge about the phonetics of e.g. Hindi can also be helpful for Telugu.

	rev-name		sur-initial		FST		Overall	
	Acc↑	ED↓	Acc↑	ED↓	Acc↑	ED↓	Acc↑	ED↓
ByT5	11.8	6.81	47.2	1.76	47.6	1.42	45.7	1.72
Charsiu	43.8	1.73	52.8	0.87	62.4	0.74	60.9	0.80
Set	79.0	1.34	41.5	3.37	68.2	0.71	67.4	0.89
TE	80.3	1.08	88.2	0.41	**95.7**	**0.11**	**94.5**	0.17
SIP-d4	92.4	0.34	**97.2**	**0.10**	91.6	0.13	91.9	**0.14**
-prefix	**97.8**	**0.10**	72.6	0.51	89.0	0.27	91.4	0.18

Table 5.5: Averages of accuracy and edit distance across 5-shot text editing tasks based on 8 draws of training examples. We report results grouped by tasks that cannot be solved by a compact FST (rev-name, sur-initial), tasks that can be solved by FSTs, and overall averages.

current LLMs may seem like the ideal choice for such tasks, they are prone to hallucinations, e.g. ignoring the input and resorting to frequent entities (see Appendix D.5 for an example).

Text editing has been studied in the context of program synthesis and we evaluate on 19 such tasks from the SyGuS competition 2017 (Alur et al., 2017). Instead of predicting a program, our model directly operates on input/output examples. We note that 17 of these tasks can be solved by compact FSTs, whereas two cannot. These two tasks are *rev-name* (Jane Doe → Doe Jane) and *sur-initial* (John Doe → Doe, J.), which require tracking information about the first name in the states.

We report results for 5-shot experiments in Table 5.5. SIP-d4 and TE excel at this, reaching well above 90% accuracy on average, whereas the other methods perform worse by a large margin. Charsiu does not perform clearly better than baselines such as Set – even though it obtains excellent results on grapheme-to-phoneme conversion. Interestingly, TE performs better than SIP-d4 on the tasks that can be solved with FSTs, potentially because the initialization of the prefix for TE follows the same distribution as during pretraining, which is not the case for SIP. However, SIP considerably outperforms TE on the two tasks that cannot be compactly represented by FSTs, suggesting that some of the dynamics acquired during pretraining can sometimes be leveraged in other contexts as well. Fine-tuning SIP-d4 without the tunable prefix leads only to a very small drop in accuracy on average.

5.5 Analysis: SIP leads to FST simulation

We motivated our approach by the hypothesis that SIP's pretraining encourages the model to simulate FSTs internally, and that this provides the structural inductive bias. In this section, we present evidence that (i) SIP models indeed approximately simulate FSTs in the hidden states, and (ii) that the dynamics responsible for simulation are reused after fine-tuning all parameters on input/output pairs only.

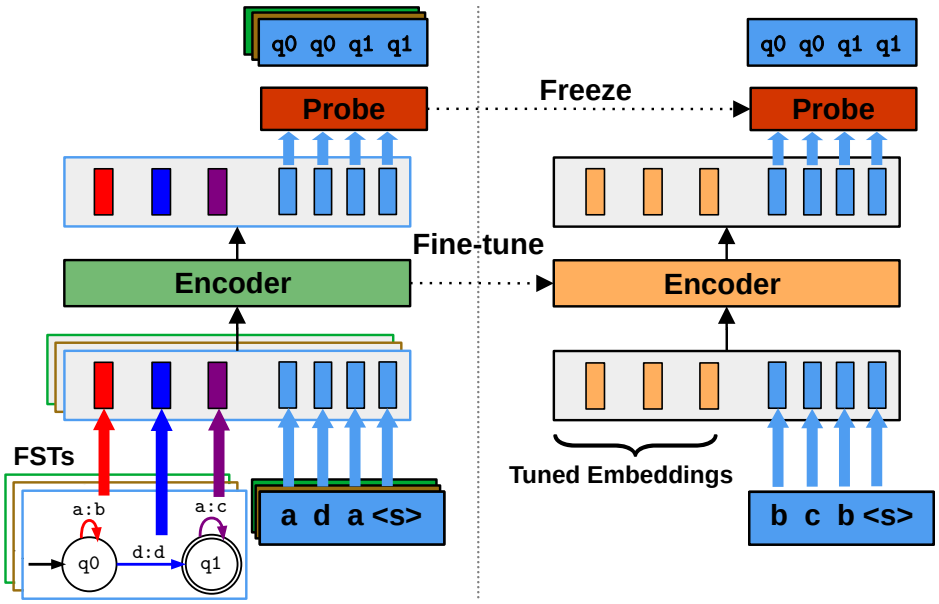


Figure 5.5: Left: we train a linear probe on the encoder representations of a SIP pretrained model to predict for each input token x_i which state the encoded FST is in before processing x_i. The end-of-sequence token is represented as <s>. Right: we freeze the trained probe, fine-tune the SIP model on input/output pairs and extract state sequences from it with the probe.

Probing for Simulation of Transitions between FST states

For a model to simulate FSTs in its hidden representations, it must be able to track the FST state when processing a string, and it should be possible to extract the FST state with a probe. To test this, we mirror the pretraining setup and provide SIP-d4 with an FST and an input string (Fig. 5.5, left). For each token, we extract the top-layer activations of the encoder, and learn a linear probe with a softmax layer to predict the ID of the state that the given FST is in *before* processing that token. Since state IDs are largely arbitrary, the probe has to learn to relate the hidden representations to the FST presented in the input.

The probe achieves 99.3% token-level accuracy on a test set with unseen FSTs, and a whole-sequence accuracy of 93.9%. We also evaluate a trivial heuristic that returns a random state that has an appropriate outgoing transition for each token in the input. This heuristic achieves a token-level accuracy of 68.9%, and a whole-sequence accuracy of only 17.8%. A probe trained on ByT5 representations, i.e. before SIP pretraining, performs even worse at 42.9% token-level accuracy and whole-sequence accuracy of only 7.1% (see Appendix D.4.2). Hence, the model has learned a non-trivial way to simulate transitions between states of the FST encoded in the prefix. This is remarkable because the pretraining procedure for SIP-d4 does not provide supervision for *how* to process strings.

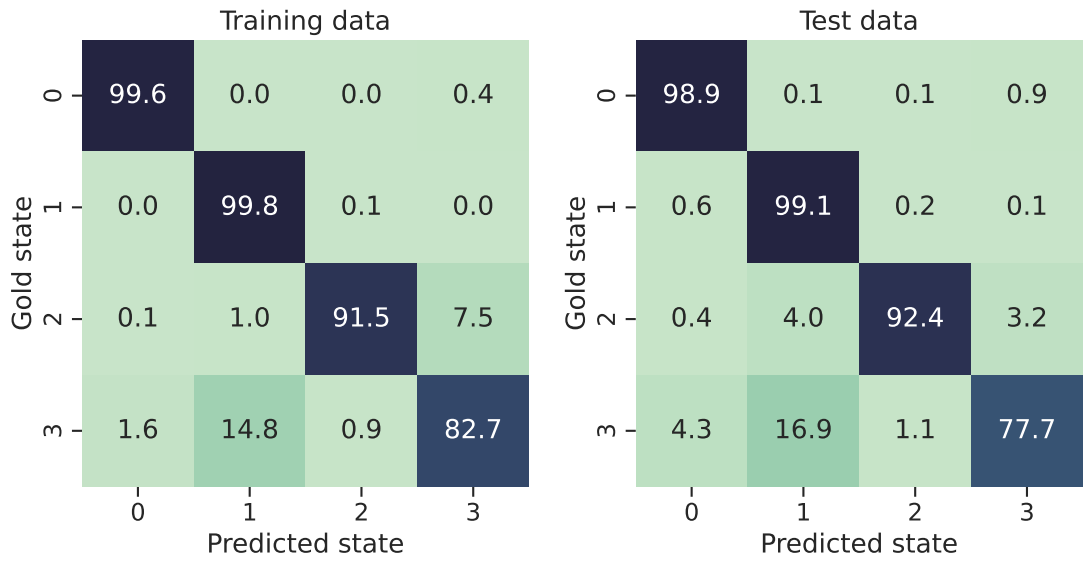


Figure 5.6: Row-normalized confusion matrices on the training and test data between ground truth and the state predicted by the frozen probe applied to fine-tuned models. We average across the 5 iteration generalization tasks (Section 5.3.3).

Reuse of Simulation Dynamics during Fine-Tuning

While the analysis so far shows that SIP leads to the simulation of state transitions after pretraining, does the model leverage the simulation ability on downstream tasks? Recall that we fine-tune all parameters of the model (Section 6.1.3), so the model could employ a very different strategy to fit the data. To investigate this, we set the trained probe aside and freeze it (Fig. 5.5, right). We then fine-tune SIP-d4 on the iteration generalization tasks with 4 states in the gold FST (cf. Table 5.1). Finally, we apply the frozen probe to the fine-tuned model to see if the state sequences we extract are similar to those of the ground truth FST. Fine-tuning SIP-d4 could induce the same FST as the ground truth but use a different numbering of the states. To account for this, for each of the five tasks, we find the isomorphism between the predicted state IDs and the ground truth that gives the best match on average.

The results are presented in Fig. 5.6 as confusion matrices between predicted and gold states. The probe extracts state sequences that resemble the state sequences of the gold FST (up to isomorphism), both on the training data and the out-of-distribution test data. We also find that deviation from the ground truth state sequence correlates with errors by the fine-tuned model: if the probe extracts correct state sequences, the model achieves an accuracy of 98.6% on the iteration generalization tasks, whereas it drops to 89.8% when the probe extracts state sequences that deviate. The difference is statistically significant (approximate permutation test, $p \approx 5 \times 10^{-5}$). Overall, this shows that the fine-tuned model reuses the dynamics for FST state tracking and can learn representations similar to the ground truth FST.

5.6 Related Work

Pretraining with synthetic tasks Pretraining a model on a synthetic task to introduce specific inductive biases has been explored by several recent works. Krishna et al. (2021) identify useful 'skills' for news summarization and develop a pretraining task accordingly. LIME (Wu et al., 2021b) targets mathematical reasoning and is pretrained on string manipulation that resembles formal reasoning. Wu et al. (2022) investigate a range of simple synthetic tasks for pretraining and show that some perform remarkably well across a range of downstream tasks. Papadimitriou and Jurafsky (2023) consider pretraining with several synthetic languages to investigate which helps most for language modeling. In contrast to these works, our approach targets simulating a computational device and maintains a closer relation to the pretraining setting because of the tunable prefix.

A challenge for using individually hand-crafted tasks is to cover a sufficient space of phenomena that are relevant to downstream tasks. Instead of training on a single task only, McCoy et al. (2020); McCoy and Griffiths (2023) meta-learn on a distribution of tasks using MAML (Finn et al., 2017). Our approach also uses a distribution of tasks, but it scales better than MAML-based methods because MAML requires computing and storing second-order derivatives. For example, the Transformer we train has a magnitude more parameters than the LSTM of McCoy and Griffiths (2023) and is pretrained on a smaller GPU. In addition, as the complexity of each individual task grows, MAML requires more examples per task. We circumvent this by using a compact and unambiguous description of each task instead.

Simulating execution The idea of using a neural network to predict the outcome of the execution of a computational device or code has come up in several contexts over the last few years. Early work by Zaremba and Sutskever (2014) investigates it as a challenging benchmark for LSTM-based seq2seq models. Recent works have explored simulating (aspects of) code execution for various downstream applications, such as program synthesis (Austin et al., 2021), or debugging and code analysis (Bieber et al., 2022) as well as reverse engineering (Pei et al., 2021). Finally, Finlayson et al. (2022) train a Transformer to interpret regular expressions: given a regular expression and a string, the task is to decide if the string is in the regular language. There are three crucial differences between their work and ours: (i) they investigate the empirical capabilities of Transformers while we take the perspective of introducing structural inductive biases for downstream tasks, (ii) they consider binary outputs whereas we consider sequential outputs, and (iii) we perform probing experiments showing strong evidence for FST simulation in the hidden representations.

Emergent World Representations Our analysis provides evidence that our model trained with SIP internally simulates transitions between FST states even though it was not explicitly supervised to do so. Similar observations have been made for Language Models trained to play Othello (Li et al., 2023b) and chess (Karvonen, 2024), where the model was found to acquire a representation of the board state simply from being trained to predict the next move.

5.7 SIP as a Learned Continuous Relaxation

In Chapters 3 and 4 we introduced two continuous, differentiable relaxations of discrete structural transformations, namely a fertility layer that makes copies of elements of the input and a permutation layer that reorders its input. Here, we argue that SIP can also be seen as a continuous relaxation (namely of FSTs) with the crucial difference that the relaxation is learned rather than hard-coded. We point out this relation informally here and make it more precise in Appendix D.1.

To clarify what we mean by continuous relaxation, consider the differentiable permutation layer from Chapter 4. We can view permuting a sequence as a discrete operation that takes a sequence of tokens and a description of a permutation as input. For instance, the description could tell us which tokens are adjacent in the output. The operation simply reads the given description of a permutation and applies it to the sequence of tokens. However, since the operation is discrete, it is not differentiable. A *continuous relaxation* replaces the discrete operation with a *probability distribution* that conditions on a description of a permutation. Crucially, to be a continuous relaxation, the description now has to be a continuous variable, e.g. a set of scores, and the probability distribution has to be differentiable wrt the scores. In addition, the probability distribution has a temperature parameter. As we decrease the temperature parameter towards 0, the distribution puts more and more probability mass on a single permutation and becomes a closer and closer approximation of the discrete permutation operation.

SIP is also a continuous relaxation in this sense. The model is a probability distribution that takes a string and a description of an FST in the form of a sequence of embeddings. If the model is properly trained, it assigns high probability to the output of the FST whose description is passed as input. The softmax over the output vocabulary at each time step has a temperature parameter that can be tweaked to make the behavior of the distribution more deterministic and more closely approximate the function represented by the true FST. In contrast to Chapters 3 and 4, the distribution is not manually designed but learned from data and occasionally produces wrong outputs (around 2% of the time on the pretraining distribution).

We can fine-tune SIP by backpropagating into the tunable prefix, i.e. the 'soft' description of FSTs. This approximately corresponds to backpropagation into the permutation or fertility model. However, when fine-tuning SIP we also backpropagate into the other model parameters, which makes the model more expressive but has no analogy in the permutation or fertility models. With sufficient fine-tuning, SIP can represent functions that FSTs cannot capture, or cannot capture concisely (as in Table 5.5). This is in contrast to the hard-coded relaxations of the fertility and permutation models.

5.8 Conclusion

SIP is a simple, efficient and adjustable method for introducing a structural inductive bias into a sequence-to-sequence model. In this chapter, we focus on an inductive bias towards FSTs, one of the simplest computational devices that is useful for NLP applications. We inject the inductive bias by pretraining a Transformer to simulate au-

tomatically generated FSTs, i.e. to predict the output of an FST given an input string and a description of the FST. Our experiments show that SIP imparts the desired inductive bias, resulting in improved systematic generalization and better few-shot learning for FST-like tasks. We show with probing experiments that a model trained with SIP simulates transitions between FST states in its hidden representations. This is somewhat surprising because the model was not explicitly trained to do so. In addition, we find that the dynamics behind this internal FST simulation can be leveraged during fine-tuning on downstream tasks, as we can extract meaningful state sequences from fine-tuned models. Finally, we have connected the methodology in this chapter with Part I as instances of continuous relaxations. In contrast to the fertility and permutation layers in Part I, SIP is learned relaxation rather than a hard-coded one.

Limitations Our investigation focuses on FSTs with a relatively small number of states. However, the results in Section 5.3.5 and in the experiments on grapheme-to-phoneme conversion show that even pretraining with FSTs with a small number of states has positive impacts for tasks that require larger or more complex FSTs.

The probing experiments show that the model simulates transitions between states similarly to an FST, but we did not perform a mechanistic interpretation of how exactly this is implemented in the weights. One potential mechanism behind the simulation behavior is the construction of Liu et al. (2022), who show that Transformer decoders can simulate transitions between states of deterministic finite automata for strings of length up to n using $O(\log(n))$ layers.

Acquiring a specific inductive bias by means of learning to simulate a computational device is a general idea that could be applied more widely. However, it might be unsuitable in cases where (i) it is difficult to formulate a reasonable computational device to simulate (such as document classification and sentiment analysis beyond keyword spotting), or (ii) the computational device would be infeasible to simulate within the computational restrictions of the model (e.g. trying to simulate Turing machines with a Transformer that can only process a fixed number of tokens, see e.g. Merrill and Sabharwal (2024)).

Our experiments focus on moderately sized models (300M parameters) with an encoder-decoder architecture, and we did not investigate large decoder-only models. Our methodology can also be applied to decoder-only models, and we do not foresee any reasons why it could be less effective in that setup.

Chapter 6

Pretraining to Perform Syntactic Transformations

In the previous chapter, we worked towards Goal 2 and introduced a framework for injecting an inductive bias into a standard Transformer by pretraining it to simulate a computational device, specifically Finite State Transducers. In this chapter, based on Lindemann et al. (2024b), we demonstrate that this framework is applicable beyond Finite State Transducers and apply an analogous methodology to a pretrained sequence-to-sequence model in order to strengthen its inductive biases for tasks that can be regarded as syntactic transformations. While standard pretraining already equips models with knowledge about syntax (Tenney et al., 2019; Hewitt and Manning, 2019; Mueller et al., 2022), structural generalization beyond the training distribution remains challenging (Yao and Koller, 2022; Li et al., 2023a). Here, we start from the hypothesis that this is because standard pretraining provides incomplete knowledge how to *use* syntactic information for *structural* tasks.

Traditionally, NLP has phrased many structural tasks as transformations of syntax trees, ranging from conversion of a sentence from active to passive voice (Oliva, 1988) to constructing a semantic representation for a sentence (Montague, 1970). Transformations of syntax trees can address a task in a very generalizable way by using the right abstractions. For example, when constructing the semantic representation of an NP, by the principle of compositionality, the same transformations can be used for NPs whether they serve as direct objects or as indirect objects. Hence, a model with an inductive bias for syntactic transformations should be capable of structural generalization and learn syntax-related tasks effectively from small amounts of training data.

In this chapter, we train a model to perform syntactic transformations, replacing the FSTs from Chapter 5 with transformation rules that act on syntax trees. Specifically, we train the model on a dataset of automatically generated syntactic transformations of English dependency trees. Given a description of the transformation as a prefix and an input sentence, the model is further pretrained to predict the output of the transformation (see Fig. 6.1). The transformation is defined in terms of a dependency tree, but the model does not have access to it. This encourages the model to strengthen its representation of syntax and means that the model can be applied to downstream tasks without requiring syntax trees. Analogously to SIP in Chapter 5, we assume that gold-standard descriptions of transformations are not available during fine-tuning and use a prefix of

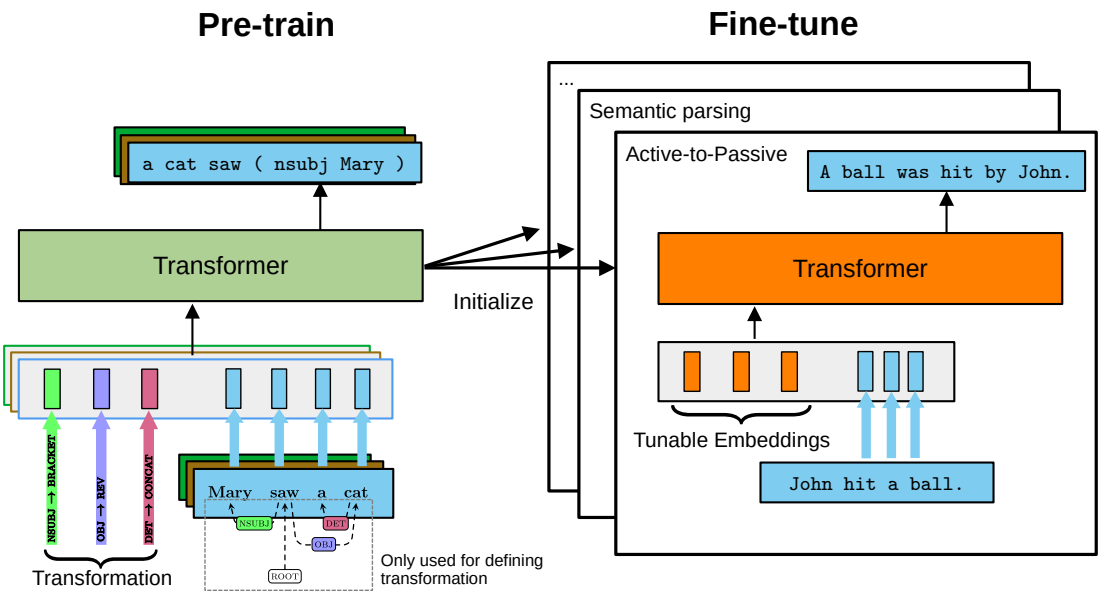


Figure 6.1: Left: Intermediate pretraining of a Transformer to perform syntactic transformations specified in the prefix; the syntax tree forms the basis of the transformation but is *not* given to the model. Right: fine-tuning the Transformer and the prefix on a downstream task. Note that parse trees are *not* required for fine-tuning or inference on downstream tasks. Tunable parameters are represented in orange.

embeddings that are fine-tuned with the rest of the model.

There are two significant differences in the setup compared to Chapter 5. First, we need to preserve more of the knowledge acquired during the original pretraining. The character-level downstream tasks we considered in Chapter 5 did not rely much on knowledge from the original pretraining because the downstream tasks were either synthetic or from low-resource languages for which the original pretraining provides a somewhat limited signal anyway. In contrast, for syntax-sensitive sequence-to-sequence tasks such as semantic parsing, preserving what was acquired during pretraining is crucial. For example, the model should still be able to identify paraphrases after our additional pretraining.

Second, in the case of FSTs, the input during pretraining provides both the FST and the input string, which makes the ground truth output a deterministic function of the input. Here, the output is not a deterministic function of the input because there can be ambiguity about the syntactic structure of the sentence. Making the output deterministic would require providing the model with the syntax tree as part of the input. This is undesirable because it would require a parser for fine-tuning and at inference time on the downstream tasks. In addition, it could discourage the model from improving its representations of syntax acquired during the original pretraining and instead lead to focusing entirely on performing transformations.

Contributions In this chapter, we extend the methodology from Chapter 5 and propose an intermediate pretraining method that strengthens the structural inductive bias of an already pretrained sequence-to-sequence model. This method improves few-shot performance for syntax-dependent tasks, such as chunking, and improves structural

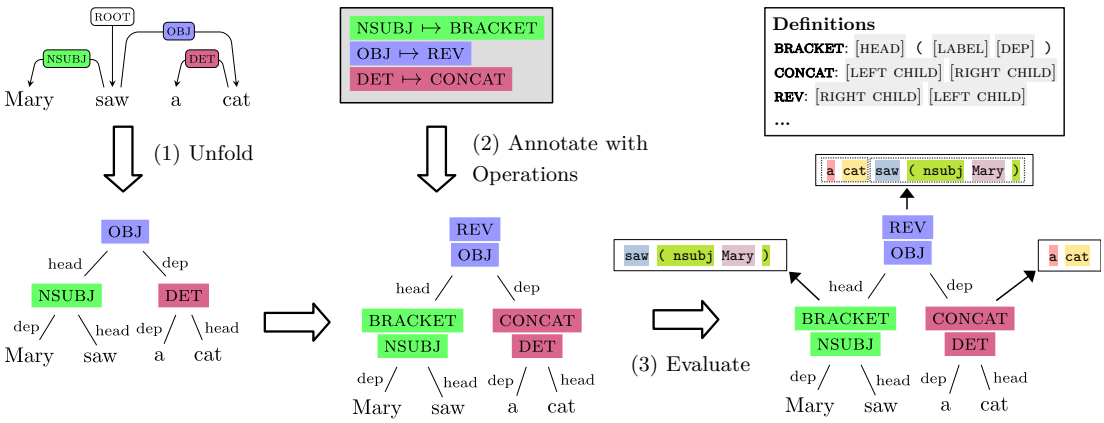


Figure 6.2: Our procedure of applying a syntactic transformation specified as edgewise transformations (grey box): (1) recursively unfolding a dependency tree into a binary tree where dependency labels serve as labels of internal nodes, (2) annotation dependency relations with edgewise transformations, (3), recursive evaluation of the edgewise transformations with partial results shown.

generalization in the context of semantic parsing.

Analysis of the pretrained model shows that it uses attention heads to track what transformation needs to be applied to which input token, and that these heads tend to follow syntactic patterns. In addition, we find that fine-tuning reuses these attention heads, suggesting that the model can leverage the transformations acquired during pretraining.

6.1 Strengthening Structural Inductive Biases

Standard pretraining objectives, e.g. with denoising objectives (Raffel et al., 2020), encourage models to acquire syntactic knowledge but provide little information about syntactic *transformations*, which are central to many syntactic and semantic seq2seq tasks. Our research hypothesis is that intermediate pretraining to perform transformations of syntax trees encourages the model to (i) strengthen its representations of the syntactic categories to which transformations can be applied (e.g. subjects, objects) and (ii) acquire *reusable* dynamics of transformations that are useful for downstream applications. By providing an explicit description of the transformation as a prefix, different transformations that the model has learned during pretraining can be 'activated' by the right choice of prefix. For this reason, we also fine-tune the model with a prefix of tunable embeddings to make it easy to leverage these transformations on downstream tasks similar to SIP (Chapter 5).

In addition to learning about transformations of trees, we also want the model to incorporate knowledge about the syntax of the underlying language (i.e. English, in this case). Hence, we do not provide syntax trees to the model during training, which also enables us to perform inference and fine-tuning without a parser.

Name	Definition	Example
CONCAT	LEFT CHILD RIGHT CHILD	Mary saw a cat
REV	RIGHT CHILD LEFT CHILD	a cat Mary saw
CONCAT-REL	LEFT CHILD LABEL RIGHT CHILD	Mary saw obj a cat
REV-REL	RIGHT CHILD LABEL LEFT CHILD	a cat obj Mary saw
BRACKET	HEAD (LABEL DEP)	Mary saw (obj a cat)
BR-INVERT	DEP (LABEL by HEAD)	a cat (obj by Mary saw)
BRACKET-2	(HEAD LABEL DEP)	(Mary saw obj a cat)
BRACKET-2-INV	(DEP LABEL HEAD)	(a cat obj Mary saw)
BRACKET-3	HEAD (DEP)	Mary saw (a cat)
BRACKET-4	HEAD LABEL (DEP)	Mary saw obj (a cat)
BRACKET-5	HEAD (LABEL DEP) if HEAD has no other BRACKET-5 arguments HEAD (LABEL DEP if DEP is the first BRACKET-5 argument HEAD , LABEL DEP) if DEP is the last BRACKET-5 argument HEAD , LABEL DEP else	Mary saw (obj a cat)
TRIPLE	HEAD (HEAD.LEMMA LABEL DEP.LEMMA) DEP	Mary saw (see obj cat) a cat
TRIPLE-INV	HEAD (DEP.LEMMA LABEL by HEAD.LEMMA) DEP	Mary saw (cat obj by see) a cat
IGNORE-DEP	HEAD	Mary saw

Table 6.1: Operations we use for edgewise transformations. We show an example transformation for the sentence *Mary saw a cat* where `HEAD = Mary saw` and `DEP = a cat`. HEAD.LEMMA (DEP.LEMMA) refers to the lemma of the head (dependent) that the edge in question was unfolded from (in the example: saw $\xrightarrow{\text{obj}}$ cat). BRACKET-5 essentially concatenates the results of all BRACKET-5 children together using a comma as joining element, and surrounds this with one matching pair of brackets, such as in `saw ( nsubj Mary , obj John )`. For an operation ℓ, we call a subtree t an ℓ *argument* in the unfolded and annotated binary tree if t resulted from unfolding a node in the dependency tree that had an incoming edge that was annotated with the operation ℓ. For example, in Fig. 6.2, the subtree corresponding to 'a cat' is a REV argument since 'cat' has an incoming OBJ edge in the dependency tree that was annotated with REV.

6.1.1 Syntactic Transformations

Our goal in designing the transformations is to create a family of syntactic transformations that resemble a broad class of real downstream transformations. We base our syntactic transformations on Universal Dependency trees (de Marneffe et al., 2021), and provide an overview in Fig. 6.2. Each transformation is fully specified by a set of *edgewise transformations* that assign a binary string operation (e.g. BRACKET) to a dependency relation (e.g. NSUBJ).

Applying a syntactic transformation to a dependency tree is a three-step process: First, we *unfold* the dependency tree into a binary 'phrase-structure'-like tree, where the dependency labels act as labels of the internal nodes.[1] This is necessary because our operations are binary, and we need a binary tree along which we can evaluate the operations. Second, we annotate the dependency labels with the corresponding operations according to the edgewise transformations. Finally, we recursively evaluate each operation in the resulting expression tree, yielding a single output string.

[1] Xia and Palmer (2001) explore related and more linguistically nuanced conversions from dependency to phrase structure trees. It is unclear if such unfolding procedures would further strengthen inductive biases. Here, we deliberately keep the unfolding simple and use a language-agnostic approach; the order in which dependents of a word are processed is fully determined by their order in the sentence.

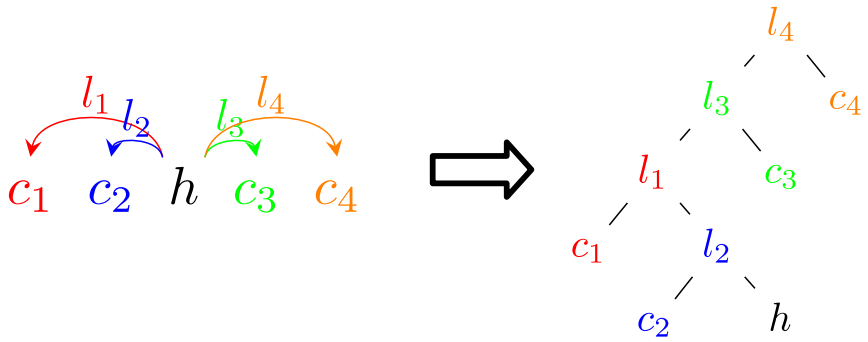


Figure 6.3: Unfolding a head h and its children. We first unfold the left children in inside-out order, followed by the right children, again in inside-out order.

Unfolding Unfolding replaces a head and its dependents with a binarized tree, as shown in Fig. 6.3. This procedure is applied bottom-up to all nodes in the tree. For example, the dependency subtree of 'a cat' unfolds to the tree $\text{DET}(a, cat)$, after which 'saw' is unfolded, leading to the final unfolded result in Fig. 6.2. Unfolding a node without children (e.g. 'Mary') simply retains that node.

Operations In order to have a wide range of syntactic transformations, we design an inventory of 14 operations (see Table 6.1). While the selection of operations is ultimately arbitrary, the goal is to use operations that cover many potentially useful transformations for downstream tasks. One such class of transformations are linearizations of dependency trees such as `saw ( nsubj Mary , obj John )`, where a single set of brackets surrounds all the children of the head. This requires an operation that can open a single bracket when applied to `saw` and `Mary` but also a guarantee that this bracket will eventually be closed. BRACKET-5 is designed to take care of such transformations and ensure well-balanced outputs. Annotating all dependency labels with BRACKET-5 results in a transformation that produces a string representation of a dependency tree and annotating all labels with CONCAT acts as the identity function, provided the underlying dependency tree is projective.

Our syntactic transformations could in principle also be implemented with synchronous grammars (Lewis and Stearns, 1968; Chiang, 2007). In contrast to our transformations, the rules of synchronous grammars generate not only the output but also the input. As a result, synchronous grammars would yield a much more verbose representation with potentially overly specific rules that depend on specific phrase structure rules to occur. In earlier experiments, we found more than 10,000 (binarized) phrase structure rules to occur in a corpus of 300,000 sentences. Due to a heavy Zipfian tail, most of these rules only appear a few times. Hence, many rules may not be at a reasonable level of abstraction and it might also be difficult to learn transformations for the rules in the tail of the distribution. In contrast, there are only about 70 UD relations, making transformations on UD trees more manageable.

6.1.2 Intermediate Pretraining

During intermediate pretraining, the model is given a sentence and a set of edgewise transformations that determine the overall transformation. The objective is to predict

what the transformation does to the parse tree of the sentence. The input to the Transformer is a sequence of vectors from $\mathbb{R}^d$, which consist of a prefix that represents the edgewise transformations and a suffix comprised of the embeddings of the input tokens:

$$\underbrace{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k}_{\text{Transformation}}, \underbrace{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n}_{\text{Sentence}}$$

Each $\mathbf{h}_i$ encodes an edgewise transformation $R \mapsto f$ by simple addition of embeddings:

$$\mathbf{h}_i = \text{EMBED}_{\text{Label}}(R) + \text{EMBED}_{\text{Transformation}}(f)$$

The training objective is the log likelihood of the correct output of the transformation, and we start from T5-base (Raffel et al., 2020) that has already been pretrained. Note that the dependency tree is *not* provided to the model to encourage it to reuse and strengthen its syntactic knowledge.

We also want to preserve the existing (syntactic) knowledge of the pretrained T5 model, e.g. to make it easy to insert the right auxiliary verb form when transforming a sentence from active to passive (see Fig. 6.1). To help preserve this, our second pretraining objective is the span-denoising objective that T5 was originally pretrained with. We train the model by alternating between gradient descent steps on the two objectives.

Data Generation We construct random syntactic transformations for a small fraction of the C4 corpus, which T5 was originally pretrained on. We tag, parse and lemmatize 2.1 million sentences with a total of around 39 million word forms using trankit (Nguyen et al., 2021). We create two random transformations per parsed sentence, resulting in approximately 4.2 million pretraining instances.

To construct a random syntactic transformation for a given sentence, we sample dependency relations present in that sentence and some additional dependency relations that are *not* present in the sentence to a maximum total of 20 relations. We uniformly sample an operation for each relation to create edgewise transformations. Relations that are not chosen by sampling are implicitly assigned the operation CONCAT. While the relations that are not present in the sentence have no bearing on the output of the transformation, we include them in the description to expose the model to a more general description that applies to a broader range of sentences.

6.1.3 Fine-Tuning

After pretraining, we apply our model to different downstream tasks via fine-tuning. Mirroring the pretraining, we replace the transformation encoding with a sequence of tunable embeddings. That is, the input to the model is a sequence of vectors:

$$\underbrace{\mathbf{h}'_1, \mathbf{h}'_2, \ldots, \mathbf{h}'_k}_{\text{Tunable embeddings}}, \underbrace{\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n}_{\text{Sentence}}$$

where $\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_n$ are the embeddings of the input tokens, $\mathbf{h}'_i \in \mathbb{R}^d$ are the tunable embeddings and k is a hyperparameter. The embeddings $\mathbf{h}'_i$ are initialized to the average

of the encoding of multiple transformations from the pretraining phase. Because the tuneable embeddings are trained on the downstream task, we hypothesize that they can be used to 'activate' transformations that help with the particular downstream task. We fine-tune all model parameters and use a higher learning rate for the prefix.

6.2 Evaluation

We evaluate on syntactic and semantic tasks for which a structural inductive bias should be helpful. Specifically, we consider learning from a small amount of task-specific data (few-shot learning) and structural generalization beyond the training distribution to unseen combinations of known phrases, novel syntactic phenomena and deeper recursion than seen during training.

Baselines

For a fair comparison, we compare our method (STEP, for **S**yntactic **T**ransformation **E**nhanced **P**re-training) with fine-tuning other seq2seq models based on T5-base (Raffel et al., 2020) that were further pretrained on the parsed corpus (Section 6.1.2) in different ways:

T5+Dep Parse is pretrained to predict a linearized dependency tree of the input, e.g. *Mary saw a cat* → `( saw nsubj Mary obj ( cat det a ) )`. Hence, this model incorporates syntactic information about English dependency trees but has limited exposure to how this information can be used other than to produce a parse tree.

Simple STEP is a simplified version of STEP, where we always assign the same edgewise transformation to *all* dependency relations. Consequently, the number of possible syntactic transformations is exactly the number of binary string operations we define (Table 6.1). We remove IGNORE-DEP because it would result in an output string with a single token. We use a special token in the prefix to indicate which transformation should be applied.

Analogously to STEP, the models above were pretrained with their specific pretraining objective and the original span denoising objective of T5.

6.2.1 Syntactic Tasks

To see if the structural inductive bias for syntactic transformations has been imparted, we first evaluate if our synthetic transformations transfer to realistic syntactic transformations. In particular, we focus on few-shot scenarios.

We revisit three structural transformation tasks we evaluated on in Chapter 3: passivization (see Fig. 6.1), emphasis of a designated adjective[2] and emphasis of a designated verb[3]. These transformations tasks were identified as challenging by Lyu et al. (2021b) because only several hundreds of training examples are available. We make

[2] *The French analysis goes further → The analysis that goes further is French*

[3] *corporate profits may also dip initially → the dipping of corporate profits may also happen initially*

Task	Model	Acc ↑	BLEU ↑	TER ↓
Verb emphasis	T5	3.5	41.7	46.7
	T5+Dep Parse	3.3	40.1	48.2
	Simple STEP	**3.6**	40.5	47.0
	STEP	3.4	**41.8**	**45.6**
Adj. emphasis	T5	7.3	47.6	38.3
	T5+Dep Parse	7.7	45.8	40.4
	Simple STEP	9.8	48.3	37.5
	STEP	**10.9**	**52.3**	**33.5**
Passivization	T5	40.2	73.7	18.3
	T5+Dep Parse	45.0	76.8	15.5
	Simple STEP	46.8	78.4	13.6
	STEP	**57.9**	**84.8**	**8.4**

Table 6.2: Evaluation on 100-shot **syntactic transformation** tasks. We report averages of 10 draws of 100 training examples each.

Model	Acc ↑	F ↑
T5	34.4$_{\pm 0.8}$	87.4$_{\pm 0.6}$
T5+Dep Parse	39.9$_{\pm 2.1}$	90.0$_{\pm 0.6}$
Simple STEP	45.3$_{\pm 2.0}$	90.6$_{\pm 0.6}$
STEP	**53.8**$_{\pm 2.1}$	**93.2**$_{\pm 0.5}$

Table 6.3: Means and standard deviations on **chunking** across 5 random draws of 100 training examples. Accuracy is exact match, i.e. predicting *all* chunks correctly.

the setup more challenging in this section by restricting the training data to only 100 training examples.

We report results in Table 6.2 using exact match accuracy, BLEU (Papineni et al., 2002) and TER (Snover et al., 2006), a normalized edit distance.[4] Using dependency parsing as the intermediate pretraining task (T5+Dep Parse) is already beneficial for passivization but somewhat deteriorates performance on adjective emphasis both in terms of BLEU and TER. Simple STEP improves on this with small gains on both adjective emphasis and small additional improvements for passivization. STEP performs best, outperforming the baselines by a sizable margin of 3.5 and 6 points BLEU on the adjective emphasis and passivization tasks. However, STEP and T5 perform similarly on the verb emphasis task, and we hypothesize STEP has difficulties reusing the transformations acquired during pretraining (see also Section 6.3).

[4]These results were obtained with different implementations of the BLEU metrics than those in Chapter 3 because of version incompatibilities with other packages. Unfortunately, this makes these two sets of results not quite comparable since implementations tend to give somewhat different results e.g. because of tokenization (Post, 2018).

Model	iid	Length
F→R	74.2±1.4	35.4±4.0
Tag & Permute	76.6±1.7	**41.4**±13.5
T5	**85.5**±1.4	32.0±4.4
T5+Dep Parse	84.9±0.4	27.5±2.0
Simple STEP	82.9±1.0	30.4±1.5
STEP	84.2±1.7	38.4±2.7

Table 6.4: Means and standard deviations of model accuracy for semantic parsing on **ATIS** for 5 random seeds. *Tag & Permute* is the model from Chapter 4 with pretrained RoBERTa and F→R is the model from Chapter 3 without pretraining but using grammar-based decoding.

Chunking We also evaluate on chunking (Tjong Kim Sang and Buchholz, 2000) phrased as a seq2seq task.[5] Different variants of chunking play an important role in information extraction (Dong et al., 2023), which often has to rely on small domain-specific corpora (Bassignana and Plank, 2022). Few-shot learning of chunking is hence relevant and particularly interesting in our setup because it requires models to predict phrase categories (e.g. NPs) that do not exist in our pretraining approach based on dependency trees.

We report results in Table 6.3. While using parsing as intermediate pretraining is already helpful in comparison to T5, STEP improves accuracy even further and outperforms T5 by almost 20 percentage points for exact match accuracy. Simple STEP also shows some improvements over T5+Dep Parse but is again outperformed by STEP.

Overall, this shows that STEP strengthens the inductive bias for realistic syntactic transformations. The improvements of STEP over T5 cannot be attributed alone to the prediction of dependency trees during pretraining as T5+Dep Parse performs worse. Pretraining the model with a narrow set of transformations (Simple STEP) is not as effective as a large set of transformations with explicit descriptions. We hypothesize that the improvements of STEP can be attributed partly to the reusability of the transformations during fine-tuning, which we analyze in Section 6.3.

6.2.2 Semantic Tasks

Semantic parsing, i.e. constructing a logical form from a sentence, can be seen as a particular transformation of the syntactic structure (Montague, 1970). Hence, we expect an inductive bias for syntactic transformations to be helpful for semantic parsing, particularly for *structural generalization*, i.e. extrapolation to unseen combinations of phrases, longer examples and deeper recursion.

ATIS (Dahl et al., 1994) is a semantic parsing dataset with questions about a flight database annotated with executable logical forms. We follow previous work in using

[5] *The chairman promised Mr. Stone a decision → (NP The chairman) (VP promised) (NP Mr. Stone) (NP a decision)*

Model	Modifiers	Novel Gaps	Wh-Questions	Recursion	Overall
*AM-Parser**	69.6	20.7	57.0	**99.9**	70.8±4.3
T5	79.5±2.5	81.4±6.2	82.4±2.3	71.1±1.2	77.6±1.4
T5+Dep Parse	82.8±3.1	**86.8**±6.6	78.8±4.4	72.2±2.0	78.3±2.1
Simple STEP	**88.6**±0.4	44.8±12.6	84.2±2.1	79.0±1.6	78.8±1.9
STEP	87.5±0.4	47.6±14.1	**84.8**±4.2	79.8±0.9	**79.3**±2.3

Table 6.5: Structural generalization on the variable-free meaning representation of **SLOG** based on 10 random seeds. Specialized architectures are represented with italics. * The AM-Parser uses a semantically more expressive meaning representation formalism based on graphs.

the variable-free FunQL version (Guo et al., 2020). However, the order of the conjuncts in the logical form tends to be somewhat unsystematic and often does not correspond to the linear order in the question. Hence, we use a pre-processing step to re-order conjuncts based on automatic alignments (see Appendix E.1). As in previous chapters, we evaluate in two setups: (i) the standard iid split and (ii) a length split, where a model is shown logical forms with up to three conjuncts during training and has to generalize to sentences that require four or more conjuncts in the logical form.

Results are shown in Table 6.4. Tag & Permute is the specialized architecture from Chapter 4 and achieves the highest accuracy on the length split. Among the non-specialized architectures, STEP performs best, narrowing the gap to the specialized model. Interestingly, T5+Dep Parse and Simple STEP perform somewhat worse than plain T5.

SLOG (Li et al., 2023a) is a synthetic benchmark that tests models on 17 different structural generalizations grouped into 4 categories: using modifiers in novel positions (e.g. PPs only modify objects during training but modify subjects at test time), novel gap positions (e.g. wh-question for an indirect object, with the training data covering wh-questions for subjects and objects), wh-questions in novel syntactic contexts (e.g. wh-questions combined with passive instead of active voice) and novel recursion depth (e.g. deeper PP recursion).

We report aggregated results in Table 6.5 and results for all 17 generalization cases in Table E.2. Overall, STEP performs best but performance on the different categories varies considerably between the approaches. STEP and Simple STEP outperform T5 on all but one category, with considerable margins for the novel modifier positions and unseen recursion depths. However, they considerably underperform in the case of the novel gap positions. T5+Dep Parse performs more similarly to T5 with typically modest improvements across the categories.

We also compare with the AM-Parser (Groschwitz et al., 2018; Weißenhorn et al., 2022), a specialized approach for semantic parsing. It performs worse than the seq2seq models on most categories, except for recursion, where it achieves close to perfect accuracy. Here, STEP reduces the gap between the more general seq2seq models and the specialized AM-Parser.

In Table 6.6 we focus on a subset of the results, namely for generalization to deeper

Generalization	STEP	Simple STEP	T5	T5+Dep Parse
Deeper CP tail recursion	**74.5**$_{\pm 4.2}$	73.0$_{\pm 8.4}$	42.5$_{\pm 4.4}$	50.9$_{\pm 9.4}$
Deeper PP recursion	**87.3**$_{\pm 2.5}$	78.5$_{\pm 4.4}$	75.0$_{\pm 4.5}$	70.8$_{\pm 5.9}$
Deeper center embedding	17.3$_{\pm 2.8}$	**22.7**$_{\pm 2.2}$	8.9$_{\pm 0.4}$	11.5$_{\pm 2.1}$

Table 6.6: Results on **SLOG** for generalization to deeper recursion depths (5 - 12). This excludes evaluations on unseen but shallow recursion depths, which all models solve perfectly.

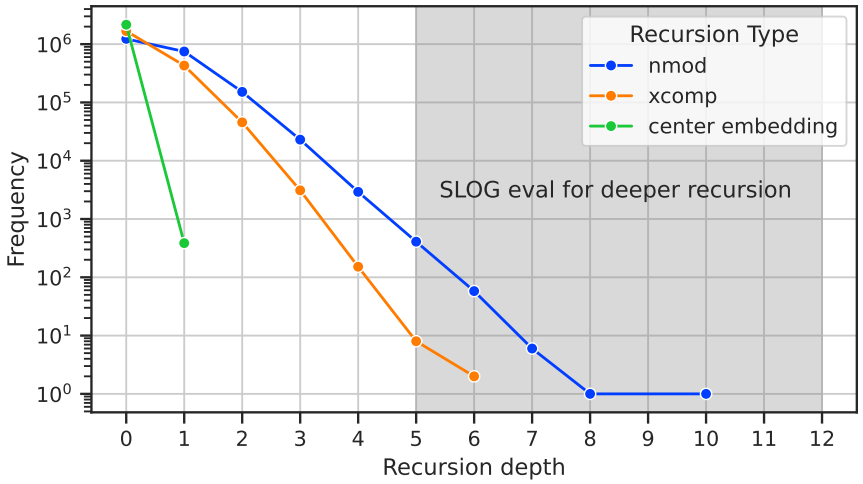


Figure 6.4: Frequency of recursion depths in our parsed corpus (Section 6.1.2) according to the dependency trees produced by trankit. Note that the y-axis is in log scale. In phrase structure terminology (e.g. on SLOG), *xcomp* recursion includes CP recursion and *nmod* recursion includes PP recursion.

recursion depths of 5 or more. All models perform best on deeper PP recursion and struggle most with center embedding, with STEP performing best on average.

Both STEP and Simple STEP considerably improve over T5 on generalization to deeper CP recursion and to a much lesser extent also on center embedding. Interestingly, this happens despite very limited empirical support for these phenomena in our intermediate pretraining data (Fig. 6.4). We counted only 10 sentences in the corpus with paths containing 5 or more *xcomp* edges, which indicate CP recursion depth. As for center embedding, we were unable to find even a single instance with depth two or more in our corpus.[6] These results provide an indication that pretraining can provide structural inductive biases somewhat beyond the syntactic phenomena encountered in the data.

[6]We use an approximate characterization of center embedding in the context of dependency trees as relative clauses whose subject again has a relative clause. We exclude copulas because UD treats them differently from full verbs.

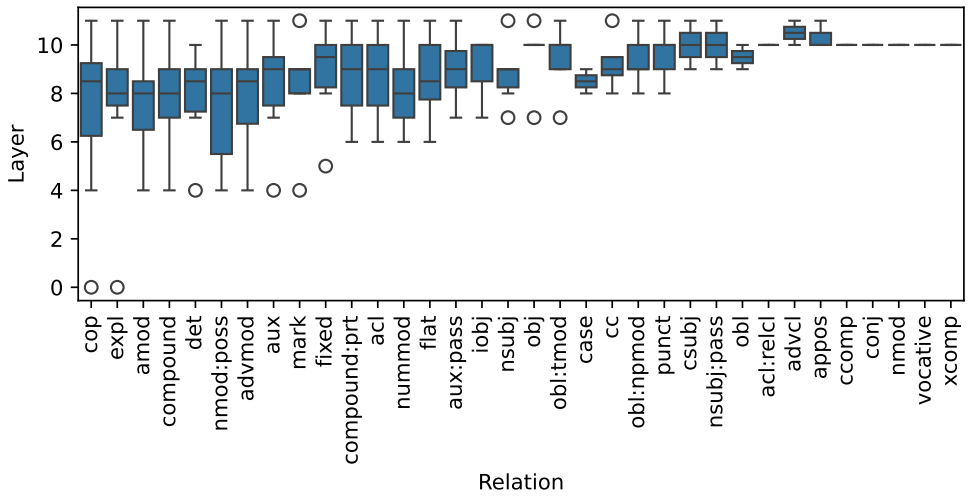


Figure 6.5: Distribution of location of look-up heads we identified per UD relation.

6.3 Analysis

Our research hypothesis is that our intermediate pretraining encourages the model to acquire reusable dynamics of syntactic transformations that can be leveraged during fine-tuning. In this section, we analyze the representations used by our model after its intermediate pretraining, and to what degree they are reused during fine-tuning.

Analysis of Pretrained Model

We first investigate how the model processes the transformation encoded in the prefix. The model has to attend to the prefix to gather information about which edgewise transformation needs to be applied to which input token. We call an attention head a *transformation look-up head* if it consistently attends to the prefix.

We find that some transformation look-up heads are interpretable and follow syntactic patterns. For example, when head 6 in layer 10 computes the attention distribution for a token that is an object in the sentence (i.e. *cat* in Fig. 6.1), it focuses the attention on the edgewise transformation that describes how to process objects (i.e. $\text{OBJ} \mapsto \text{REV}$).

Identifying interpretable look-up heads We consider each attention head H and dependency relation R separately. For a sample of sentences with corresponding transformations, we count how many times the following conditions are true: (i) the instance has an edgewise transformation involving R, (ii) a token x_i has an incoming edge labelled R and (iii) H focuses at least 50% of its attention from x_i on a single position j. If in over 70% of cases, position j refers to the edgewise transformation of R then we call H a transformation look-up head for the dependency relation R.

We find that there are often multiple transformation look-up heads per dependency relation. For example, we identify 7 look-up heads for `amod`. These interpretable heads are typically located in the mid or higher layers (see Fig. 6.5), which is expected

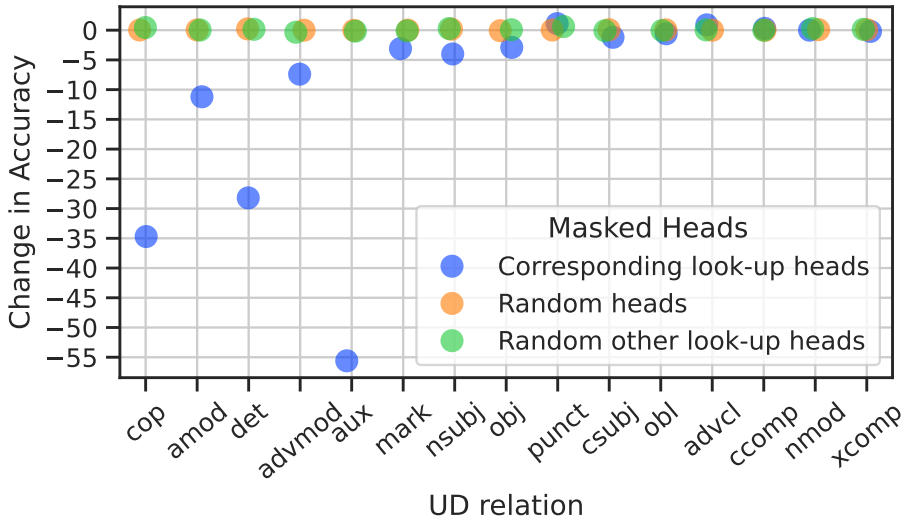


Figure 6.6: Change in accuracy of predicting the output of edgewise transformations when masking different attention heads. We show accuracy relative to no masking. The effect of masking random heads and random other look-up heads tend to be very similar, leading to overlap in the plot.

because the model first needs to identify the syntactic role each token has.

Intervening on look-up heads Next, we verify that the transformation look-up heads we identified contribute to the model prediction with an interventional analysis. We evaluate the role of the transformation look-up heads separately for different dependency relations: if the heads $H_1, H_2, \ldots H_n$ play an important role in performing transformations for dependency relation R, then masking all of them should reduce accuracy for instances with an edgewise transformation for R. As a comparison, we also evaluate (i) masking n randomly chosen heads, and (ii) masking n randomly chosen heads that are transformation look-up heads, but not for R. Since the look-up heads can also have other functions within the model, we only mask out the attention to the prefix. When masking randomly selected attention heads, we ensure comparability by masking a random subset of tokens equal to the length of the prefix.

We show the results in Fig. 6.6. Masking transformation look-up heads reduces accuracy for many dependency relations while masking other transformation look-up heads or random heads has very little impact. This provides evidence that the identified heads play an important role within the model. For some relations (e.g. punct, advcl, nmod), masking the respective look-up heads does not reduce accuracy, suggesting that responsibility for these relations is more spread out through the network.

Analysis of Fine-Tuned Models

How does a model pretrained with STEP learn during fine-tuning? We hypothesize that the pretraining provides a scaffolding, which finetuning can build upon. In particular, we expect that aspects of the downstream task that can be expressed with our transfor-

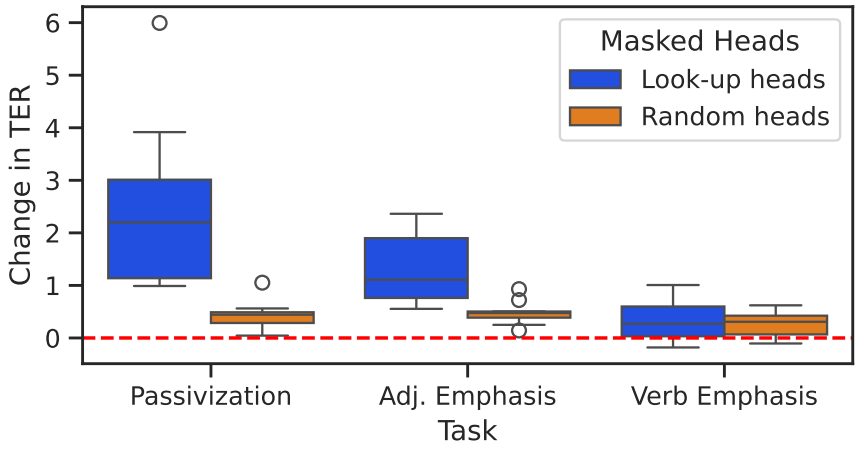


Figure 6.7: Effect of masking look-up heads of models that have been fine-tuned on downstream syntactic tasks. For each task, we show the distribution for the 10 fine-tuned models from Section 6.2.1.

mations to be captured in the same way as during pretraining, i.e. with the prefix and with the transformation look-up heads.

Masking look-up heads after fine-tuning To test this hypothesis, we create 10 new synthetic transformation tasks with 5 edgewise transformations each and fine-tune the model (Section 6.1.3). Then, we take the attention heads we identified in Section 6.3 and repeat the masking intervention, i.e. we mask the attention to the prefix of all look-up heads for the dependency relations involved in the edgewise transformations.

Masking the look-up heads of the dependency relations involved in the transformations leads to an average drop in accuracy of 30 percentage points (see also Fig. E.1), whereas masking random look-up heads reduces accuracy by less than one point.

Reading off transformations from fine-tuned prefix The strong impact of masking the look-up heads in the fine-tuned model suggests that it uses the tunable prefix to encode task-specific information about which edgewise transformation to apply. To gain more insight into this, we try to *extract* edgewise transformations from the fine-tuned prefix: For each vector $\mathbf{h}'$ in the prefix, we find the edgewise transformation whose embedding is closest to $\mathbf{h}'$ in terms of cosine similarity. In this manner, we can read off a candidate for the transformation that the model might be using under the hood and compare it to the correct transformation that generated the synthetic data. We find that the edgewise transformations extracted in this way agree with the gold edgewise transformations with an average F-score of ≈ 77. This provides evidence for the hypothesis that the model re-uses the transformations learned during pretraining and 'activates' them with the prefix.

Fine-tuning on realistic transformations Finally, we investigate the role of transformation look-up heads in models fine-tuned on realistic syntactic transformations outside of the pretraining distribution (see Section 6.2.1). Since there are no ground

truth edgewise transformations in this case, we mask the attention to the prefix of *all* transformation look-up heads and compare with masking an equal number of random heads. Fig. 6.7 shows that masking the transformation look-up heads deteriorates outputs more than masking random heads for passivization and the adjective emphasis task. However, results are comparable for verb emphasis. This is in line with our findings that STEP improves over T5 for passivization and adjective emphasis but not for verb emphasis, and suggests that the lack of improvement for the verb emphasis task could be due to difficulties in reusing the transformations seen during intermediate pretraining.

6.4 Related Work

In recent years, several works have explored injecting syntactic knowledge through pretraining or multi-task learning. Most of these approaches have focused on learning contextualized word representations with task-specific layers on top and have shown that syntactic knowledge can improve tasks such as parsing (Zhou et al., 2020), semantic role labeling (Swayamdipta et al., 2018; Zhou et al., 2020), coreference resolution (Swayamdipta et al., 2018), grammatical error detection (Zhang et al., 2022a) and relation extraction (Bassignana et al., 2023). Because these works focus on encoder-only models, they cannot be directly applied to sequence-to-sequence tasks.

In the context of sequence-to-sequence models, Xu et al. (2020) focus on broad-coverage semantic parsing and explore pretraining on multiple tasks, including constituency parsing with linearized trees. Finally, Mulligan et al. (2021) present proof-of-concept experiments in which they show that multi-task learning of syntactic transformations can provide a bias towards hierarchical generalizations when data with a hierarchical structure is provided for the auxiliary tasks. In contrast to our work, they consider a setup with training from scratch using multi-task learning rather than pretraining. They only use three manually selected syntactic transformations and focus entirely on synthetic data.

To our knowledge, we are the first to explore pretraining with a large space of synthetic transformations of syntax trees. In addition, rather than using an atomic and unstructured task id to distinguish different tasks (Johnson et al., 2017; Xu et al., 2020; Mulligan et al., 2021), we provide the model with an explicit description of the transformation.

Another line of work has also used synthetic data to augment the training data, e.g. to improve compositional generalization (Andreas, 2020; Qiu et al., 2022a; Yang et al., 2022; Li et al., 2023d; Yao and Koller, 2024; Cazzaro et al., 2024) (see also Section 2.4). Data augmentation aims to generate more data for the task at hand and plug gaps in the data collection. Hence, it is task-specific and needs to be repeated for every new task. Data augmentation also inherently risks introducing errors and noise to the training data. In contrast, our approach pretrains a model once to perform syntactic transformations and can then be fine-tuned for different downstream tasks.

6.5 Conclusion

We extend the approach we introduced in Chapter 5 and propose STEP, a method of strengthening the structural inductive bias of a Transformer by pretraining the model to perform syntactic transformations based on dependency trees. Effectively, the transformations demonstrate the principles of compositionality and recursion to a model. We show that this results in a better few-shot performance for syntax-dependent sequence-to-sequence tasks, and improves structural generalization for semantic parsing. This demonstrates the viability of our framework beyond the use case of Finite State Transducers.

Analysis of the pretrained model shows that it uses attention heads to track what transformation needs to be applied to which input token, and that these heads tend to follow syntactic patterns. In addition, we find that fine-tuning reuses these attention heads, suggesting that the model can leverage the transformations acquired during pretraining.

Limitations The structural inductive bias that is emphasized by our intermediate pretraining depends on the inventory of operations. Due to the computational cost of pretraining, we did not systematically explore which set of operations performs best, or which operations do not provide much benefit and could be omitted.

In this work, we focus on a moderately sized encoder-decoder model (T5) and do not investigate large decoder-only models. However, we conjecture that the decoder-only architecture is compatible with our approach. While larger models trained on more data might have stronger syntactic capabilities and therefore likely have less of a need to address issues of systematic generalization overall, we think our approach could be helpful in addressing remaining issues with structural generalization.

Our analysis focuses on the encoder of the Transformer, and on the transformation look-up heads in particular. It remains much less clear how the decoder of our model operates internally and future work could investigate its role in applying syntactic transformations.

Future work could also apply the idea behind STEP to more languages and more syntactic or semantic formalisms, including those with variables. Another important direction is better preservation of the knowledge acquired during the original pretraining.

Chapter 7

Conclusion

In this thesis, we have explored methods to enhance the inductive biases of sequence-to-sequence models for linguistic structure and linguistic principles, in particular recursion and compositionality. These inductive biases are intended to help models learn effectively from small amounts of task-specific training data and make structural generalizations beyond the training distribution, e.g. for unseen combinations of phrases and deeper recursion. We have taken two different approaches to incorporating structural inductive biases in parts I and II, and demonstrated their effectiveness.

Part I In Part I, we emphasize the structural inductive biases that are useful for semantic parsing. The theme of this part are structured architectures that phrase sequence-to-sequence problems as a composition of differentiable edit operations. These approaches explicitly model correspondences between fragments of the input and the output, inspired by grammar-based approaches.

In Chapter 3, we propose a sequence-to-sequence model that composes a new differentiable fertility layer with a reordering layer from previous work. The fertility layer is inspired by statistical machine translation and decides for every input token, how many output tokens it corresponds to. Subsequently, the reordering layer determines the order of the output. These discrete edit operations are turned into 'soft', differentiable edit operations by using expected values, which are computed with dynamic programming. The fertility layer and its dynamic program form the core of the technical contribution of this chapter, and we demonstrate that composing the fertility and reordering layer into a sequence-to-sequence model is effective for structural generalization. While the method is effective for generalization, we observe that the reordering layer is computationally expensive and can be overly restrictive, limiting its applicability.

In Chapter 4, we address these issues with a new expressive permutation model that can predict any permutation. We parameterize our new permutation model by giving a score to every pair of tokens how much they prefer to be adjacent in the output. Finding the highest-scoring permutation under this model is NP-hard. We approximate this optimization problem by relaxing it to a linear program and make it differentiable with an additional entropy regularization term. We propose a method for solving this optimization problem that is an instance of Bregman's method (Bregman, 1967) and

a generalization of Sinkhorn's algorithm (Sinkhorn, 1964). Despite the NP-hardness of inference, empirically, this approach is computationally cheaper than the reordering we used in Chapter 3. We show that the resulting model performs well for structural generalization on several semantic parsing datasets. The success of this approach demonstrates for the first time that a model with an inductive bias that does not explicitly rely on trees can achieve high accuracy on generalization to deeper recursion.

Part II In Part II, we widened the scope of the kinds of inductive biases we consider. The overall contribution of the second part is a general framework for injecting inductive biases into a standard Transformer without modifying its architecture. The key idea behind this framework is that we can often operationalize an inductive bias as a family of symbolic functions, and we can generate such functions and corresponding inputs and outputs automatically. This synthetic data is used to continue pretraining a model. We apply this framework to two setups, i.e. with different families of functions, and show that downstream tasks that are related to but outside the distribution over functions can benefit from the conferred inductive bias.

In Chapter 5, our family of functions are Finite State Transducers (FSTs). We pretrain a Transformer to *simulate* the behavior of FSTs by giving it a description of a randomly generated FST and an input string and train it to predict the corresponding output of the given FST. We demonstrate that this procedure gives the model a helpful inductive bias for NLP tasks that have been traditionally approached with FSTs, such as text editing and grapheme-to-phoneme conversion. In our analysis, we study how the inductive bias is injected and how the model can leverage the pretraining to succeed for FST-like tasks. Our probing experiments show that the model learns to internally simulate transitions between FST states using its hidden representations during pretraining. Interestingly, this happens without the model being explicitly trained to do so. In controlled fine-tuning experiments, we find that the model can leverage the machinery for simulating transitions between FST states, resulting in activations that suggest the model approaches the downstream task by simulating the transitions of the true underlying FST.

In Chapter 6 we apply the framework to transformations acting on syntax trees. We further pretrain a T5 model (Raffel et al., 2020) on a dataset of automatically generated syntactic transformations of English dependency trees. Given a description of the transformation and an input sentence, the model is pretrained to predict the output of the transformation without access to the underlying dependency tree. This procedure encourages the model to strengthen its representations of syntax and acquire reusable dynamics of syntactic transformations that can be leveraged for downstream tasks. Our experiments confirm that this improves structural generalization for semantic parsing and also helps with few-shot learning of syntactic tasks such as chunking. Our analysis focuses on the encoder and finds that our additional pretraining leads to attention heads keeping track which syntactic transformation needs to be applied to which token. We also find that the model can leverage these attention heads on downstream tasks after fine-tuning.

In comparison to our analysis in Chapter 5, it is much more difficult to test if the model internally simulates the transformations on syntax trees. In fact, there are reasons to be skeptical that the model has the expressive capacity to faithfully simulate

this process internally. In particular, Transformers have a computation graph with fixed depth, which is hard to reconcile with the recursive computations on syntax trees that may be significantly deeper than the depth of the model we used. From this perspective, we take the empirical successes we observed in Chapter 6 as a potentially encouraging signal that simulation may not have to be completely internalized to provide a useful inductive bias.

The methods we propose in the two parts change inductive biases by different means, either via the architecture (Part I) or via the weight initialization and fine-tuning procedure (Part II). As a consequence, they navigate trade-offs differently. The methods in Part I can achieve perfect generalization in some cases, as we demonstrated on generalizing to longer strings for simple synthetic mappings such as mirroring strings or repeating an input string twice. In Part II, we observe strong or very strong structural generalization, but perfect out-of-distribution generalization with 100% accuracy is quite rare even in favorable conditions. However, a clear advantage of the framework we developed in Part II is that it is much more general, easier to use, and models pretrained in this way are more versatile. For example, STEP from Chapter 6 could likely still be used for fine-tuning a summarization model, whereas the model in Chapter 4 makes too many independence assumptions to be useful in this context and would require considerable adaptation.

Despite all these differences in the nature of the technical contributions, the methods we introduced in the two parts can be seen as differentiable relaxations of discrete computational processes (Section 5.7). In the first part, the relaxations are hand-designed, while in the second part, the relaxations are learned from synthetic data and provide a softer guidance through the initial values of the parameters before fine-tuning.

Implications and Future Work

To conclude this thesis, we reflect on some of the implications of this thesis beyond the immediate contributions and what remains relevant given the rapidly evolving field that has undergone a paradigm shift with LLMs. Whilst many of the empirical results will quickly lose most of their significance, more general ideas will likely retain relevance.

New Applications of the Computational Methods The core technical contributions from Part I of this thesis, i.e. the fertility layer and the permutation model, are more general than the models with which we introduced them and could play a role as components in future methods. For instance, the fertility layer could be used to *compress* sequences by restricting fertilities to be 0 or 1, i.e. either copying a token to the output or dropping it. Transformer models require a quadratic amount of computation steps with respect to the sequence length, so compressing sequences can be beneficial to make models more computationally efficient (e.g. Nawrot et al. (2023)). The fertility layer could enable learning which tokens should be kept for more processing to reduce the impact on the quality of the output.

Permutations are an important object in discrete mathematics and are relevant in several contexts in NLP, most notably for the word order of sentences. The method we propose to predict permutations in this thesis could help study how order affects NLP models. In particular, our method can be used to obtain differentiable soft samples of permutations (with a perturb-and-MAP approach, Papandreou and Yuille (2011); Paulus et al. (2020)). Hence, it could be used to identify word orders with specific properties encoded by a loss function, for instance, finding word orders that flip the decision of a classifier. In principle, it could also be used to study how the ordering of in-context examples affects LLMs.

Computational Linguistics The findings and contributions of this thesis also raise interesting follow-up questions for computational linguistics, i.e. the study of language and its properties with computational methods. Related to our permutation model, one could, for instance, regard word orders attested in natural languages as an approximately optimal solution to an optimization problem and try to reverse-engineer the objective function, e.g. in terms of distances between dependencies and information locality (Hahn and Xu, 2022). Our differentiable permutation model might help with tackling this challenging optimization problem, or could at least provide a starting point.

Traditionally, trees have played a crucial role in computational linguistics for the conceptualization of syntax and its interface with semantics. This makes our finding interesting, that a semantic parsing model without an explicit notion of trees can perform well (although not perfectly) on generalization to deeper recursion. It would be interesting to extend this from a single observation on a synthetic corpus to natural data. If these findings were corroborated, it would provide a step towards identifying weaker inductive biases for learning how sentence-level meaning arises from lexical meaning.

Inductive Biases by Simulation for Cognitive Science Hierarchical Bayesian methods are a standard tool within the Cognitive Science community to test if specific hypothesized inductive biases can explain how humans learn from small amounts of data for specific small-scale tasks, such as concept learning (Griffiths et al., 2010). However, Bayesian models are computationally very expensive, and recent work has suggested meta learning with MAML (Finn, 2018) as an alternative. Here, meta learning is applied with synthetically generated data that conforms with the hypothesized inductive bias (McCoy et al., 2020; McCoy and Griffiths, 2023). While MAML and its variants are more efficient than Bayesian models and better suited to exploit the computational power of GPUs, it is still quite expensive and restricts the size of models that can be used and which kinds of hypotheses can be tested. The framework we propose in Part II could potentially address these challenges. The idea behind this framework is much more general than the two applications we have demonstrated, and the positive results we obtained in this thesis are encouraging that this framework could be applied more widely. First steps in that direction could aim to replicate previous experiments in this new framework and establish the relative strengths and weaknesses of the approaches.

Structural Generalization, Reasoning and LLMs For now, structural generalization remains an unsolved problem, despite recent progress, in particular for semantic parsing. Going forward, we think that structural generalization should not be viewed as limited to such a narrow range of tasks, or traditional NLP tasks for that matter. As LLMs are perceived by some as 'one-stop shops' for AI, they increasingly also face tasks with a strong algorithmic component, such as logical and algorithmic reasoning and planning. All these problems benefit from structural generalization from simple, common instances to a long tail of harder instances that require the same underlying algorithmic approach.

As we approach the end of naive scaling, where models are pretrained on virtually all high-quality text data available on the internet, methods similar to the ones in Part II could further push systematic generalization by including synthetic data into the pretraining process that demonstrates structural and algorithmic principles. Models could also be trained to simulate synthetically generated physical events to improve their physical and spatial representations, which are difficult to acquire from language alone. Similar approaches might also be beneficial during "post-training" to give a model a desired set of capabilities. For instance, for the domain of planning, one might generate planning problems in machine-readable formats such as PDDL (McDermott, 2000), and turn these into training data by translating the problem specification into natural language while providing gold standard outputs with an existing planning algorithm.

Part III

Appendix

Appendix A

Open-sourced Materials

Chapter 3 Code is available at https://github.com/namednil/f-then-r.

Chapter 4 Code is available at https://github.com/namednil/multiset-perm.

Chapter 5 Code is available at https://github.com/namednil/sip and the pre-trained SIP-d4 model is available at https://huggingface.co/namednil/sip-d4.

Chapter 6 Code is available at https://github.com/namednil/step and the pre-trained STEP model can be downloaded from https://huggingface.co/namednil/STEP.

Appendix B

Structured Reordering and Fertility

B.1 Data, Grammars, Pre- and Post-Processing

Synthetic data

In both setups, we generate 4000 training examples, 200 development examples and 1000 test examples. We use an alphabet size of $|\Sigma| = 11$ for the Length setup and $|\Sigma| = 11 + 3$ for the UC setup to accommodate for `x,y,z`. Symbols are chosen uniformly at random. In the Length setup, we choose the length of the example uniformly at random. In the UC setup, we do so as well but with probability 0.2 we insert an `xyz` cluster if this does not exceed the maximum length.

In the UC setup, we use development data from the training distribution.

We do not use grammar-based decoding or copying on the synthetic data.

Geoquery

We remove all parentheses in the logical forms, as they can be restored in post-processing. We also remove quotes around named entities in preprocessing to enable copying (`'texas'` becomes `texas`) and restore them in post-processing. Following Herzig and Berant (2021), we only allow copying of named entities and do not copy predicate symbols (e.g. `river`).

We use the grammar of Wong and Mooney (2006) for decoding as provided by Guo et al. (2020).

ATIS

We found that a naive length split led to having very few examples in the training set that used a date since both the month and the day count as one conjunct each. Therefore, we created 33 templates with three conjuncts based on existing ATIS examples (with four or more conjuncts) that contain dates and add 8 instances of each template with randomly chosen dates to the training set. In addition, we removed any exact duplicate samples from the data.

Similarly to Geoquery, we remove those parentheses that can be deterministically recovered in post-processing. However, in contrast to Geoquery the parentheses for `or`

Dataset	Split/Version	Train	Dev	Test
Geoquery	iid	540	60	280
	template	544	60	276
	length	540	60	280
ATIS	iid	4465	497	448
	length	4017	942	331
Okapi/length	Calendar	1145	200	1061
	Document	2328	412	514
	Email	2343	200	991
Style transfer	active to passive	2462	136	137
	adjective emphasis	627	34	35
	verb emphasis	1081	60	60

Table B.1: Number of examples per dataset/split.

and `intersection` need to be kept because the arities of those operators are not fixed. We run all our experiments on this representation of ATIS.

For grammar-based decoding we use the "typed" grammar provided by Guo et al. (2020) and do not use the copy mechanism.

Okapi

We found there was too much distributional overlap in the original length split provided by Hosseini et al. (2021) and therefore use our own split:

For the document domain, our development set contains examples with 4 parameters, and the test set contains examples with at least 5 parameters. For the other two domains, there is insufficient data for such out-of-distribution development sets. Therefore, we chose our test set to contain all examples with at least 4 parameters and our development sets to consist of 95% in-distribution data and 5% of examples from the the examples with 4 parameters (which makes the bulk of the test distribution).

We manually create a grammar of well-formed logical forms for the three domains of Okapi (included in the code).

StylePTB

For comparability, we also tokenize on whitespace following Kim (2021). We do not restrict the output of the model with a grammar.

B.2 Additional Results

Geoquery Table B.3 shows *exact match* accuracy of our models for comparison and Table B.4 shows results on the development set.

Transfer Type	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR	ROUGE-L	CIDEr
Active to passive	94.1±0.3	88.6±0.2	83.8±0.2	79.7±0.3	57.6±0.1	87.9±0.5	7.883±0.061
Adj. emphasis	78.4±1.2	64.3±1.0	54.5±0.7	46.9±0.9	44.0±0.4	69.5±0.6	4.809±0.113
Verb emphasis	67.6±0.5	51.2±0.7	40.6±0.9	32.7±1.3	36.8±0.3	59.1±0.7	3.587±0.118

Table B.2: Development accuracy of our F→R model with copying on StylePTB (syntactic transformation tasks). We report means and standard deviations of three random initialisations.

Model	iid	Template	Length
R→F †	83.5±0.7	72.7±2.0	54.0±4.0
R→F	76.4±1.4	65.8±2.2	39.4±6.2
F→R †	83.4±2.6	72.0±3.3	55.2±5.3
F→R	76.9±2.2	63.6±4.7	46.0±6.6
AR F→R	77.4±1.3	47.5±4.2	34.4±2.2

Table B.3: **Exact match** accuracy on the splits of **Geoquery**. We report mean and standard deviation of the 5 random initializations shown in the main text.

Model	iid	Template	Length
F→R	84.3±3.0	85.3±1.4	83.7±1.4
R→F	83.7±2.2	83.0±3.0	79.0±3.2
AR F→R	83.7±2.7	81.7±2.0	87.3±2.5

Table B.4: Mean and standard deviations of execution accuracy on the development sets of the **Geoquery** splits.

ATIS Table B.5 shows results on the development set. Table B.6 shows the average deviation from the gold length.

Okapi Table B.7 shows results on the development set.

StylePTB Table B.2 shows results on the development set.

B.3 Evaluation Metrics

We provide code for all evaluation metrics in our repository/dependencies.

Geoquery We use the code of Herzig and Berant (2021) to compute execution accuracy (`https://github.com/jonathanherzig/span-based-sp`).

ATIS We allow for different order of conjuncts between system output and gold parse in computing accuracy. We do this by sorting conjuncts before comparing two trees node by node.

Okapi We follow Hosseini et al. (2021) and disregard the order of the parameters for computing accuracy. Since Hosseini et al. (2021) did not make their evaluation code publicly available, we use our own implementation. Our implementation uses sets and does not punish a model for predicting a correct parameter multiple times. For example, if the gold logical form contains `FILTER message.isRead eq False`,

Model	iid	Length
LSTM seq2seq	81.53±0.77	43.86±2.93
Rel. Transformer	81.53±0.44	34.84±5.28
BART-base	90.54±0.45	65.29±1.01
F→R	73.80±1.62	54.42±1.25

Table B.5: Mean and standard deviations accuracy on the development sets of the **ATIS** splits (without grammar-based decoding).

Model	iid	Length
LSTM seq2seq	0.46±0.09	5.39±0.42
Rel. Transformer	0.49±0.07	6.19±0.52
BART-base	0.24±0.02	3.40±0.25
F→R	0.41±0.06	1.49±0.18

Table B.6: Means and standard deviations of the average absolute deviation from the gold length by model on **ATIS**, i.e. lower is better.

Model	Calendar	Document	Email
BART-base	94.8±0.3	56.2±10.9	91.4±0.4
F→R	86.4±3.4	66.1±4.8	85.0±2.8

Table B.7: Mean and standard deviations accuracy on the development sets of the **Okapi** splits (without grammar-based decoding).

a necessary condition for a prediction to count as correct is that it must contain this string *at least* once.

StylePTB We follow Kim (2021) and use `https://github.com/Maluuba/nlg-eval` (commit `7f79930`) for all evaluation metrics, ensuring that BLEU, ROUGE and METEOR are scaled to 0 - 100.

B.4 Hyperparameters

We provide a configuration file for each of our models with the chosen hyperparameters in our code repository (`configs/`). We set the maximum fertility value d to $d = 4$ for all datasets except for the style transfer tasks where we set it to $d = 3$.

At test time, we explore the top $k = 5$ most likely lengths when using grammar-based decoding. Without grammar-based decoding we used $k = 1$ as using $k = 5$ provided little improvement.

Many but not all instances of Geoquery require identity permutations. We found this to lead to the issue that the model gets stuck in a very steep local minimum within the first epoch where it would predict only identity permutations. We fixed this issue by reducing the learning rate in the feedforward network that predicts the scores for the permutation trees to 1×10^{-6}.

Hyperparameter selection

We select hyperparameters using a combination of manual selection and a random search. We optimize hyperparameters for accuracy on the development set of com-

positional generalization splits, where available, (Length setup for the synthetic data, template split for Geoquery), and then use those hyperparameters for all splits of a (domain of a) dataset.

The high variance in accuracy across random initializations often observed in compositional generalization setups makes it difficult to tune hyperparameters even if an out-of-distribution development set exists. We restrict random hyperparameter search to two random seeds. After the hyperparameter search, we pick the two most promising configurations (according to (execution) accuracy), pick a new random seed and train them again to choose the one which provides the most stable accuracy (approximated as the highest accuracy on the new seed). We then run the main experiments with the chosen hyperparameters and completely new random seeds.

We randomly sample 20 configurations per hyperparameter search. Since this procedure is expensive, we do not run train our models fully to convergence. The bounds of the hyperparameter search are reported in our repository.

For all our models, we initially chose hyperparameters manually, and then ran a random hyperparameter search as described above. If the manually chosen hyperparameters resulted in same or better performance (on the development set) on average, we kept those, and otherwise used the ones found by the hyperparameter search.

For example, on Geoquery, we noticed a particular sensitivity to hyperparameters, and the manually selected hyperparameters for F→R performed best with low variance, whereas for R→F, the hyperparameters found by the random search were better than the manually chosen ones. We think this sensitivity is at least in part caused by the small size of the dataset.

StylePTB In contrast to the other tasks we evaluate on, we did not run a hyperparameter search for the syntactic style transfer tasks and use the same manually determined hyperparameters for F→R across all style transfer tasks.

B.5 Run Times

All experiments were run on GeForce GTX 1080 Ti or GeForce GTX 2080 Ti with 12GB RAM and Intel Xeon Silver or Xeon E5 CPUs.

The run time of one run contains the time for training, evaluation on the devset after each epoch and running the model on the test set. We show the run time the models we train in Table B.8.

Dataset	Model	Epochs	Run Time	Instance/s
Mirror	F→R	7	8 min	63
	R→F	20	8 min	177
	AR F→R	20	30 min	47
	AR R→F	20	20 min	71
	Transformer	200	15 min	934
	LSTM	60	4 min	1054
Geo	F→R	100	20 min	50
ATIS	F→R	100	11-12h	12
	Transformer	20	10 min	166
	LSTM	20	10 min	166
	BART	50	1.3h	52
Okapi / Calendar	F→R	70	1 h	26
	BART	40	15 min	61
Okapi / Document	F→R	70	1.5 h	36
	BART	60	30 min	92
Okapi / Email	F→R	70	1.5 h	33
	BART	40	30 min	57
Active → Passive	F→R	60	1 h	43
Adj. emphasis	F→R	60	20 min	33
Verb emphasis	F→R	60	30 min	38

Table B.8: Average total run time of the models we train. Run times comprise time for training, validation after every epoch and inference on the test set. Likewise, instance per second is calculated across the entire run. For comparison on Geoquery, Herzig and Berant (2021) report a run time of 2 hours on comparable hardware to ours.

Appendix C

Multiset Tagging and Latent Permutations

C.1 Proofs and Derivations

C.1.1 NP-hardness

We show that Eq. (4.5) can be used to decide the Hamiltonian Path problem. Let $G = (V,E)$ be a graph with nodes $V = \{1,2,\ldots,n\}$. A Hamiltonian path $P = v_1, v_2, \ldots, v_n$ is a path in G (i.e. $(v_i, v_{i+1}) \in E$ for all i) such that each node of G appears exactly once. Deciding if a graph has a Hamiltonian path is NP-complete.

Reduction of Hamiltonian path to Eq. (4.5) Note that a necessary but not sufficient condition for P to be a Hamiltonian path is that P is a permutation of V. This will be ensured by the constraints on the solution in Eq. (4.5).

We construct a score function

$$\mathbf{s}_{k \curvearrowright i} = \begin{cases} 1 & \text{if } (k,i) \in E \\ 0 & \text{else} \end{cases} \tag{C.1}$$

and let $s_{i \mapsto j} = 0$ for all i, j. If we find the solution of Eq. (4.5) for the score function Eq. (C.1), we obtain a permutation P of V, which may or may not be a path in G. In a path of n nodes, there are $n-1$ edges that are crossed. If the score of the solution is $n-1$, then all node pairs (v_i, v_{i+1}) that are adjacent in P must have had a score of 1, indicating an edge $(v_i, v_{i+1}) \in E$. Therefore, P must be a Hamiltonian path. If the score of the solution is less than $n-1$, then there is no permutation of V that is also a path, and hence G has no Hamiltonian path.

□

C.1.2 Proof of Proposition 2

We now prove Prop. 2. The proof is very similar to a proof by Kushinsky et al. (2019) (their Theorem 2) but we go through it in more detail. As the constraints in Prop. 2 for any value of j do not interact with constraints for other values of j, we can assume

w.l.o.g. that j takes a single value only and drop it in the notation. This simplifies our notation and now we want to solve:

$$\begin{aligned} \underset{\mathbf{x},\mathbf{Y}}{\arg\min} \quad & \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{z}) + \mathrm{KL}(\mathbf{Y} \mid\mid \mathbf{W}) \\ \text{subject to} \quad & \sum_i \mathbf{x}_i = 1 \\ & \sum_k \mathbf{Y}_{ik} = \mathbf{x}_i \ \forall i \end{aligned} \tag{C.2}$$

We can find $\mathbf{Y}^* = \arg\min_{\mathbf{Y}} \mathrm{KL}(\mathbf{Y} \mid\mid \mathbf{W})$ subject to $\sum_j \mathbf{Y}_{i,j} = \mathbf{x}_i$ based on Prop. 1: $\mathbf{Y}^*_{i,j} = \frac{\mathbf{x}_i \mathbf{W}_{i,j}}{\sum_j \mathbf{W}_{i,j}}$. That is, we can express $\mathbf{Y}^*$ as a function of $\mathbf{x}$ (which we write $\mathbf{Y}^*(\mathbf{x})$), and therefore our overall problem is now a problem in one variable ($\mathbf{x}$):

$$\begin{aligned} \underset{\mathbf{x}}{\arg\min} \quad & \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{z}) + \mathrm{KL}(\mathbf{Y}^*(\mathbf{x}) \mid\mid \mathbf{W}) \\ \text{subject to} \quad & \sum_i \mathbf{x}_i = 1 \end{aligned} \tag{C.3}$$

The objective function can now be simplified. First, we simplify the generalized KL divergence $\mathrm{KL}(\mathbf{x} \mid \mathbf{y}) = \sum_i \mathbf{x}_i \log \frac{\mathbf{x}_i}{\mathbf{y}_i} - \mathbf{x}_i + \mathbf{y}_i$ to $\sum_i \mathbf{x}_i (\log \frac{\mathbf{x}_i}{\mathbf{y}_i} - 1)$ which does not change the identity of the minimizer wrt to $\mathbf{x}$. The objective function is simplified as follows:

$$\begin{aligned} & \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{z}) + \mathrm{KL}(\mathbf{Y}^*(\mathbf{x}) \mid\mid \mathbf{W}) \\ &= \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{z}) + \sum_{i,j} \frac{\mathbf{x}_i \mathbf{W}_{i,j}}{\sum_{j'} \mathbf{W}_{i,j'}} (\log \frac{\frac{\mathbf{x}_i \cancel{\mathbf{W}_{i,j}}}{\cancel{\sum_{j'} \mathbf{W}_{i,j'}}}}{\cancel{\mathbf{W}_{i,j}}} - 1) \\ &= \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{z}) + \sum_i \frac{\mathbf{x}_i \cancel{\sum_j \mathbf{W}_{i,j}}}{\cancel{\sum_{j'} \mathbf{W}_{i,j'}}} (\log \frac{\mathbf{x}_i}{\sum_{j'} \mathbf{W}_{i,j'}} - 1) \\ &= \sum_i \mathbf{x}_i (\log(\frac{\mathbf{x}_i}{\mathbf{z}_i}) - 1 + \log \frac{\mathbf{x}_i}{\sum_{j'} \mathbf{W}_{i,j'}} - 1) \\ &= \sum_i \mathbf{x}_i (2 \log \mathbf{x}_i - \log \mathbf{z}_i - \log \left(\sum_{j'} \mathbf{W}_{i,j'} \right) - 2) \\ &= 2 \sum_i \mathbf{x}_i (\log \mathbf{x}_i - 1 - \underbrace{\frac{1}{2} (\log \mathbf{z}_i + \log \left[\sum_{j'} \mathbf{W}_{i,j'} \right])}_{\log \mathbf{q}_i}) \\ &= 2 \cdot \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{q}) \end{aligned}$$

In effect, Eq. (C.3) is equivalent to the following simpler optimization problem:

$$\begin{aligned} \underset{\mathbf{x}}{\arg\min} \quad & \mathrm{KL}(\mathbf{x} \mid\mid \mathbf{q}) \\ \text{subject to} \quad & \sum_i \mathbf{x}_i = 1 \end{aligned} \tag{C.4}$$

where $\mathbf{q}_i = \sqrt{\mathbf{z}_i \cdot \sum_{j'} \mathbf{W}_{i,j'}}$.

The rewritten optimization problem Eq. (C.4) has the right form to apply Prop. 1 a second time. We obtain the solution:

$$\mathbf{x}_i^* = \frac{\mathbf{q}_i}{\sum_{i'} \mathbf{q}_{i'}}$$

By plugging this into $\mathbf{Y}^*(\mathbf{x})$, we obtain the solution to the overall optimization problem. □

C.1.3 Derivation of Loss Function as Evidence Lower Bound

We now show how the training procedure we use to train our permutation model can be derived from a form of evidence lower bound (ELBO).

Ideally, our permutation model would be a distribution $P_\theta(\mathbf{R}|x,z')$ over permutation matrices $\mathbf{R}$ and we would maximize the marginal likelihood, i.e. marginalizing over all permutations:

$$P_\theta(y|x,z') = \sum_{\mathbf{R}\in\mathcal{P}} P_\theta(\mathbf{R}|x,z')P(y|z',\mathbf{R}) \tag{C.5}$$

where $P(y|z',\mathbf{R}) = \prod_j \sum_i R_{ij} \cdot \mathbb{1}(y_j = z_i)$ with $\mathbb{1}$ being the indicator function. $P(y|z',\mathbf{R})$ returns 1 iff applying the permutation $\mathbf{R}$ to z' results in y. Unfortunately, computing Eq. (C.5) exactly is intractable in general due to the sum over permutation matrices. We instead use techniques from variational inference and consider the following evidence lower bound (ELBO):

$$\begin{aligned}\log P_\theta(y|x,z') \geq &\max_Q \mathbb{E}_{\mathbf{R}\sim Q(\mathbf{R}|x,z',y)} \log P(y|z',\mathbf{R}) \\ &- \mathrm{KL}(Q(\mathbf{R}|x,z',y) \;||\; P_\theta(\mathbf{R}|x,z'))\end{aligned} \tag{C.6}$$

where $Q(\mathbf{R}|x,z',y)$ is an approximate variational posterior. We now relax the restriction that $P_\theta(\mathbf{R}|x,z')$ places non-zero mass only on permutation matrices and use the following definition of $P_\theta(\mathbf{R}|x,z')$:

$$P_\theta(\mathbf{R}_{ij} = 1|x,z') = \mathbf{U}^*(\mathbf{s})_{ij}$$

where $\mathbf{U}^*(s)$ is the solution to Eq. (4.6) with added entropy regularization.

It turns out, in our case, we can easily construct a variational posterior Q that has zero reconstruction loss (the first term on the right side in Eq. (C.6)): we can choose any $Q(\mathbf{R}|x,z',y) \in Q(y,z')$ where $Q(y,z')$ is the set of bistochastic matrices such that $Q(\mathbf{R}|x,z',y)_{i,j} = 0$ iff $z'_i \neq y_j$. To see that this gives zero reconstruction error, consider position j in the output: The probability mass is distributed across precisely those positions i in z' where the right kind of token lives. In other words, any alignment with non-zero probability will reconstruct the output token at position j.

Therefore we can use the following lower bound to the log-likelihood:

$$\begin{aligned}&\log P_\theta(y|x,z') \geq \\ &- \min_{Q\in Q(y,\mathbf{z})} \mathrm{KL}(Q(\mathbf{R}|x,z',y) \;||\; P_\theta(\mathbf{R}|x,z'))\end{aligned} \tag{C.7}$$

During training, we need to compute the gradient of Eq. (C.7). By Danskin's theorem (Danskin, 1967), this is:

$$-\nabla_\theta \mathrm{KL}(\mathbf{Q}^* \mid\mid P_\theta(\mathbf{R}|x,z')) \tag{C.8}$$

where $\mathbf{Q}^* \in Q(y,z')$ is the minimizer of Eq. (C.7). Note that $\mathbf{Q}^*$ is the same as $\hat{\mathbf{U}}$ (Eq. (4.8)).

In practice, we also add $-\mathrm{KL}(\hat{\mathbf{W}}|\mathbf{W}^*(\mathbf{s}))$ to our objective in Eq. (C.7) to speed up convergence; this does not change the fact that we use a lower bound.

C.2 Further Model Details

Permutation Model

We do not share parameters between the multiset tagging model and the permutation model.

Due to the entropy regularization term, the output of Algorithm 1 is always dense, i.e. $\mathbf{U}^*$ and $\mathbf{W}^*$ contain no values that are exactly 0. For long sequences, small but non-zero values can accumulate and make the output a softer, more diluted distribution in comparison to shorter sequences with similar scores. To counteract this, we found it helpful to reduce the temperature τ for longer sequences and set $\tau = (\log n)^{-1}$ (see also Section 5.2 and 5.3 of Chiang and Cholak (2022)).

Since the permutation model is designed to model permutations and only permutations, the input z' to the permutation model and the desired output y must have the same elements during training. For simplicity, we discard training instances where z' predicted by the multiset model does not have the same elements as y. In practice, this leads to a very small loss in training data for the permutation model; we do not discard any test data. If one could not afford to discard any portion of the training data, one could constrain the prediction of the multiset model to output a correct sequence of multisets, i.e. find the maximizer of the posterior $P(\mathbf{z}|y,x)$.

Lexicon Mechanism

The lexicon L is a lookup table that deterministically maps an input token x_i to an output token $L(x_i)$, and we modify the distribution for multiset tagging as follows:

$$P'_\theta(\mathbf{z}_{i,v} = k|x) = \begin{cases} P_\theta(\mathbf{z}_{i,\mathscr{L}} = k|x_i) & \text{if } v = L(x_i) \\ P_\theta(\mathbf{z}_{i,v} = k|x) & \text{else} \end{cases}$$

where $P_\theta(\mathbf{z}_{i,v} = k|x)$ is as defined in Eq. (4.3) and $\mathscr{L}$ is a special lexicon symbol in the vocabulary. $P_\theta(\mathbf{z}_{i,\mathscr{L}}|x_i)$ is a distribution over the multiplicity of $L(x_i)$, independent of the identity of $L(x_i)$. We use the 'simple' lexicon induction method by Akyürek and Andreas (2021). Unless otherwise specified during learning, $L(x_i) = x_i$ like in a copy mechanism.

Handling of variables in COGS For the COGS dataset, a model has to predict variable symbols. The variables are numbered (0-based) by the input token that introduced it (e.g. in Fig. 4.1, slept, the third token, introduces a variable symbol $\texttt{x}_2$). In order to robustly predict variable symbols for sentences with unseen length, we use a similar mechanism as the lexicon look up table: we introduce another special symbol in the vocabulary, `Var`. If `Var` is predicted with a multiplicity of k at i-th input token, it adds the token $\texttt{x}_{i-1}$ to its multiset k times.

Initialization of Multiset Tagging Model

If there are l alignments with a posterior probability of at least χ that an input token i produces token v, we add the term $\lambda \log P_\theta(\mathbf{z}_{i,v} \geq l \mid x)$ to Eq. (4.4). λ is the hyperparameter determining the strength. This additional loss is only used during the first g epochs.

C.3 Datasets and Preprocessing

We show basic statistics about the data we use in Tables B.1 and C.1. Except for the doubling task, all our datasets are in English.

Doubling task For the doubling task, we use an alphabet of size $|\Sigma| = 11$. To generate inputs with a specific range of lengths, we first draw a length from the range uniformly at random. The symbols in the input are also drawn uniformly at random and then concatenated into a sequence. Examples of lengths 5 - 10 are used as training, examples of length 11 are used as development data (e.g. for hyperparameter selection), and examples of length 11 - 20 are used as test data.

Preprocessing

COGS Unlike Zheng and Lapata (2022); Qiu et al. (2022a); Drozdov et al. (2022) we do not apply structural preprocessing to the original COGS meaning representation and keep the variable symbols: all our preprocessing is local and aimed at reducing the length of the logical form (to keep run times low). We delete any token in {`","`,`"_"`,`"("`,`")"`,`"x"`,`"."`,`";"`,`"AND"`} as these do not contribute to the semantics and can be reconstructed easily in post-processing. The tokens {`"agent"`, `"theme"`, `"recipient"`, `"ccomp"`, `"xcomp"`, `"nmod"`, `"in"`, `"on"`, `"beside"`} are always preceded by a `"."` and we merge `"."` and any of those tokens into a single token. Example:

```
* cookie ( x _ 3 ) ; * table ( x _ 6 ) ; lend .  agent ( x _ 1 ,
Dylan ) AND lend .  theme ( x _ 1 , x _ 3 ) AND lend .  recipient ( x
_ 1 , x _ 9 ) AND cookie .  nmod .  beside ( x _ 3 , x _ 6 ) AND girl
( x _ 9 )
```

Becomes

```
* cookie 3 * table 6 lend .agent 1 Dylan lend .theme 1 3 lend
.recipient 1 9 cookie .nmod .beside 3 6 girl 9
```

Dataset	Train	Dev	Test
Doubling	4,000	500	1,000
COGS	24,155	3,000	21,000

Table C.1: Number of examples per dataset. See also Table B.1.

C.4 Evaluation Metrics

We provide code for all evaluation metrics in our repository.

Doubling We use exact match accuracy on the string.

COGS For COGS we use exact match accuracy on the sequence in one evaluation setup. The other evaluation setup disregards the order of conjuncts: we first remove the 'preamble' (which contains all the definite descriptions) from the conjunctions. We count a prediction as correct if the set of definite descriptions in the preamble matches the set of definite descriptions in the gold logical form *and* the set of clauses in the prediction match the set of clauses in the gold logical form.

ATIS We allow for different order of conjuncts between system output and gold parse in computing accuracy. We do this by sorting conjuncts before comparing two trees node by node. This is the same evaluation metric as used in Chapter 3.

Okapi As in Chapter 3, we disregard the order of the parameters for computing accuracy. We use a case-insensitive string comparison.

C.5 Hyperparameters

We use the same hyperparameters for all splits of a dataset. For our model, we only tune the hyperparameters of the multiset tagging model; the permutation model is fixed, and we use the same configuration for all tasks where we use RoBERTa. For model ablations where we use an LSTM instead of RoBERTa, we use the same hyperparameters for Okapi and ATIS, and a smaller model for the doubling task. These configurations were determined by hand without tuning. For BART, we use the same hyperparameter as in Chapter 3.

We use the random hyperparameter search procedure from Chapter 3 for the multiset tagging models and the LSTM/transformer we train from scratch: we sample 20 configurations and evaluate them on the development set. We run the two best-performing configurations again with a different random seed and pick the one with the highest accuracy (comparing the union of the predicted multisets with the gold multiset). We then train and evaluate our model with entirely different random seeds.

The chosen hyperparameters along with the search space are provided in the github repository.

Dataset	Model	First stage	Second stage
Doubling	LSTM	5	-
	Transformer	3	-
	Ours	3	14*
COGS	Ours	20	80
ATIS	Ours	30	50
	Ours/LSTM	30	110
Okapi/Calendar	Ours	9	35
	Ours/LSTM	5	30
Okapi/Email	Ours	12	40
	Ours/LSTM	9	40
Okapi/Document	Ours	12	70
	Ours/LSTM	10	55

Table C.2: Average run time for train/evaluate on dev and test in **minutes**. * For the doubling task, we note that our model has converged usually after $\frac{1}{4}$ of the time in the table, but we train a little longer.

C.6 Run Times

Experiments were run on the same hardware as the experiments in Chapter 3. The run time of one run contains the time for training, evaluation on the devset after each epoch and running the model on the test set. We show run times of the model we train in Table C.2. Since we evaluate on 5 random seeds (10 for COGS due to high variance of results), our experiments overall took around 64 hours of compute time on our computing infrastructure.

Appendix D

Injecting an FST-like Inductive Biases into a Transformer

D.1 Formalization of SIP as Continuous Relaxation

To make the argument from Section 5.7 precise, let $x = x_1, \ldots, x_n$ be an input sequence. Let $f_\theta(x) = y$ be a discrete function that maps x to the output y. f_θ is parameterized by some object θ that more closely describes what the function does. In the case of permutations, θ encodes a specific permutation and $f_\theta(x)$ reorders x according to the given permutation. For the fertility layer, θ is a sequence of fertility values, each indicating the fertility of one token and $f_\theta(x)$ goes over each token and makes the corresponding number of copies.

For defining a continuous relaxation of f_θ, we assume we have another function $\text{ENCODE}(\theta, x) = \phi$ which produces parameters ϕ in continuous space. In addition, we assume a distribution $P_{\text{RELAX}(\phi,\tau)}(y)$ over output sequences y that is a *differentiable function* of the parameters ϕ and the temperature τ. We say that $P_{\text{RELAX}(\text{ENCODE}(\theta,x),\tau)}$ is a continuous relaxation of f_θ if it places more and more probability mass on $f_\theta(x)$ as we anneal the temperature to 0:

$$\lim_{\tau \to 0^+} P_{\text{RELAX}(\text{ENCODE}(\theta,x),\tau)}(f_\theta(x)) = 1$$

This definition can be instantiated as follows for the methods we have presented:

Fertility $\text{ENCODE}(\theta, x) = \phi$ produces scores for fertility values as follows: $\phi_{i,k} = \exp[\![x_i \text{ has fertility } k \text{ according to } \theta]\!]$. In the notation of Chapter 3, $P_{\text{RELAX}(\phi,\tau)}$ first computes a distribution over fertility values $P(\mathbf{f}_i = k) \propto \tau^{-1} \cdot \phi_{i,k}$ and then computes the marginal alignment distribution indicating which position in the input each output came from (Eq. (3.9)). The alignment distribution can then straightforwardly be turned into a distribution over sequences $\mathbf{y}$. As we decrease the temperature τ towards 0, $P(\mathbf{f}_i)$ becomes more and more certain about the fertility of token x_i. As a result, the marginal alignment distribution also becomes certain about the correspondence between input and output. In turn, this causes the distribution over outputs $\mathbf{y}$ to place more and more mass on a single output sequence.

Permutation Assume θ encodes a permutation that maps input i to output j as $\theta_i = j$. Then $\text{ENCODE}(\theta, x)$ produces scores $\mathbf{s}_{k \curvearrowright i} = [\![\exists j.\theta_j = k \wedge \theta_{j+1} = i]\!]$, i.e. it gives high values for the jumps that are needed for the permutation represented by θ. $\text{ENCODE}(\theta, x)$ also gives high scores to $s_{i \mapsto 1} = [\![\theta_i = 1]\!]$ and $s_{j \mapsto n} = [\![\theta_j = n]\!]$ for correctly starting and ending the permutation. $P_{\text{RELAX}(\phi,\tau)}$ uses Algorithm 1 to define a distribution over alignments. The alignments can be straightforwardly used to define a distribution over outputs. Recall that τ in Algorithm 1 determines the strength of the entropic regularizer added to the linear program Eq. (4.6). Consequently, as τ goes to 0, the regularizer vanishes and the optimal solution is the permutation encoded in θ.

FSTs with SIP For SIP, θ represents a functional FST and $f_\theta(x)$ simply applies the FST to the input x. $\text{ENCODE}(\theta, x)$ encodes the FST and the input string into a sequence of vectors as given by Eq. (5.1). Finally, $P_{\text{RELAX}(\phi,\tau)}(y)$ refers to the distribution produced by the *trained* SIP model when given the prefix ϕ. Here, τ is the temperature of the softmax over the vocabulary. Since SIP does not achieve 100% accuracy in predicting the output an FST produces, the limit is not satisfied for all FSTs θ and x but for around 98% within the pretraining distribution.

D.2 Generation of Synthetic Data and Splits

D.2.1 Generating deterministic FSTs

Before describing our procedure for sampling deterministic FSTs, we briefly establish notation. An FST is a tuple $\langle Q, \Sigma, \Gamma, I, F, \Delta \rangle$, where Q is a finite set of states, Σ is the input alphabet, Γ is the output alphabet, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$ are the transitions. We assume $\Sigma = \Gamma$ and call it V for vocabulary.

For technical reasons, we exclude the three characters `[`, `]` and `\` from the vocabulary as they are interpreted as special characters by OpenFST, which we use for constructing and representing FSTs.

In addition to the shorthand for identity transitions (`id`), we also have shorthands for converting upper case to lower case and vice-versa, i.e. `lower-to-upper` and `upper-to-lower`. We describe our procedure to generate a deterministic FST with pseudo code in Algorithm 2. It receives as argument n (the number of states in the FST), f (number of final states), V (the vocabulary of this FST), and probabilities P-ID, P-DROP, P-SHORTHAND. These probabilities control the likelihood of using a shorthand, not drawing an outgoing edge (P-DROP) with a given symbol, and creating a single identity transition (P-ID). We use CHOICE to denote a uniform random choice from a finite set.

We use $\text{P-ID} = 0.2, \text{P-DROP} = 0.4, \text{P-SHORTHAND} = 0.15$ in our experiments.

D.2.2 Generating Non-deterministic Functional FSTs

It is not straightforward to directly generate non-deterministic FSTs that are guaranteed to express a function. However, we can directly generate a bimachine, which then can be converted into an FST.

Algorithm 2 Generate a random deterministic FST

function GEN-DET-FST(n, f, V, P-ID, P-DROP, P-SHORTHAND)
  $Q = \{0, \dots n-1\}$
  $\Delta = \emptyset$
  $I = \{0\}$
  for $q \in Q$ **do**
    $q' = \text{CHOICE}(Q)$
    with prob P-SHORTHAND
      $s = \text{CHOICE}([\texttt{id}, \texttt{lower-to-upper}, \texttt{upper-to-lower}])$
      $\Delta := \Delta \cup \{q \xrightarrow{s:s} q'\}$
    else
      for $\sigma \in V$ **do**
        with prob P-DROP
          no-op ▷ No outgoing edge with σ at q
        else with prob P-ID
          $\Delta := \Delta \cup \{q \xrightarrow{\sigma:\sigma} q'\}$
        else
          $\Delta := \Delta \cup \{q \xrightarrow{\sigma:\text{CHOICE}(V \cup \{\varepsilon\})} q'\}$
  Eliminate states from Q through which no accepting path can go
  Choose random subset F of Q with $|F| = \min(f, |Q|)$
  return minimized FST with states Q, transitions Δ, initial states I and final states F

Bimachines (Schützenberger, 1961) represent the functions expressible by FSTs, i.e. for every functional FST there is a bimachine that represents it (and vice-versa). A bimachine consists of two deterministic finite state automata (called left and right) and an output function. Let A^L be the left FSA with states Q^L and transition function $\delta^L : Q^L \times \Sigma \to Q^L$), and let A^R bet the right FS with states Q^R and transition function $\delta^R : Q^R \times \Sigma \to Q^R$. The output function is $\psi : Q^l \times \Sigma \times Q^r \to \Gamma^*$. All states of A^L and A^R are final states. Given an input string $x = \sigma_1\sigma_2\sigma_3 \dots \sigma_n$, a bimachine runs A^L from left to right over x, keeping track of the states $q^l_0, q^l_1, q^l_2, \dots q^l_n$. It also runs A^R over the string x but this time from right to left, again keeping track of the states $q^r_0, q^r_1, q^r_2, \dots q^r_n$ that are visited. Then, the state sequence of the right automaton is reversed and ψ is applied 'elementwise' as illustrated in Fig. D.2. More formally, the output of the bimachine is $\psi(q^l_0, \sigma_1, q^r_{n-1})\psi(q^l_1, \sigma_1, q^r_{n-2}) \dots \psi(q^l_{n-1}, \sigma_1, q^r_0)$.

Bimachines can be compiled into FSTs with a simple product construction. For a bimachine $\langle A^L, A^R, \psi \rangle$, one can construct an equivalent FST as follows:

$$\langle Q^L \times Q^R, \Sigma, \Gamma, \{s^L\} \times Q^R, Q^L \times \{s^R\}, \Delta \rangle$$

where s^L and s^R are initial states of A^L and A^R, and Δ contains all transitions

$$\Delta = \{\langle q^L, q^R \rangle \xrightarrow{\sigma:\rho} \langle q'^L, q'^R \rangle \mid \delta^L(q^L, \sigma) = q'^L, \\ \delta^R(q'^R, \sigma) = q^R, \\ \rho = \psi(q^L, \sigma, q'^R)\}$$

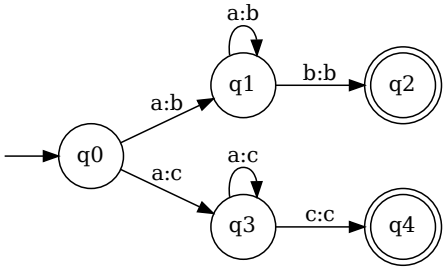


Figure D.1: A functional but non-deterministic FST.

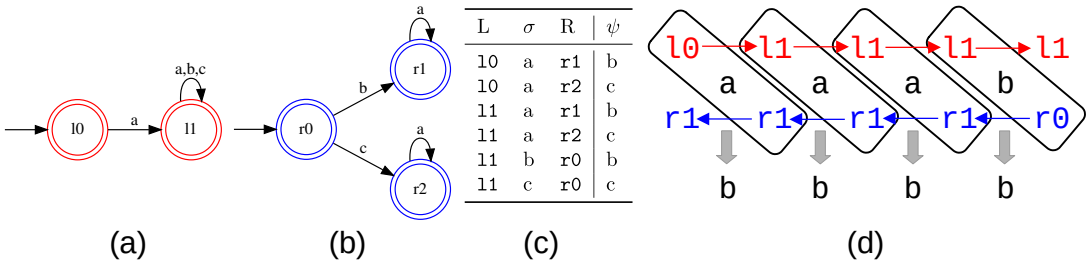


Figure D.2: (a) - (c) shows a bimachine that is equivalent to Fig. D.1. (a) Left automaton A^l, (b) Right automaton A^r, (c) output function ψ. (d) shows an example run of the bimachine on the input `aaab` which is mapped to `bbbb`.

We refer to Mihov and Schulz (2019) for details and further information about bimachines.

To sample bimachines, we re-use Algorithm 2 with P-SHORTHAND $= 0$, and ignore the outputs of the transitions, treating them as FSAs. We sample the output function according to Algorithm 3. For the test data creation (Table 5.3), we use 5 states in the left FSA and 4 states in the right FSA, and set P-DROP $= 0.4$. This results in FSTs with $4 \times 5 = 20$ states - any subset of which can be initial states. If a unique initial state is desired (as required by OpenFST), this leads to a total of 21 states. For creating the training data for SIP-nd7, we use 2 or 3 states in either left or right automaton and set P-DROP $= 0.6$ to keep the length of the prefix low to save GPU memory.

D.2.3 Unseen Combinations of Transitions

We briefly describe how we select *which* pairs of transitions we want to withhold. We only select adjacent transitions, i.e. transitions where one can be used immediately after the other, excluding self-loops. In addition, some transitions cannot be deleted without cutting off a vital initial or final state, which can lead to f_{train} being undefined for any string (and hence no training data). We ensure this never happens by never withholding the first transition into each state based on a depth-first traversal of the FST.

Algorithm 3 Generate output function for bimachine

function GEN-OUTPUT-$\psi(n^L, n^R, V, \text{P-ID} = 0.2)$
 for $q^L \in 0, \ldots, n^L - 1$ **do**
  for $q^R \in 0, \ldots, n^R - 1$ **do**
   for $\sigma \in V$ **do**
    with prob P-ID
     $\psi(q^L, \sigma, q^R) := \sigma$
    else
     $\psi(q^L, \sigma, q^R) := \text{CHOICE}(V \cup \{\varepsilon\})$
 return ψ

Num. states	Split	Min	Max	Mean
4	train	2	11	4.66
4	test	4	30	18.97
5	train	2	14	5.39
5	test	4	30	19.53
7	train	2	20	6.12
7	test	4	30	20.13
10	train	2	25	7.31
10	test	4	30	20.62
21	train	2	30	11.80
21	test	5	30	23.07

Table D.1: Distribution of input lengths of the train/test data we generate for the iteration generalization experiments in Section 5.3. The tasks with 21 states are the non-deterministic FSTs from Section 5.3.6.

D.2.4 Additional Dataset Information

For all experiments with synthetic data (generated by FSTs), we generate 5000 training examples and 1000 test examples. To reduce variance across tasks, we fix the vocabulary size to its maximum value (25) in the pretraining data and choose the vocabulary only from the printable ASCII characters.

Length distribution The input strings in the pretraining data we generate for SIP-d4 have a minimum length of 1, an average length of 15.57 and a maximum length of 35. We report the length distributions for the iteration generalization experiments in Section 5.3 in Table D.1.

SyGuS We took the data from the SyGuS competition github `https://github.com/SyGuS-Org/benchmarks/tree/master/comp/2017/PBE_Strings_Track`, and extracted the 'constraints'. For each text editing tasks, there are usually three files, e.g. `firstname`, `firstname-long`, `firstname-long-repeat`. We only consider data from the `*-long` variant because the non-marked variant (e.g. `firstname`) is a subset

Gen. Type	Num States / Model	4 Acc↑	4 ED↓	5 Acc↑	5 ED↓	7 Acc↑	7 ED↓	10 Acc↑	10 ED↓
Iteration	ByT5	37.8	5.87	58.7	3.21	48.2	3.71	45.7	3.87
	Naive	42.6	4.41	60.5	2.20	47.7	3.16	43.6	3.65
	Set	44.4	4.58	62.2	2.41	48.0	3.49	45.3	3.71
	TE	61.3	2.49	78.9	0.86	55.7	2.29	50.7	2.95
	SIP-d4	**94.8**	**0.12**	**89.6**	**0.27**	**64.3**	**1.34**	**56.9**	**2.39**
UC	ByT5	47.4	1.49	62.6	1.05	61.9	1.29	54.1	1.70
	Naive	44.9	1.52	61.6	1.08	59.3	1.30	51.8	1.68
	Set	43.6	1.47	60.6	1.09	60.8	1.31	51.1	1.71
	TE	57.3	1.13	65.9	0.98	65.7	1.17	55.3	1.60
	SIP-d4	**73.1**	**0.61**	**74.3**	**0.69**	**73.2**	**0.85**	**58.0**	**1.44**

Table D.2: Evaluation on deterministic FSTs with more states, showing absolute accuracies and edit distances, corresponding to Fig. 5.4

of the `*-long` variant, and we exclude `*-long-repeat` as it contains repeated data points. We also exclude some text editing tasks that have insufficient amounts of data for reliable evaluation (`bikes`) and some tasks where the input is not a single string but a pair of strings if concatenating the strings results in particularly long inputs (`univ`), or if the concatenation of the string pair makes the task trivial (name-combine, which would correspond to an identity operation). For a few-shot experiment, we sample 5 training examples and evaluate on the rest.

We note that the original intention in the design of the benchmark data was for program synthesis rather than few-shot learning. The data contains names, and separately it contains phone numbers (but not combined). However, we believe both to be synthetically generated.

Grapheme-to-phoneme We obtain data from Lee et al. (2020), and conduct experiments mainly on the broad transcription, except for Telugu and Tamazight, where we use the narrow transcription. For each experiment, we randomly sample 100 training examples, and use the rest as test data.

D.3 Additional Results

D.3.1 Additional Results with More States

In Fig. 5.4, we show accuracy relative to the accuracy of ByT5. Here, we show the absolute accuracies and edit distances in Table D.2.

D.3.2 Full Results for Grapheme-to-Phoneme Conversion

Table D.3 shows the full results of our grapheme-to-phoneme conversion experiments, including phoneme error rate (PER).

	ban		cop		got		lao		syl		tel		tzm		Avg	
	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc↑	PER↓
Charsiu	68.3	.110	7.8	.579	67.0	.067	35.1	.238	47.6	.196	73.3	.070	18.6	.403	45.4	.238
ByT5	50.2	.233	1.0	.847	30.7	.269	1.9	.760	9.8	.598	6.9	.597	2.7	.851	14.8	.594
Set	53.9	.216	2.2	.742	58.2	.094	5.8	.595	28.2	.353	27.7	.293	6.4	.658	26.1	.421
TE	54.7	.183	1.9	.756	37.0	.174	5.1	.573	30.0	.309	16.2	.377	7.4	.644	21.8	.431
SIP-d4	**59.2**	**.152**	**6.6**	**.563**	56.5	.096	**8.2**	**.498**	**39.8**	**.252**	**33.1**	**.228**	**11.0**	**.544**	**30.6**	**.333**
-prefix	55.1	.168	3.2	.681	**63.9**	**.072**	7.8	.508	28.0	.333	28.9	.252	7.0	.593	27.7	.372

Table D.3: Grapheme-to-phoneme conversion with 100 training examples. We show averages of 5 selections of training examples. PER is Phoneme Error Rate: edit distance / length of gold output (lower is better).

	Iteration		UC	
	Acc↑	ED↓	Acc↑	ED↓
T5-Set	26.6	6.26	55.1/54.6	1.18/1.02
T5-SIP-d4	94.5	0.11	75.4/99.5	0.54/0.01

Table D.4: Evaluating systematic generalization on FST tasks with 4 states (cf. Table 5.1). Due to an outlier task on UC, we additionally report the median after '/'.

	Iteration		UC	
	Acc↑	ED↓	Acc↑	ED↓
T5-Set	77.9	0.73	81.7	0.53
T5-SIP-d4	83.3	0.56	86.1	0.37

Table D.5: Evaluation with T5-Base on non-deterministic FSTs (cf. Table 5.3)

D.3.3 Additional results with T5-Base

We run a subset of the experiments starting off from a pretrained T5-Base (Raffel et al., 2020) instead of ByT5. This model is about one-third smaller than ByT5 (around 200 million instead of 300 million parameters). T5-Base uses a different vocabulary than ByT5, so we resize the output layer to the vocabulary size of ByT5 and re-initialize it. For the input embeddings, we re-purpose the first n embeddings in the T5-Base embedding matrix to represent the token ids according to the ByT5 tokenizer. While this is suitable as a starting point for further pretraining, we found that directly fine-tuning T5-Base with these modifications on downstream tasks led to very poor results and do not include them here. Instead, we train T5-Set (analogous to Set) for a fair point of comparison.

We report a subset of the results from the main paper in for T5-Base in Tables D.4 to D.6.

We also tried to pretrain a ByT5-style model from scratch (i.e. from random initialization). However, we could not find a setting of hyperparameters that would make the model converge well. We hypothesize that the model already needs to be in a reasonable space to make learning feasible.

	ban		cop		got		lao		syl		tel		tzm		Avg	
	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc	PER	Acc↑	PER↓
T5-Set	47.9	.231	1.2	.783	6.7	.458	3.6	.643	6.6	.611	4.9	.612	2.7	.797	10.5	.591
T5-SIP-d4	59.1	.154	4.7	.640	69.6	.059	5.9	.566	22.1	.447	35.4	.191	12.5	.509	29.9	.367

Table D.6: Grapheme-to-phoneme conversion with 100 training examples based on T5-Base. In contrast to the experiments in the main text, we found that T5-SIP-d4 did not perform well on completely unseen scripts, so we mapped all Unicode code points to arbitrary ASCII characters. This maintains the structure of the task and is completely reversible. T5-Set is evaluated in the same way.

D.4 Additional Analyses

D.4.1 Analysis of Fine-Tuned Prefixes

To gain some understanding of how the prefix of tunable embeddings is used by the model and what it contains, we consider the setup of fine-tuning only the prefix and keeping the rest of the model unchanged. That is, all the task-specific information has to be captured in these embeddings. Specifically, we fine-tune on the 5 FSTs from Section 5.3.3 for iteration generalization for 20 epochs with a learning rate of 0.5.

We explore two questions:

1. Is the model robust towards different permutations of the fine-tuned prefixes? Intuitively, these permutations correspond to changing the order in which transitions are listed, so ideally the model should not be sensitive to that order.

2. Does the fine-tuned prefix represent the task-specific information in a similar way to how FSTs were encoded during pretraining?

To address the first question, we randomly permute the tuned prefixes and compute accuracy on the iteration generalization data before and after permuting the tuned prefixes. We use 20 permutations per learned prefix and average results across the 5 FSTs. Overall, we find that this results only in a small drop in accuracy: the median drop in accuracy is only around 1.3 percentage points, and the arithmetic mean of the drop is around 7.1 percentage points. Most permutations do not have a big impact on how the prefix is interpreted but a few permutations do have a stronger negative impact, skewing the arithmetic mean.

To address the second question, we test if the learned prefix for a task t resembles an encoding of an FST that solves t. For each of the 5 FSTs, we generate 10,000 distractors, i.e. FSTs that have the same number of states and use the same vocabulary as the FST solving t. We define the similarity of two prefixes p, q as follows:

$$sim(p,q) = \max_{\pi} \frac{1}{n} \sum_{i} \frac{p_i^T q_{\pi(i)}}{||p_i||_2 \cdot ||q_{\pi(i)}||_2}$$

where π is a permutation, and p_i is the i-th vector in prefix p, and prefixes p and q both have length n. That is, we define the similarity between p and q as the highest

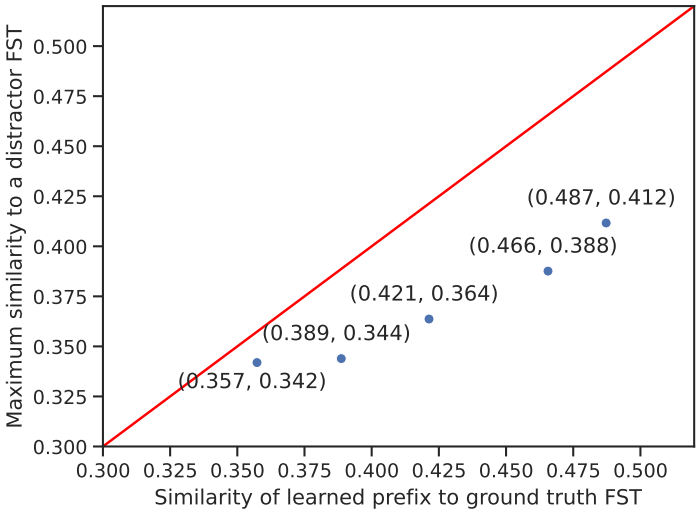


Figure D.3: Each dot represents a fine-tuned prefix when the rest of the model remains frozen during fine-tuning. The x-coordinates represent the similarity to a ground truth gold prefix, and the y-coordinates represent the maximum similarity to any of the 5×10000 distractor FSTs. All dots are below the diagonal, hence all learned prefixes are most similar to an encoding of the ground truth FST.

possible average cosine similarities between positions in p and q that one can achieve by assigning a position in p to exactly one position in q and vice-versa.[1] Taking the maximum over all permutations is justified by our results to the first question above, which showed that the model is largely invariant to different permutations of the tuned prefix.

For every task t, we compute the similarity between the prefix p learned by fine-tuning on input/output pairs and the union of encodings of the distractors and encodings of the gold standard FST for task t. Where necessary, we truncate encodings of FSTs to have the same length as the learned prefix. We present the results in Fig. D.3, showing that all learned prefixes are more similar to an encoding of the ground truth FST than to a distractor FST.

D.4.2 Probing non-SIP Models

All probes are trained for one epoch on activations produced by passing 8,000 FSTs with 5 inputs each (i.e. 40,000 instances) through the model.

For the baseline probe, we take the trained SIP-d4 model (including matrix W and embeddings from 6.1.2) and re-initialize the Transformer to ByT5-small. The probe achieves only a token-level accuracy of 42.9% and whole-sequence accuracy of 8.1%. We see very similar results for a probe trained on a randomly initialized Transformer

[1] Computing the similarity $sim(p,q)$ is relatively expensive because it involves solving the assignment problem (e.g. with the Hungarian algorithm). Instead of solving the assignment problem exactly, we approximate it with the Sinkhorn algorithm (Sinkhorn, 1964). We then take the output of the algorithm (a matrix of 'soft' assignments) and for each position in p, we greedily select a matching position in q.

in this setup: a token-level accuracy of 42.5% and a whole-sequence accuracy of 7.1%.

D.5 Hallucination Example

We briefly show an example where an LLM ignores a part of the input and resorts to outputting a high-frequency entity. Consider the following in-context examples for a simple text editing task:

Input	**Output**
Howard Phillips Lovecraft	H.P. Lovecraft
John Ronald Reuel Tolkien	J.R.R. Tolkien
Thomas Stearns Eliot	T.S. Eliot

At the time of the paper submission, ChatGPT frequently output "J.K. Rowling" for the name "John Edward Rowling", hallucinating the K.

D.6 Additional Model Details and Hyperparameters

SIP For completeness, we now describe the order in which we arrange the transitions. While the ordering of the transitions does not matter for expressing FSTs, the Transformer uses positional encodings which might have impacts on the pretraining (though see Appendix D.4.1). We assemble the overall prefix by stacking the individual vectors $h_0, \ldots, h_n$ of the transitions $p_0 \xrightarrow{\sigma_0:\rho_0} q_0, \ldots, p_n \xrightarrow{\sigma_n:\rho_n} q_n$. We group the transitions by their originating state (i.e. p_i) and go over the states by their id, starting with 0, the initial state.

During pretraining, we might encounter FSTs with different numbers of transitions within the same batch. To handle this, we use padding encodings by reserving a special padding state and padding symbol in the embedding matrices of states and symbols. To initialize the prefix for fine-tuning, we use the average of 32 FST encodings (chosen at random) from pretraining.

For pretraining, we use embeddings of dimensionality 64 for states, embeddings of dimensionality 256 for symbols, and of dimensionality 16 to indicate final/non-final states.

Task embeddings To enable faster adaption of the task embeddings than the rest of the model to fit a particular task, we use a higher learning rate for the task embeddings (1.0) than for the rest of the model ($5 \cdot 10^{-4}$) during pretraining. We also use a higher learning rate for the prefix during fine-tuning, analogously to SIP.

Because we have to store 40,000 task embeddings (one for each generated FST), TE requires a lot of memory. To reduce memory consumption, the task embeddings have a dimensionality of 180 and are up-projected to fit into the Transformer, analogously to W in Section 6.1.2. Nevertheless, the memory consumption of the embeddings is substantial and we store them on a separate GPU. Analogously to SIP-d4, we pretrain for 20 epochs.

Naive We pretrain for a single epoch only as we found this achieved better results on downstream tasks than training for 20 epochs.

Set We sample 200,000 examples according to the procedure described by Wu et al. (2022) to match our pretraining dataset size. Again, we found it more helpful for downstream task performance to train for a single epoch rather than 20 epochs.

Fine-tuning Hyperparameters Like pretraining, we finetune with the Adam optimizer. The main hyperparameters involved for both SIP and TE are the learning rates for the main model, and (separately) the learning rate of the tunable prefix. We chose these manually. Generally, we found that using a learning rate of 1.0 was a good choice for the prefix. Lester et al. (2021) report a similarly high learning rate to be useful for prompt tuning. For the rest of the model, we found $3 \cdot 10^{-4}$ and $5 \cdot 10^{-4}$ to work well for SIP-d4 and TE, respectively. For few-shot experiments, we use a somewhat smaller learning rate for TE for the main model ($3 \cdot 10^{-4}$). We noticed that T5-SIP-d4 (see Appendix D.3.3) was more sensitive to the learning rate choice in general than SIP-d4.

D.7 Run Times

We ran our experiments within the same computing infrastructure as in Chapters 3 and 4. Specifically, we used NVIDIA GeForce RTX 2080 Ti GPUs (12 GB RAM) with driver version 535.54.03 and cuda version 12.2.

Pretraining SIP-d4 took around 30 hours for 20 epochs. One training run on synthetic data (including evaluation) takes around one hour, and one training run for low-resource grapheme-to-phoneme conversion takes between 5 and 10 minutes.

Appendix E

Pretraining to Perform Syntactic Transformations

E.1 Pre-Processing

SLOG For SLOG, we remove `nmod.` from the logical forms to shorten them and to avoid giving models pretrained with syntax trees a potential advantage simply because the downstream logical form uses a similar token to a dependency label. Hence, the logical form for 'Isabella forwarded a box on a tree to Emma.' becomes `forward ( agent = Isabella , theme = box ( on = tree ) , recipient = Emma )` after pre-processing with the original one being `forward ( agent = Isabella , theme = box ( nmod . on = tree ) , recipient = Emma )`.

ATIS We train an IBM-1 alignment model on the pairs of sentences and logical forms, and then sort the conjuncts by their sum-total expected alignment: let $A_{i,j}$ be the posterior probability that the input token at position i is aligned to output the output at position j. Let C be the set of output token positions belonging to a conjunct. We then let $A(C) = \sum_{j \in C} \sum_i A_{i,j} \cdot i$. We repeat this for every conjunct and then sort them.

E.2 Experimental Setup

Syntactic Transformations and Chunking We evaluate in a few-shot scenario with only 100 training examples, and do not assume access to a development set. For this reason, we don't tune hyperparameters and fine-tune for a fixed number of epochs. As performance can differ from checkpoint to checkpoint, for each run, during the last 10 epochs, we evaluate on the test set and use the average result as the performance of that run.

For adjective emphasis, verb emphasis and passivization, we use all examples besides the 100 training examples as test set, i.e. 2635 test examples for passivization, 596 for adjective emphasis, and 1101 for verb emphasis. For chunking, we use the test set from Tjong Kim Sang and Buchholz (2000).

ATIS As in Chapters 3 and 4, we use the development set to select the best epoch based on accuracy metric.

SLOG SLOG does not have an out-of-distribution development set, so we train for a fixed number of epochs that was determined by the hyperparameter search (see Appendix E.4).

Identifying look-up heads We use a sample of 1000 unseen sentences from the C4 corpus along with randomly generated transformations as described in Section 6.1.1 to identify interpretable look-up heads.

Intervening on look-up heads Since we want to evaluate the impact of look-up heads for particular dependency relations, we create a dataset with 1000 examples of transformations per dependency relation. To avoid confounding factors, each instance has only a single edgewise transformation (for the specific dependency relation).

When we mask random attention heads or random look-up heads, it is computationally too expensive to do this for all possible attention heads and we approximate this with a Monte Carlo estimate: we select random heads 20 times and take the average of the results.

Analysis of fine-tuned models on synthetic data When generating synthetic downstream tasks, we exclude the CONCAT operation for edgewise transformations. We take a sample of 5000 sentences from our parsed corpus and randomly divide it into an 80/20 train/test split. We use a prefix of tunable embeddings of the same length as the ground truth, i.e. we set it to a length of 5. When masking random (look-up) heads, we repeat this 50 times to estimate the expected change in accuracy.

E.3 Additional Results

We report results full results on the syntactic transformation tasks in Table E.1 (including standard deviations). For SLOG, results by generalization case can be found in Table E.2.

E.4 Hyperparameters

Pretraining When generating pretraining data for STEP, we only use sentences with 90 or less tokens (in terms of the T5 tokenizer) and exclude any instances with outputs of 180 T5 tokens or more. However, we do not impose a limit on the length of the output for our baselines (T5+Dep Parse and Simple STEP) because it would remove too much of the pretraining data. We use Adafactor for our intermediate pretraining with a learning rate of 3×10^{-4} and without warm-up, and a batch size of 80 (for STEP), 30 (for Simple STEP) and 48 (for T5+Dep Parse). We maintain separate optimizers for the main objective (e.g. predicting the transformation) and the original span-denoising objective. We train for a single epoch, except for T5+Dep Parse, which we train for

Task	Model	Acc ↑	BLEU ↑	TER ↓
Adj. emphasis	STEP	10.9±1.0	52.3±0.7	33.5±0.6
	Simple STEP	9.8±0.8	48.3±0.8	37.5±0.9
	T5	7.3±0.8	47.6±0.8	38.3±0.8
	T5+Dep Parse	7.7±0.8	45.8±0.9	40.4±1.1
Passivization	STEP	57.9±2.0	84.8±0.7	8.4±0.5
	Simple STEP	46.8±2.2	78.4±0.8	13.6±0.6
	T5	40.2±1.7	73.7±0.8	18.3±0.7
	T5+Dep Parse	45.0±1.6	76.8±0.5	15.5±0.4
Verb emphasis	STEP	3.4±0.4	41.8±0.6	45.6±0.4
	Simple STEP	3.6±0.7	40.5±0.6	47.0±0.6
	T5	3.5±0.4	41.7±0.5	46.7±0.4
	T5+Dep Parse	3.3±0.7	40.1±0.8	48.2±0.6

Table E.1: Evaluation on 100-shot **syntactic transformation** tasks. We report averages of 10 draws of 100 training examples each. We also include standard deviations on the results across the 10 runs.

two epochs. This is because STEP and Simple STEP have two instances with syntactic transformations per parsed sentence but T5+Dep Parse only has a single one. For the denoising objective, we impose a limit of 80 tokens per instance (truncating longer instances) and use a batch size of 50.

Fine-tuning During fine-tuning, the main hyperparameters are the learning rates. We use Adafactor for fine-tuning using a learning rate of 1×10^{-4}. For the prefix of STEP, we use a learning rate of 10. These hyperparameters apply to all experiments and all models, except for SLOG, as described below:

SLOG We found that accuracy on SLOG was very sensitive to hyperparameters and used a hyperparameter selection strategy similar to that of Conklin et al. (2021) for COGS: we draw a sample of around 10% from the generalization data. We fixed one random seed and ran 10 randomly sampled hyperparameter configurations and selected the one with the highest accuracy that was most stable across the epochs. We then discarded that random seed and used different ones for fine-tuning the model. We sample the learning rate from LogUniform$[2 \times 10^{-6}, 1 \times 10^{-4}]$ and the batch size uniformly from $[24, 48, 72, 96, 120]$. STEP also has an additional learning rate for the prefix, which we sample from LogUniform$[0.1, 10]$ during the search. The chosen hyperparameters can be found in Table E.3.

Number of parameters T5-base has 222 million parameters. When we fine-tune STEP with a prefix of tunable embeddings, this adds 7860 parameters to that, which is an increase of 0.035 ‰.

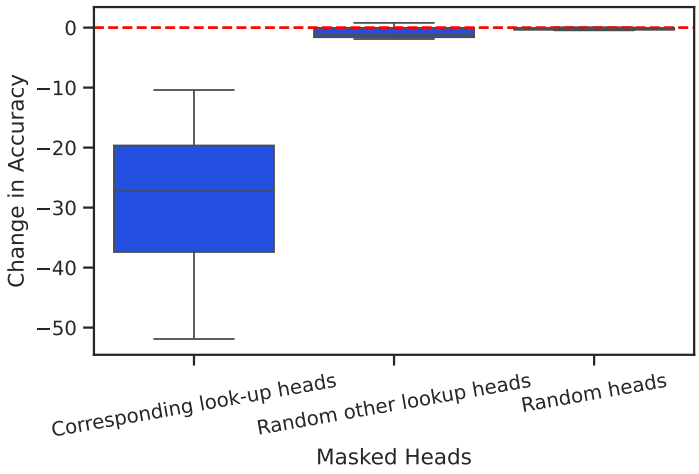


Figure E.1: Effect of masking look-up heads of models fine-tuned on synthetic tasks. The boxplot shows the distribution for 10 synthetic downstream tasks.

E.5 Evaluation Metrics

We use SacreBLEU (Post, 2018) v2.3 to compute BLEU and TER. For the experiments with ATIS, we use the code of Chapter 3 for computing accuracy.

SLOG Li et al. (2023a) argue for using semantic equivalence for evaluation but they focus on a variable-based formalism and use exact string match for evaluating the variable-free representation. We take semantic equivalence into account, in particular, the order of the children does not matter because the roles are represented in the logical form. Hence, `offer ( theme = donut , recipient = * turtle )` and `offer ( recipient = * turtle , theme = donut )` are equivalent. We achieve this by parsing the string representation into a tree and instead of a list of children we maintain a set of children, and then compare trees to evaluate accuracy.

E.6 Run Times

As in the other chapters, all our experiments were run on Nvidia 2080TI or 1080TI GPUs with 12 GB RAM. Pretraining STEP took around 30 hours. Since we used longer maximum sequence lengths for the baselines, and had to decrease the physical batch size, training of the baselines took 50 (T5+Dep Parse) and 95 hours (Simple STEP).

Generalization	**STEP**	**Simple STEP**	**T5**	**T5+Dep Parse**
Deeper CP tail recursion	**74.5**±4.2	73.0±8.4	42.5±4.4	50.9±9.4
Deeper PP recursion	**87.3**±2.5	78.5±4.4	75.0±4.5	70.8±5.9
Deeper center embedding	17.3±2.8	**22.7**±2.2	8.9±0.4	11.5±2.1
Shallower CP tail recursion	**100.0**±0.0	**100.0**±0.0	**100.0**±0.0	**100.0**±0.0
Shallower PP recursion	**100.0**±0.0	**100.0**±0.0	99.9±0.2	**100.0**±0.0
Shallower center embedding	**100.0**±0.0	**100.0**±0.0	**100.0**±0.0	**100.0**±0.0
PP in indirect object NPs	99.5±0.4	99.8±0.1	**100.0**±0.0	99.7±0.1
PP in subject NPs	**95.3**±0.1	**95.3**±0.1	74.5±8.2	**95.3**±0.2
RC in indirect object NPs	65.7±0.8	70.7±0.8	**73.0**±0.8	67.1±1.2
RC in subject NPs	**89.4**±0.9	88.4±1.4	70.4±2.6	69.1±12.3
Indirect object-extracted RC	16.5±13.0	11.7±13.9	62.9±12.3	**73.8**±13.2
Indirect object wh-questions	78.8±18.3	77.8±14.6	**100.0**±0.1	99.8±0.2
Direct object wh-questions	83.4±14.5	91.0±4.5	**99.2**±0.6	83.8±18.0
Wh-questions long movement	**55.8**±12.5	46.2±7.5	40.4±8.1	35.2±6.2
Wh-questions with modified NPs	**85.9**±2.7	85.2±1.9	72.9±4.8	77.7±2.8
Active subject wh-questions	**100.0**±0.1	99.1±1.8	99.6±0.3	97.1±5.6
Passive subject wh-questions	98.8±2.4	99.5±0.6	**100.0**±0.0	**100.0**±0.0
Average	**79.3**±2.3	78.8±1.9	77.6±1.4	78.3±2.1

Table E.2: Full SLOG results.

Model	**Epochs**	**Batch size**	**LR**	**LR Prefix**
T5	50	96	1.62E-05	-
T5+ Dep Parse	50	24	9.51E-05	-
Simple STEP	22	72	6.75E-05	-
STEP	15	48	1.30E-05	2.52

Table E.3: Hyperparameters used for **SLOG**. LR is learning rate.

Bibliography

Samira Abnar, Mostafa Dehghani, and Willem Zuidema. 2020. Transferring inductive biases through knowledge distillation. *arXiv preprint arXiv:2006.00555*.

Ekin Akyürek and Jacob Andreas. 2021. Lexicon learning for few shot sequence modeling. In *Proceedings of the ACL-IJCNLP (Volume 1: Long Papers)*, pages 4934–4946, Online.

Ekin Akyurek and Jacob Andreas. 2023. LexSym: Compositionality as lexical symmetry. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. Openfst: A general and efficient weighted finite-state transducer library: (extended abstract of an invited talk). In *Implementation and Application of Automata: 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers 12*, pages 11–23. Springer.

Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. 2017. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. *Advances in neural information processing systems*, 30.

Rajeev Alur, Dana Fisman, Rishabh Singh, and Armando Solar-Lezama. 2017. Sygus-comp 2017: Results and analysis. *arXiv preprint arXiv:1711.11438*.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the ACL*, pages 7556–7566, Online.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. 2019. How to train your MAML. In *International Conference on Learning Representations*.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate.

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*, Sofia, Bulgaria. Association for Computational Linguistics.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2).

Trapit Bansal, Salaheddin Alzubi, Tong Wang, Jay-Yoon Lee, and Andrew McCallum. 2022. Meta-adapters: Parameter efficient few-shot fine-tuning through meta-learning. In *International Conference on Automated Machine Learning*, pages 19–1. PMLR.

Elisa Bassignana, Filip Ginter, Sampo Pyysalo, Rob van der Goot, and Barbara Plank. 2023. Silver syntax pre-training for cross-domain relation extraction. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6984–6993, Toronto, Canada. Association for Computational Linguistics.

Elisa Bassignana and Barbara Plank. 2022. CrossRE: A cross-domain dataset for relation extraction. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3592–3604, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Leonard E Baum. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3(1):1–8.

Heinz H Bauschke and Jonathan M Borwein. 1996. On projection algorithms for solving convex feasibility problems. *SIAM review*, 38(3):367–426.

Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. 2015. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138.

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*, volume 35, pages 12564–12573.

David Bieber, Rishab Goel, Dan Zheng, Hugo Larochelle, and Daniel Tarlow. 2022. Static prediction of runtime errors by learning to execute programs with external resource descriptions. In *Deep Learning for Code Workshop*.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. 2022. Efficient and modular implicit differentiation. *Advances in neural information processing systems*, 35:5230–5242.

Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. 1998. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283.

Lev M Bregman. 1967. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ziqiang Cao, Furu Wei, Wenjie Li, and Sujian Li. 2018. Faithful to the original: Fact aware neural abstractive summarization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Francesco Cazzaro, Davide Locatelli, and Ariadna Quattoni. 2024. Align and augment: Generative data augmentation for compositional generalization. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 369–383, St. Julian's, Malta. Association for Computational Linguistics.

Francesco Cazzaro, Davide Locatelli, Ariadna Quattoni, and Xavier Carreras. 2023. Translate first reorder later: Leveraging monotonicity in semantic parsing. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 227–238, Dubrovnik, Croatia. Association for Computational Linguistics.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Ann Arbor, Michigan. Association for Computational Linguistics.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland. Association for Computational Linguistics.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT press.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Florence, Italy. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4).

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. In *Proceedings of the 2016 Conference of the North American Chapter of the ACL: Human Language Technologies*, pages 876–885, San Diego, California.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. 2021. Meta-learning to compositionally generalize. In *Proceedings of the ACL-IJCNLP (Volume 1: Long Papers)*, pages 3322–3335, Online.

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 EMNLP*, pages 619–634, Online and Punta Cana, Dominican Republic.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 2022. The neural data router: Adaptive control flow in transformers improves systematic generalization. In *International Conference on Learning Representations*.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

John M. Danskin. 1967. *The Theory of Max-Min and its Application to Weapons Allocation Problems*. Springer Berlin Heidelberg.

Donald Davidson. 1967. The logical form of action sentences. *The Logic of Decision and Action*.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22.

Nicolas Devatine, Caio Corro, and François Yvon. 2022. Ré-ordonnancement via programmation dynamique pour l'adaptation cross-lingue d'un analyseur en dépendances (sentence reordering via dynamic programming for cross-lingual dependency parsing). In *Actes de la 29e Conférence sur le Traitement Automatique des Langues Naturelles. Volume 1 : conférence principale*, Avignon, France. ATALA.

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)*, Minneapolis, Minnesota. Association for Computational Linguistics.

Kuicai Dong, Aixin Sun, Jung-jae Kim, and Xiaoli Li. 2023. Open information extraction via chunks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15390–15404, Singapore. 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)*, Berlin, Germany. Association for Computational Linguistics.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models. *arXiv preprint arXiv:2209.15003*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. 2023. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*.

Jason Eisner and Roy W Tromble. 2006. Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pages 57–75.

Javier Ferrando, Gerard I. Gállego, Belen Alastruey, Carlos Escolano, and Marta R. Costa-jussà. 2022. Towards opening the black box of neural machine translation: Source and target interpretations of the transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Pedro Ferreira, Ivan Titov, and Wilker Aziz. 2025. Explanation regularisation through the lens of attributions. In *Proceedings of the 31st International Conference on Computational Linguistics*, Abu Dhabi, UAE. 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 ACL (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia.

Matthew Finlayson, Kyle Richardson, Ashish Sabharwal, and Peter Clark. 2022. What makes instruction learning hard? an investigation and a new challenge in a synthetic environment. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 414–426, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

Chelsea B. Finn. 2018. *Learning to Learn with Gradients*. Ph.D. thesis.

Jerry A Fodor. 1975. *The language of thought*, volume 5. Harvard university press.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

Kuzman Ganchev, Joao Graça, Jennifer Gillenwater, and Ben Taskar. 2010. Posterior regularization for structured latent variable models. *The Journal of Machine Learning Research*, 11:2001–2049.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

Michael Goodale, Salvador Mascarenhas, and Yair Lakretz. 2025. Meta-learning neural mechanisms rather than bayesian priors. *arXiv preprint arXiv:2503.16048*.

Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. 2010. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14(8):357–364.

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.

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 ACL (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany.

Jiaqi Guo, Qian Liu, Jian-Guang Lou, Zhenwen Li, Xueqing Liu, Tao Xie, and Ting Liu. 2020. Benchmarking meaning representations in neural semantic parsing. In *Proceedings of the 2020 EMNLP (EMNLP)*, pages 1520–1540, Online.

Michael Hahn and Yang Xu. 2022. Crosslinguistic word order variation reflects evolutionary pressures of dependency and information locality. *Proceedings of the National Academy of Sciences*, 119(24):e2122604119.

Hardy Hardy and Andreas Vlachos. 2018. Guided neural language generation for abstractive summarization using Abstract Meaning Representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2021. Span-based semantic parsing for compositional generalization. In *Proceedings of the ACL-IJCNLP (Volume 1: Long Papers)*, pages 908–921, Online.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Unlocking compositional generalization in pre-trained models using intermediate representations. *arXiv preprint arXiv:2104.07478*.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. 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 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Saghar Hosseini, Ahmed Hassan Awadallah, and Yu Su. 2021. Compositional generalization for natural language interfaces to web apis. *arXiv preprint arXiv:2112.05209*.

I-Hung Hsu, Zhiyu Xie, Kuan-Hao Huang, Prem Natarajan, and Nanyun Peng. 2023. AMPERE: AMR-aware prefix for generation-based event argument extraction model. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: how do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795.

Dieuwke Hupkes, Mario Giulianelli, Verna Dankers, Mikel Artetxe, Yanai Elazar, Tiago Pimentel, Christos Christodoulopoulos, Karim Lasri, Naomi Saphra, Arabella Sinclair, et al. 2023. A taxonomy and review of generalization research in nlp. *Nature Machine Intelligence*, 5(10):1161–1174.

Dora Jambor and Dzmitry Bahdanau. 2022. LAGr: Label aligned graphs for better systematic generalization in semantic parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Dublin, Ireland. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany. Association for Computational Linguistics.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Hans Kamp and Uwe Reyle. 2013. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer Science & Business Media.

Fred Karlsson. 2007. Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 43(2):365–392.

Adam Karvonen. 2024. Emergent world models and latent variable estimation in chess-playing language models. *Preprint*, arXiv:2403.15498.

Rohit J. Kate, Yuk Wah, Wong Raymond, and J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of AAAI-05*.

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

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 EMNLP (EMNLP)*, pages 9087–9105, Online.

Najoung Kim, Tal Linzen, and Paul Smolensky. 2022. Uncontrolled lexical exposure leads to overestimation of compositional generalization in pretrained models. *arXiv preprint arXiv:2212.10769*.

Yoon Kim. 2021. Sequence-to-sequence learning with latent neural grammars. In *Advances in Neural Information Processing Systems*, volume 34, pages 26302–26317. Curran Associates, Inc.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

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

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, Prague, Czech Republic. Association for Computational Linguistics.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Kundan Krishna, Jeffrey Bigham, and Zachary C. Lipton. 2021. Does pretraining for summarization require knowledge transfer? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3178–3189, Punta Cana, Dominican Republic. Association for Computational Linguistics.

David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. 2021. Out-of-distribution generalization via risk extrapolation (rex). In *International conference on machine learning*, pages 5815–5826. PMLR.

Harold W Kuhn. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Yam Kushinsky, Haggai Maron, Nadav Dym, and Yaron Lipman. 2019. Sinkhorn algorithm for lifted assignment problems. *SIAM Journal on Imaging Sciences*, 12(2):716–735.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Brenden M Lake. 2019. Compositional generalization through meta sequence-to-sequence learning. *Advances in neural information processing systems*, 32.

Brenden M Lake and Marco Baroni. 2023. Human-like systematic generalization through a meta-learning neural network. *Nature*, 623(7985):115–121.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. 2017. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6.

Jackson L. Lee, Lucas F.E. Ashby, M. Elizabeth Garza, Yeonju Lee-Sikka, Sean Miller, Alan Wong, Arya D. McCarthy, and Kyle Gorman. 2020. Massively multilingual pronunciation modeling with WikiPron. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4223–4228, Marseille, France. European Language Resources Association.

Katherine Lee, Orhan Firat, Ashish Agarwal, Clara Fannjiang, and David Sussillo. 2018. Hallucinations in neural machine translation. In *NIPS 2018 Interpretability and Robustness for Audio, Speech and Language Workshop*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Itay Levy, Ben Bogin, and Jonathan Berant. 2023. Diverse demonstrations improve in-context compositional generalization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. 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*, Online. Association for Computational Linguistics.

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

Mike Lewis and Mark Steedman. 2014. A* CCG parsing with a supertag-factored model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. Association for Computational Linguistics.

Philip M Lewis and Richard Edwin Stearns. 1968. Syntax-directed transduction. *Journal of the ACM (JACM)*, 15(3):465–488.

Bingzhi Li, Lucia Donatelli, Alexander Koller, Tal Linzen, Yuekun Yao, and Najoung Kim. 2023a. SLOG: A structural generalization benchmark for semantic parsing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3213–3232, Singapore. Association for Computational Linguistics.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2023b. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*.

Xiang Li, Thien Huu Nguyen, Kai Cao, and Ralph Grishman. 2015. Improving event detection with Abstract Meaning Representation. In *Proceedings of the First Workshop on Computing News Storylines*, Beijing, China. Association for Computational Linguistics.

Yafu Li, Yongjing Yin, Yulong Chen, and Yue Zhang. 2021. On compositional generalization of neural machine translation. 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 4767–4780, Online. Association for Computational Linguistics.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Xin Zhao, and Ji-Rong Wen. 2023c. Evaluating object hallucination in large vision-language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore. Association for Computational Linguistics.

Zhaoyi Li, Ying Wei, and Defu Lian. 2023d. Learning to substitute spans towards improving compositional generalization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2791–2811, Toronto, Canada. Association for Computational Linguistics.

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*, Portland, Oregon, USA. Association for Computational Linguistics.

Percy Liang, Ben Taskar, and Dan Klein. 2006. Alignment by agreement. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, page 104.

Matthias Lindemann, Alexander Koller, and Ivan Titov. 2023a. Compositional generalisation with structured reordering and fertility layers. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2172–2186, Dubrovnik, Croatia. Association for Computational Linguistics.

Matthias Lindemann, Alexander Koller, and Ivan Titov. 2023b. Compositional generalization without trees using multiset tagging and latent permutations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14488–14506, Toronto, Canada. Association for Computational Linguistics.

Matthias Lindemann, Alexander Koller, and Ivan Titov. 2024a. SIP: Injecting a structural inductive bias into a Seq2Seq model by simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand. Association for Computational Linguistics.

Matthias Lindemann, Alexander Koller, and Ivan Titov. 2024b. Strengthening structural inductive biases by pre-training to perform syntactic transformations. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Miami, Florida, USA. Association for Computational Linguistics.

Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2022. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*.

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

Jiashuo Liu, Zheyan Shen, Yue He, Xingxuan Zhang, Renzhe Xu, Han Yu, and Peng Cui. 2023. Towards out-of-distribution generalization: A survey. *arXiv preprint arXiv:2108.13624v2*.

Wei Liu, Sihan Chen, Longteng Guo, Xinxin Zhu, and Jing Liu. 2021b. Cptr: Full transformer network for image captioning. *arXiv preprint arXiv:2101.10804*.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Chunchuan Lyu, Shay B. Cohen, and Ivan Titov. 2021a. A differentiable relaxation of graph segmentation and alignment for AMR parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chunchuan Lyu and Ivan Titov. 2018. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia. Association for Computational Linguistics.

Yiwei Lyu, Paul Pu Liang, Hai Pham, Eduard Hovy, Barnabás Póczos, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2021b. StylePTB: A compositional benchmark for fine-grained controllable text style transfer. In *Proceedings of the 2021 Conference of the North American Chapter of the ACL: Human Language Technologies*, pages 2116–2138, Online.

Chaitanya Malaviya, Pedro Ferreira, and André F. T. Martins. 2018. Sparse and constrained attention for neural machine translation. In *Proceedings of the 56th Annual Meeting of the ACL (Volume 2: Short Papers)*, pages 370–376, Melbourne, Australia.

R Thomas McCoy, Erin Grant, Paul Smolensky, Thomas L Griffiths, and Tal Linzen. 2020. Universal linguistic inductive biases via meta-learning. In *Proceedings of the 42nd Annual Conference of the Cognitive Science Society*.

R Thomas McCoy and Thomas L Griffiths. 2023. Modeling rapid language learning by distilling bayesian priors into artificial neural networks. *arXiv preprint arXiv:2305.14701*.

Drew M McDermott. 2000. The 1998 ai planning systems competition. *AI magazine*, 21(2):35–35.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. 2018. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*.

William Merrill and Ashish Sabharwal. 2024. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32.

Stoyan Mihov and Klaus U. Schulz. 2019. *Finite-State Techniques: Automata, Transducers and Bimachines*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2018. A simple neural attentive meta-learner. In *International Conference on Learning Representations*.

Richard Montague. 1970. English as a formal language. In Bruno Visentini, editor, *Linguaggi nella societa e nella tecnica*, pages 188–221. Edizioni di Communita.

Aaron Mueller, Robert Frank, Tal Linzen, Luheng Wang, and Sebastian Schuster. 2022. Coloring the blank slate: Pre-training imparts a hierarchical inductive bias to sequence-to-sequence models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1352–1368, Dublin, Ireland. Association for Computational Linguistics.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. 2022. Transformers can do bayesian inference. In *International Conference on Learning Representations*.

Karl Mulligan, Robert Frank, and Tal Linzen. 2021. Structure here, bias there: Hierarchical generalization by jointly learning syntactic transformations. In *Proceedings of the Society for Computation in Linguistics 2021*, pages 125–135, Online. Association for Computational Linguistics.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient transformers with dynamic token pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Benjamin Newman, John Hewitt, Percy Liang, and Christopher D. Manning. 2020. The EOS decision and length extrapolation. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 276–291, Online.

Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh, and Thien Huu Nguyen. 2021. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 80–90, Online. Association for Computational Linguistics.

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1).

Karel Oliva. 1988. Syntactic functions in GPSG. In *Coling Budapest 1988 Volume 2: International Conference on Computational Linguistics*.

OLMo Team, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2024. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Isabel Papadimitriou and Dan Jurafsky. 2023. Injecting structural hints: Using language models to study inductive biases in language learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8402–8413, Singapore. Association for Computational Linguistics.

George Papandreou and Alan L Yuille. 2011. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 international conference on computer vision*, pages 193–200. IEEE.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Terence Parsons. 1990. Events in the semantics of english: A study in subatomic semantics.

Barbara Partee. 1984. Compositionality. *Varieties of Formal Semantics*, 3.

Max Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J Maddison. 2020. Gradient estimation with stochastic softmax tricks. *Advances in Neural Information Processing Systems*, 33:5691–5704.

Kexin Pei, Jonas Guan, Matthew Broughton, Zhongtian Chen, Songchen Yao, David Williams-King, Vikas Ummadisetty, Junfeng Yang, Baishakhi Ray, and Suman Jana. 2021. Stateformer: Fine-grained type recovery from binaries using generative state modeling. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 690–702, New York, NY, USA. Association for Computing Machinery.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 EMNLP (EMNLP)*, pages 1532–1543, Doha, Qatar.

Alban Petit and Caio Corro. 2023. On graph-based reentrancy-free semantic parsing. *Transactions of the Association for Computational Linguistics*.

Alban Petit, Caio Corro, and François Yvon. 2023. Structural generalization in COGS: Supertagging is (almost) all you need. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1089–1101, Singapore. Association for Computational Linguistics.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? 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)*, Hong Kong, China. Association for Computational Linguistics.

Steven T Piantadosi, Joshua B Tenenbaum, and Noah D Goodman. 2016. The logical primitives of thought: Empirical foundations for compositional cognitive models. *Psychological review*, 123(4):392.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Alan Prince and Paul Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. 2020. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online. Association for Computational Linguistics.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Pawel Nowak, Tal Linzen, Fei Sha, and Kristina Toutanova. 2022a. Improving compositional generalization with latent structure and data augmentation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4341–4362, Seattle, United States. Association for Computational Linguistics.

Linlu Qiu, Peter Shaw, Panupong Pasupat, Tianze Shi, Jonathan Herzig, Emily Pitler, Fei Sha, and Kristina Toutanova. 2022b. Evaluating the impact of model scale for compositional generalization in semantic parsing. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9157–9179, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Colin Raffel, Minh-Thang Luong, Peter J Liu, Ron J Weiss, and Douglas Eck. 2017. Online and linear-time attention by enforcing monotonic alignments. In *International Conference on Machine Learning*, pages 2837–2846. PMLR.

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

Alan Ramponi and Barbara Plank. 2020. Neural unsupervised domain adaptation in NLP—A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Florencia Reali and Thomas L Griffiths. 2009. The evolution of frequency distributions: Relating regularization to inductive biases through iterated learning. *Cognition*, 111(3):317–328.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4.

Jonathan Richens and Tom Everitt. 2024. Robust agents learn causal world models. In *The Twelfth International Conference on Learning Representations*.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. In *Proceedings of the web conference 2020*, pages 2962–2968.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. 2023. Randomized positional encodings boost length generalization of transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Toronto, Canada. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Marcel-Paul Schützenberger. 1961. A remark on finite transducers. *Information and Control*, 4:185–196.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany. Association for Computational Linguistics.

Louis Shapiro and A. B. Stephens. 1991. Bootstrap percolation, the schröder numbers, and the n-kings problem. *SIAM Journal on Discrete Mathematics*, 4(2):275–280.

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 ACL-IJCNLP (Volume 1: Long Papers)*, pages 922–938, Online.

Richard Sinkhorn. 1964. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems*, volume 35, pages 9460–9471. Curran Associates, Inc.

David Smith and Jason Eisner. 2006. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 23–30, New York City.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Paul Soulos, Henry Conklin, Mattia Opper, Paul Smolensky, Jianfeng Gao, and Roland Fernandez. 2024. Compositional generalization across distributional shifts with sparse tree operations. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

J. F. Staal. 1963. Sanskrit and sanskritization. *The Journal of Asian Studies*, 22(3):261–275.

Miloš Stanojević and Khalil Sima'an. 2015. Reordering grammar induction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal. Association for Computational Linguistics.

Miloš Stanojević and Mark Steedman. 2021. Formal basis of a language universal. *Computational Linguistics*, 47(1).

Mark Steedman. 1996. Surface structure and interpretation. *Linguistic Inquiry Monograph*, No. 30.

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

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

Sara Stymne, Miryam de Lhoneux, Aaron Smith, and Joakim Nivre. 2018. Parser training with heterogeneous treebanks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 619–625, Melbourne, Australia. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A. Smith. 2018. Syntactic scaffolds for semantic structures. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3772–3782, Brussels, Belgium. 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.

The Unicode Consortium. 2024. *The Unicode Standard, Version 16.0 – Core Specification*, chapter 12. The Unicode Consortium.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task chunking. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Yulia Tsvetkov, Sunayana Sitaram, Manaal Faruqui, Guillaume Lample, Patrick Littell, David Mortensen, Alan W Black, Lori Levin, and Chris Dyer. 2016. Polyglot neural language models: A case study in cross-lingual phonetic representation learning. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1357–1366, San Diego, California. Association for Computational Linguistics.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 76–85, Berlin, Germany.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *Computational Linguistics in the Netherlands Journal*, 7:93–108.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

John von Neumann. 1951. *Functional Operators, Volume 2*. Princeton University Press, Princeton.

Bailin Wang, Mirella Lapata, and Ivan Titov. 2021a. Structured reordering for modeling latent alignments in sequence transduction. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

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*, Online. Association for Computational Linguistics.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021b. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online. Association for Computational Linguistics.

Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. 2022. What language model architecture and pretraining objective works best for zero-shot generalization? In *International Conference on Machine Learning*, pages 22964–22984. PMLR.

Shira Wein and Juri Opitz. 2024. A survey of AMR applications. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Miami, Florida, USA. Association for Computational Linguistics.

Pia Weißenhorn, Lucia Donatelli, and Alexander Koller. 2022. Compositional generalization with a broad-coverage semantic parser. In *Proceedings of the 11th Joint Conference on Lexical and Computational Semantics*, pages 44–54, Seattle, Washington. Association for Computational Linguistics.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Terry Winograd. 1971. *Procedures as a representation for data in a computer program for understanding natural language*. Ph.D. thesis.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 439–446, New York City, USA.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.

Shijie Wu and Ryan Cotterell. 2019. Exact hard monotonic attention for character-level transduction. In *Proceedings of the 57th Annual Meeting of the ACL*, pages 1530–1537, Florence, Italy.

Shijie Wu, Ryan Cotterell, and Mans Hulden. 2021a. Applying the transformer to character-level transduction. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, Online. Association for Computational Linguistics.

Yuhuai Wu, Felix Li, and Percy S Liang. 2022. Insights into pre-training via simpler synthetic tasks. *Advances in Neural Information Processing Systems*, 35:21844–21857.

Yuhuai Wu, Markus N Rabe, Wenda Li, Jimmy Ba, Roger B Grosse, and Christian Szegedy. 2021b. Lime: Learning inductive bias for primitives of mathematical reasoning. In *International Conference on Machine Learning*, pages 11251–11262. PMLR.

Zhengxuan Wu, Christopher D. Manning, and Christopher Potts. 2023a. ReCOGS: How incidental details of a logical form overshadow an evaluation of semantic interpretation. *Transactions of the Association for Computational Linguistics*, 11.

Zhengxuan Wu, Christopher D Manning, and Christopher Potts. 2023b. Recogs: How incidental details of a logical form overshadow an evaluation of semantic interpretation. *arXiv preprint arXiv:2303.13716*.

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proceedings of the First International Conference on Human Language Technology Research*.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020. Improving AMR parsing with sequence-to-sequence pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2501–2511, Online. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online. Association for Computational Linguistics.

Jingfeng Yang, Le Zhang, and Diyi Yang. 2022. SUBS: Subtree substitution for compositional semantic parsing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 169–174, Seattle, United States. Association for Computational Linguistics.

Yuekun Yao and Alexander Koller. 2022. Structural generalization is hard for sequence-to-sequence models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Yuekun Yao and Alexander Koller. 2024. Simple and effective data augmentation for compositional generalization. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 434–449, Mexico City, Mexico. Association for Computational Linguistics.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online. Association for Computational Linguistics.

Yongjing Yin, Jiali Zeng, Yafu Li, Fandong Meng, Jie Zhou, and Yue Zhang. 2023. Consistency regularization training for compositional generalization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Lei Yu, Jan Buys, and Phil Blunsom. 2016. Online segment to segment neural transduction. In *Proceedings of the 2016 EMNLP*, pages 1307–1316, Austin, Texas.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada. Association for Computational Linguistics.

Wojciech Zaremba and Ilya Sutskever. 2014. Learning to execute. *arXiv preprint arXiv:1410.4615*.

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 Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, Czech Republic. Association for Computational Linguistics.

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.

Hao Zhang and Daniel Gildea. 2007. Factorization of synchronous context-free grammars in linear time. In *Proceedings of SSST, NAACL-HLT 2007 / AMTA Workshop on Syntax and Structure in Statistical Translation*, Rochester, New York. Association for Computational Linguistics.

Shuai Zhang, Wang Lijie, Xinyan Xiao, and Hua Wu. 2022a. Syntax-guided contrastive learning for pre-trained language model. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2430–2440, Dublin, Ireland. Association for Computational Linguistics.

Zhuosheng Zhang, Shuohang Wang, Yichong Xu, Yuwei Fang, Wenhao Yu, Yang Liu, Hai Zhao, Chenguang Zhu, and Michael Zeng. 2022b. Task compass: Scaling multi-task pre-training with task prefix. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5671–5685, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Hao Zheng and Mirella Lapata. 2021. Compositional generalization via semantic tagging. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1022–1032, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hao Zheng and Mirella Lapata. 2022. Disentangled sequence to sequence learning for compositional generalization. In *Proceedings of the 60th Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 4256–4268, Dublin, Ireland.

Hao Zheng and Mirella Lapata. 2023. Real-world compositional generalization with disentangled sequence-to-sequence learning. In *Findings of the Association for Computational Linguistics: ACL 2023*, Toronto, Canada. Association for Computational Linguistics.

Junru Zhou, Zhuosheng Zhang, Hai Zhao, and Shuailiang Zhang. 2020. LIMIT-BERT : Linguistics informed multi-task BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4450–4461, Online. Association for Computational Linguistics.

Jian Zhu, Cong Zhang, and David Jurgens. 2022. ByT5 model for massively multilingual grapheme-to-phoneme conversion. In *Proc. Interspeech 2022*, pages 446–450.