

Systematic Learning

Tim Klinger
IBM Research



LLMs can do wonderful things but...

Google DeepMind

2024-2-15

Premise Order Matters in Reasoning with Large Language Models

Xinyun Chen^{1*}, Ryan A. Chi^{1,2*}, Xuezhi Wang¹ and Denny Zhou¹

*Equal contribution, ¹Google DeepMind, ²Stanford University
{xinyunchen, xuezhiw, dennyzhou}@google.com, ryanchi@cs.stanford.edu

Large language models (LLMs) have shown remarkable performance in various reasoning tasks. However, in the domain of reasoning, they often fail to maintain consistency when the order of premises is changed, despite being trained to solve such problems.

Faith and Fate: Limits of Transformers on Compositionality

FUND/
IN LAI

When can transformers reason with abstract symbols?

Enric Boix-Adserà^{1,2} Omid Saremi¹ Emmanuel Abbe^{1,3}
Samy Bengio¹ Eta Littwin¹ Joshua Susskind¹

¹Apple ²MIT ³EPFL
eboix@mit.edu, emmanuel.abbe@epfl.ch
{osaremi, bengio, elittwin, jsusskind}@apple.com

October 17, 2023

Abstract

We investigate the capabilities of transformer large language models (LLMs) on *relational reasoning* tasks involving abstract symbols. Such tasks have long been studied in the neuroscience literature as fundamental building blocks for more complex abilities in programming, mathematics, and verbal reasoning. For (i) regression tasks, we prove that transformers generalize when trained, but require astonishingly large quantities of training data. For (ii) next-token-prediction tasks with symbolic labels, we show an “inverse scaling law”: transformers fail to generalize as their embedding dimension increases. For both settings (i) and (ii), we propose subtle transformer modifications which can reduce the amount of data needed by adding two trainable parameters per head.

PAR AI New York Univers

Limits of Transformer Language Models on Lea

Jonathan Thomm^{1,2} Aleksandar Terzic^{1,2} Geethan Karuna
Bernhard Schölkopf^{2,3} Abbas I

WHAT ALGORITHMS CAN TRANSFORMERS LEARN? A STUDY IN LENGTH GENERALIZATION

Hattie Zhou^{*1,2}, Arwen Bradley¹, Eta Littwin¹, Noam Razin^{*1,3}, Omid Saremi¹, ¹g Large Language
Josh Susskind¹, Samy Bengio¹, and Preetum Nakkiran¹

Trained to Solve

v Hardy Thomas L. Griffiths

¹Apple

²Mila, Université de Montréal

³Tel Aviv University

we trained them to be.

nt to recognize their strengths and these systems we need to consider internet text. By recognizing the that LLMs will adopt, allowing us ll the teleological approach—leads

Abstract

Large language models exhibit surprising emergent generalization properties, yet also struggle on many simple reasoning tasks such as arithmetic and parity. This raises the question of if and when Transformer models can learn the true algorithm for solving a task. We study the scope of Transformers’ abilities in the specific setting of length generalization on algorithmic tasks. Here, we propose a unifying framework to understand when and how Transformers can exhibit strong length generalization on a given task. Specifically, we leverage RASP (Weiss, Lukas Berglund*, Meg Tong[†], Max Kaufmann^{††}, Mikita Balesni^{§‡})

Asa Cooper Stickland^{¶†} Tomasz Korbak^{††} Owain Evans^{‡‡}

^{*}Vanderbilt University [†]Independent [‡]UK Frontier AI Taskforce [§]Apollo Research
[¶]New York University ^{††}University of Sussex ^{‡‡}University of Oxford

Abstract

We expose a surprising failure of generalization in auto-regressive large language models (LLMs). If a model is trained on a sentence of the form “A is B”, it will not automatically generalize to the reverse direction “B is A”. This is the **Reversal Curse**. For instance, if a model is trained on “Olaf Scholz was the ninth Chancellor of Germany”, it will not automatically be able to answer the question, “Who was the ninth Chancellor of Germany?”. Moreover, the likelihood of the correct answer (“Olaf Scholz”) will not be higher than for a random name. Thus, models exhibit a basic failure of logical deduction and do not generalize a prevalent pattern in their training set (i.e. if “A is B” occurs, “B is A” is more likely to occur).

Marquez
nputting & AI
iversity, Tempe,
casu.edu

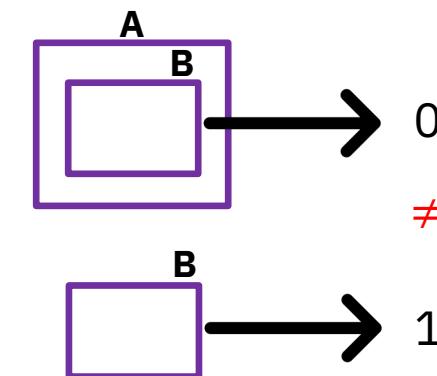
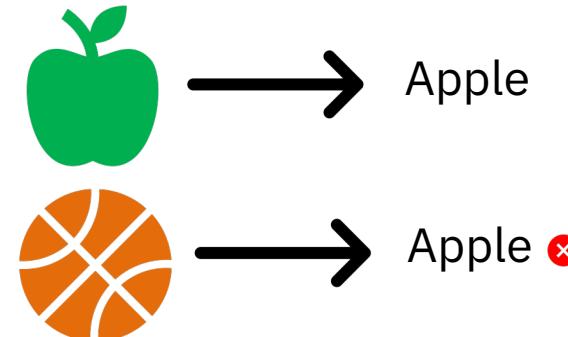
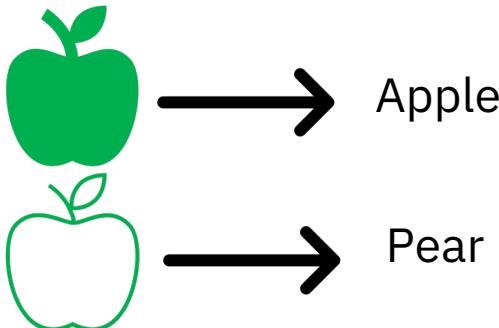
edharan^{*}
nputer Science,
rsity, Fort Collins.
t@colostate.edu

Agenda

- Discuss examples of LLM failures of systematicity
- Desiderata for systematic learning
- Compositional program generation (CPG) as an architecture for systematic models
- Experiments on standard benchmarks SCAN and COGS
- Wrap-up

What we've got here is a failure of systematicity...

- Similar inputs can yield different responses
 - They are very sensitive to rephrasings and reorderings.
- Dissimilar inputs can yield the same response
 - Two similar looking but different inputs mistakenly get the same response.
- No consistent compositional ability
 - The solution to task A which contains task B as a sub-task should be consistent with the solution of task B alone.



Example: Different responses for similar things

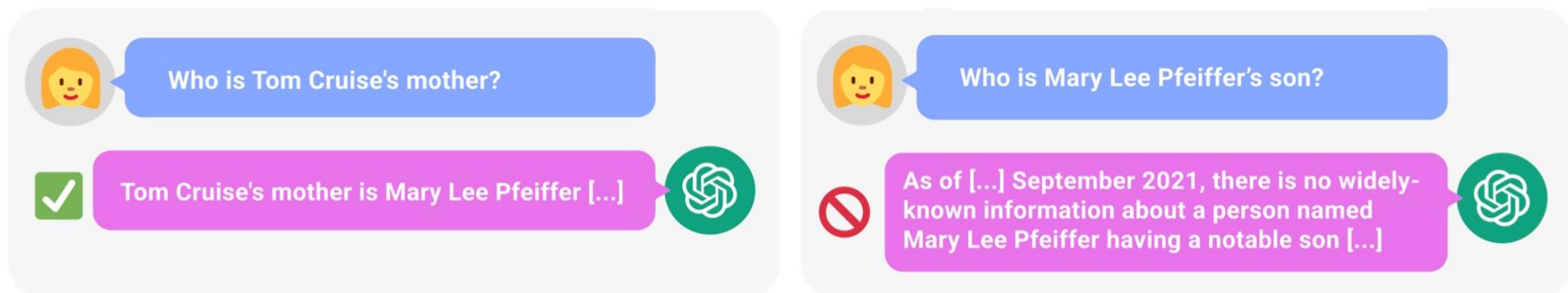


Image adapted from [Berglund et al. 2023]

Example: Same response for dissimilar things

The missing rungs on the ladder to general AI. [Chollet 2023]

The Monty Hall Problem



TK

You

Suppose I'm on the Price Is Right game show and Monty Hall is there and shows me 3 doors. He tells me that behind two doors there is nothing, but behind the third door is a shiny new motorcycle. He asks me to pick a door. I do and he then opens one of the remaining doors to reveal nothing there. He then asks me if I want to switch my door. Should I do it?



ChatGPT

Yes, you should switch your door. This scenario is a classic probability problem known as the Monty Hall problem.

When you initially choose one of the three doors, there's a $1/3$ chance that you've chosen the door with the motorcycle behind it, and a $2/3$ chance that the motorcycle is behind one of the other two doors.

When Monty opens one of the remaining doors to reveal nothing, he's essentially giving you extra information. By doing so, he doesn't change the initial probabilities. The door he opens will never reveal the motorcycle because he knows where it is and he won't open that door.

Now, when you're asked if you want to switch, there's a $2/3$ chance that the motorcycle is behind one of the doors you didn't initially choose. Therefore, switching increases your chances of winning to $2/3$, while sticking with your initial choice keeps your chances at $1/3$.

So, statistically speaking, you should switch your door to maximize your chance of winning the motorcycle.



The NOT Monty Hall Problem



TK

You

Suppose I'm on the Price Is Right game show and Monty Hall is there and shows me 3 doors. He tells me that behind two doors there is nothing, but behind the third door is a shiny new motorcycle. He asks me to pick a door. I do and he then opens it to reveal the motorcycle. He then asks me if I want to switch my door. Should I do it?



ChatGPT

Yes, you should switch your door. This scenario is a variation of the Monty Hall problem, a famous probability puzzle. When you initially picked your door, there was a $1/3$ chance that you chose the correct door containing the motorcycle and a $2/3$ chance that the motorcycle was behind one of the other two doors.

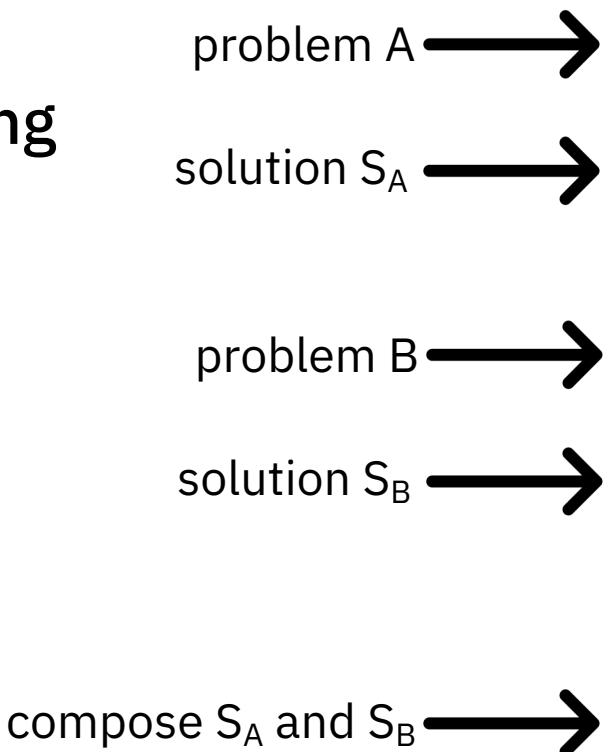
When Monty reveals one of the doors you didn't pick and shows that it doesn't contain the motorcycle, the probability distribution changes. The remaining unopened door still has a $2/3$ chance of hiding the motorcycle, while your initially chosen door retains its $1/3$ chance.

Therefore, switching doors increases your chances of winning the motorcycle from $1/3$ to $2/3$. It might seem counterintuitive at first, but statistically, it's advantageous to switch.



Example: Lack of Compositionality

- 1-5 digit multiplication
- Poor generalization
- Can't build up the answer compositionally – even with helpful prompting



Faith and Fate: Limits of Transformers on Compositionality [Dziri et al.]

Question: What is 35 times 90?

Scratchpad: Let's perform the multiplication step by step:

Let's multiply 35 by the digit in the ones place of 90, which is 0.

1. Multiply 0 by the digit in the ones place of 35, which is 5. This gives $5 \times 0 = 0$. Write down the result 0.
2. Multiply 0 by the digit in the tens place of 35, which is 3. This gives $3 \times 0 = 0$. Write down the result 0.
3. The partial product for this step is $A=0$ which is the concatenation of the digits we found in each step.

Now, let's multiply 35 by the digit in the tens place of 90, which is 9.

4. Multiply 9 by the digit in the ones place of 35, which is 5. This gives $5 \times 9 = 45$. Write down the result 5 and carry over the 4 to the next step.
5. Multiply 9 by the digit in the tens place of 35, which is 3. Add the carryover from the previous step to account for this. This gives $(3 \times 9) + 4 = 31$. Write down the result 31.
6. The partial product for this step is $B=315$ which is the concatenation of the digits we found in each step.

Now, let's sum the 2 partial products A and B, and take into account the position of each digit: $A=0$ (from multiplication by 0) and $B=315$ (from multiplication by 9 but shifted one place to the left, so it becomes 3150). The final answer is $0 \times 1 + 315 \times 10 = 0 + 3150 = 3150$.

Figure 9: A sample scratchpad for the multiplication task.

Systematic Models

1. Are low complexity algorithms

- low Kolmogorov complexity

2. Treat “similar” things “similarly” (and different things differently)

- “similar” is a similarity measure that determines important classes for the domain
- “similarly” means “using the same abstract program”

3. Learn and predict compositionally

- learn to solve new problems by decomposing them into parts with solutions that are known or easy to generate and compose.

Hypothesis: A systematic model should be able to learn with much less data.

Neural nets vs Systematic Models

Neural nets

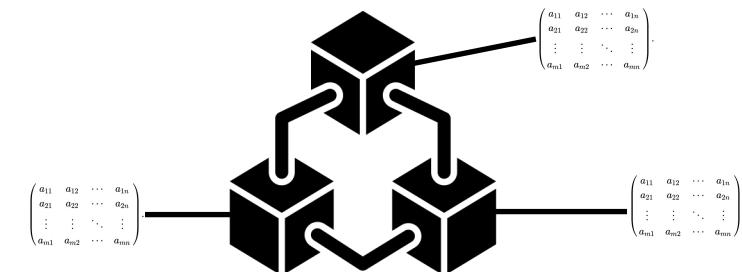
1. models are programs which are relatively simple structurally (i.e. number of loops, branches, compositions)
2. the structure of the computation is **static** and fixed for all inputs; to compensate for this lack of flexibility, the **tensors have to get very large as the complexity of the task increases.**
3. models are **not simple** in terms of algorithmic complexity because the number of weights is huge and usually incompressible.

Systematic models

1. models are programs which can be structurally complex.
2. the structure of the computation is **dynamic** and varies with the class of the input; because of this flexibility, the **tensors can remain small as the complexity of the task increases.**
3. models are **simple** in terms of algorithmic complexity because they use only a small number of very low dimensional tensors.

A diagram illustrating a systematic model. On the left, a small cube represents the input tensor. A curved arrow points from this input to a large, multi-dimensional output tensor, represented by a matrix with rows labeled $a_{11}, a_{12}, \dots, a_{1n}$, $a_{21}, a_{22}, \dots, a_{2n}$, and so on down to $a_{m1}, a_{m2}, \dots, a_{mn}$.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}.$$



Compositional Program Generation (CPG)

A **systematic** architecture for learning in compositional domains

Parses each input into an abstract class hierarchy.

Compositionally **generates** an input-specific abstract program from simple modules associated with each class

Evaluates that program on the input to make the prediction

Compositional Program Generation for Few-Shot Systematic Generalization

Tim Klinger^{1,*} and Luke Liu^{2,*} and Soham Dan,¹ and Maxwell Crouse¹ and Parikshit Ram¹ and Alex Gray^{1,3}

¹IBM Research AI

²New York University

³Centaur AI Institute

{tklinger, Soham.Dan, Maxwell.Crouse, parikshit.ram, alexander.gray}@ibm.com, ql2078@nyu.edu

Abstract

Compositional generalization is a key ability of humans that enables us to learn new concepts from only a handful of examples. Neural machine learning models, including the now ubiquitous Transformers, struggle to generalize in this way, and typically require thousands of examples of a concept during training in order to generalize meaningfully. This difference in ability between humans and artificial neural architectures motivates this study on a neuro-symbolic architecture called the Compositional Program Generator (CPG). CPG

of *systematicity* since the model seems unable to systematically compose its knowledge of how to perform individual actions like “jump” and its knowledge of how to “left twice”. Similarly, a model may fail to *productively* generalize to a recursive solution, working for some length inputs but not others. Moreover, one might expect that a model which is able to generalize systematically should be able to learn with less data (few-shot). If it is able to combine its knowledge of things it already knows to solve a new problem during training, then it shouldn’t need training data to learn how to solve that problem. Neural networks like Transformers have so far not been able to exploit any systematicity they have to learn with *less data*.

Compositional Program Generation for Few-Shot Systematic Generalization
<https://arxiv.org/abs/2309.16467>

Example: Instruction following in SCAN

English Command

walk left twice

jump right thrice

Robot Action

I_TURN_LEFT WALK I_TURN_LEFT WALK

I_TURN_RIGHT JUMP I_TURN_RIGHT JUMP I_TURN_RIGHT JUMP

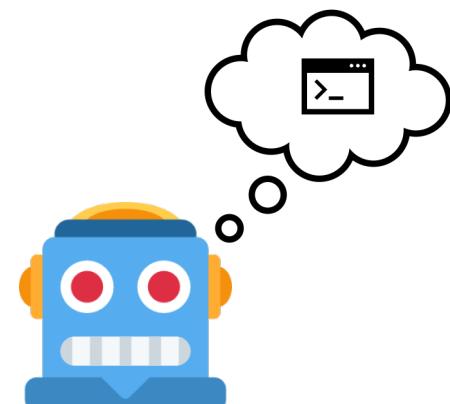
What about this composition?

jump left twice

I_TURN_LEFT JUMP I_TURN_LEFT JUMP

In this case the model needs to recognize that **jump** and **walk** are both **actions** and therefore it should handle **jump left twice** with the same program it uses on **walk left twice**.

The *answers* are different but the *program* for handling both is the same



Example: Instruction following in SCAN

```
translate_action(a : Action) {  
    return upper_case(a)
```

e.g. jump → JUMP

```
translate_direction(d : Direction) {  
    return "I_TURN_" + upper_case(d)
```

e.g. left → I_TURN_LEFT

```
translate_directional_action(da : DirectionalAction) {  
    a = translate_action(da.action)  
    d = translate_direction(da.direction)  
    # swap order of direction and action  
    return d, a
```

e.g. jump left → I_TURN_LEFT JUMP

```
translate_command(c : Command) {  
    da = c.directional_action  
    r = c.repetition  
    # repeat the result r times  
    return repeat(translate_directional_action(da), r)
```

e.g. jump left twice →
I_TURN_LEFT JUMP I_TURN_LEFT JUMP

Generalization To Longer Lengths

Systematic learning may require handling longer lengths of inputs than seen in training

English Command

walk left twice

jump right thrice

walk and jump

Robot Action

I_TURN_LEFT WALK I_TURN_LEFT WALK

I_TURN_RIGHT JUMP I_TURN_RIGHT JUMP I_TURN_RIGHT JUMP

WALK JUMP

What about this composition which is length 5?

jump left twice and walk I_TURN_LEFT JUMP I_TURN_LEFT JUMP WALK

For infinite (recursive) classes the input length may be unbounded

Compositional Program Generation (CPG)

A **systematic** architecture for learning in compositional domains



Figure 1: CPG architecture

Parsing – recognizes the structure of the input and the types of its parts using a CFG

Generation – builds a corresponding abstract function that depends on those types

Evaluation – evaluates the function on the input

Invariant: input expressions in the same class are evaluated using the same generated function; expressions in different classes may use different functions.

Compositional Program Generation for Few-Shot Systematic Generalization

Tim Klinger^{1,*} and Luke Liu^{2,*} and Soham Dan,¹ and Maxwell Crouse¹ and Parikshit Ram¹ and Alex Gray^{1,3}

¹IBM Research AI

²New York University

³Centaur AI Institute

{tklinger, Soham.Dan, Maxwell.Crouse, parikshit.ram, alexander.gray}@ibm.com, ql2078@nyu.edu

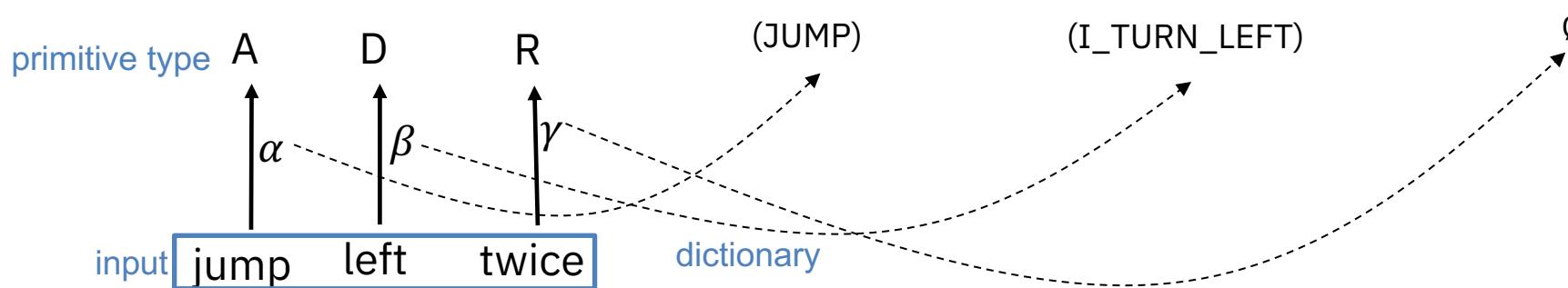
Abstract

Compositional generalization is a key ability of humans that enables us to learn new concepts from only a handful of examples. Neural machine learning models, including the now ubiquitous Transformers, struggle to generalize in this way, and typically require thousands of examples of a concept to learn it in a systematic and productive way. This difference in ability between humans and artificial neural architectures motivates this study on a neuro-symbolic architecture called the Compositional Program Generator (CPG). CPG

of *systematicity* since the model seems unable to systematically compose its knowledge of how to perform individual actions like “jump” and its knowledge of how to “left twice”. Similarly, a model may fail to *productively* generalize to a recursive solution, working for some length inputs but not others. Moreover, one might expect that a model which is able to generalize systematically should be able to learn with less data (few-shot). If it is able to combine its knowledge of things it already knows to solve a new problem during training, then it shouldn’t need training data to learn how to solve that problem. Neural networks like Transformers have so far not been able to exploit any systematicity they have to learn with *less data*.

Compositional Program Generation for Few-Shot Systematic Generalization
<https://arxiv.org/abs/2309.16467>

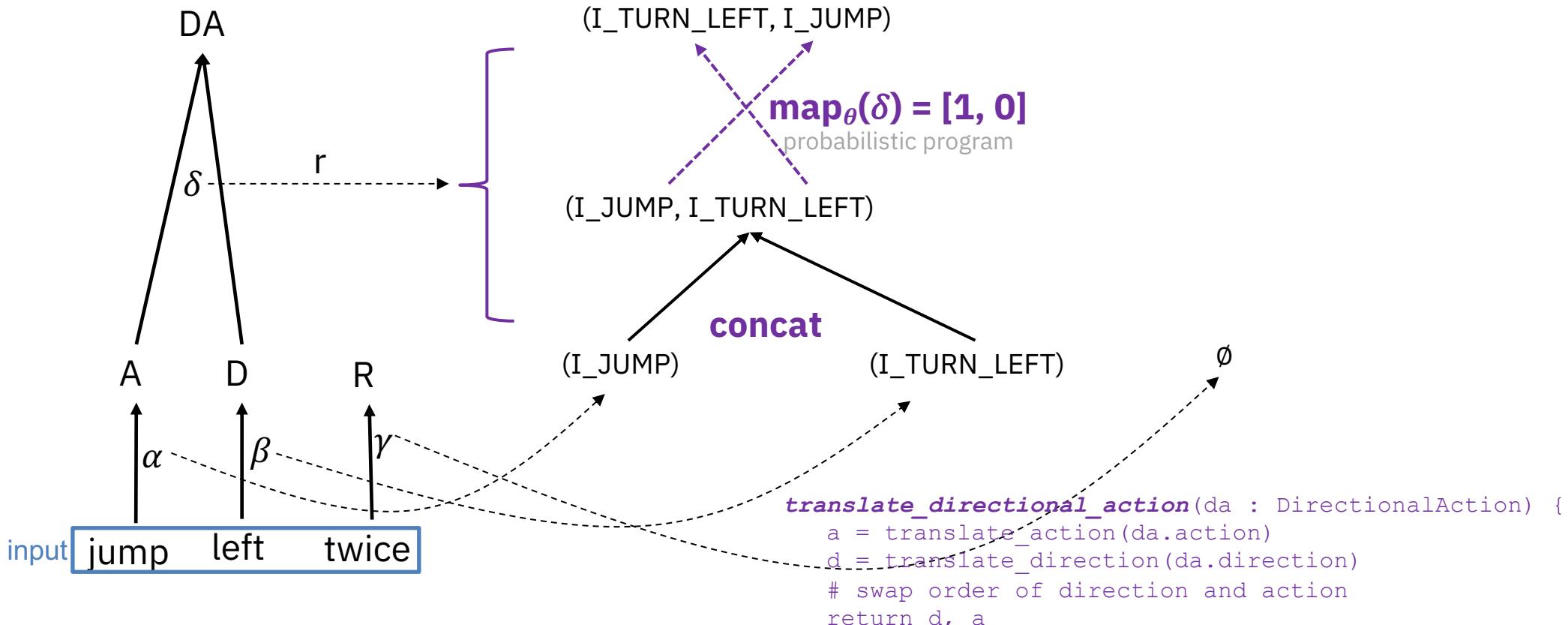
Example: SCAN inference Step 1



$\text{dictionary}_\theta(\alpha)$ implements `translate_action`
 $\text{dictionary}_\theta(\beta)$ implements `translate_direction`

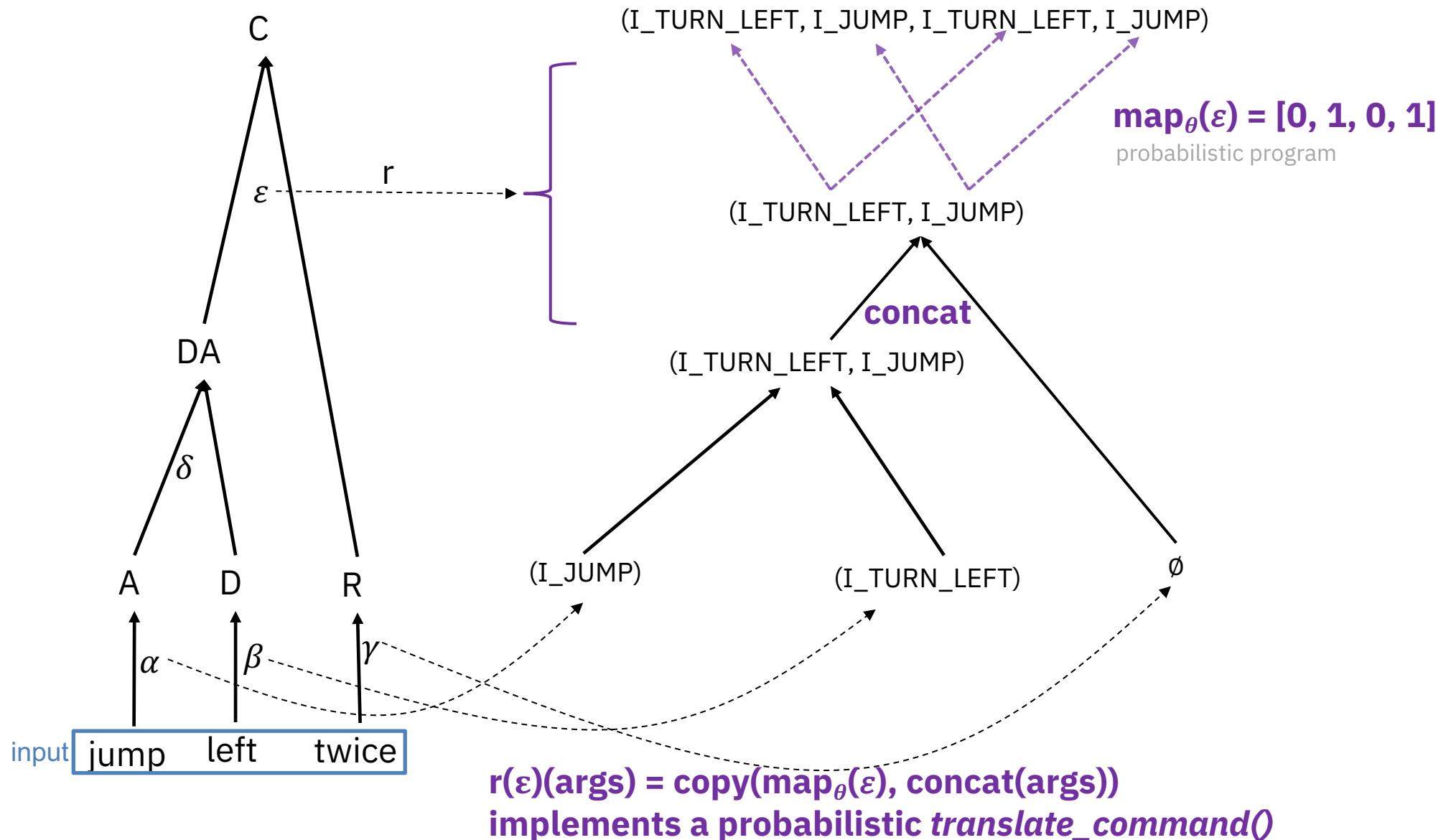
```
translate_action(a : Action) {  
    return upper_case(a)  
  
translate_direction(d : Direction) {  
    return "I_TURN_" + upper_case(d)
```

Example: SCAN inference Step 2

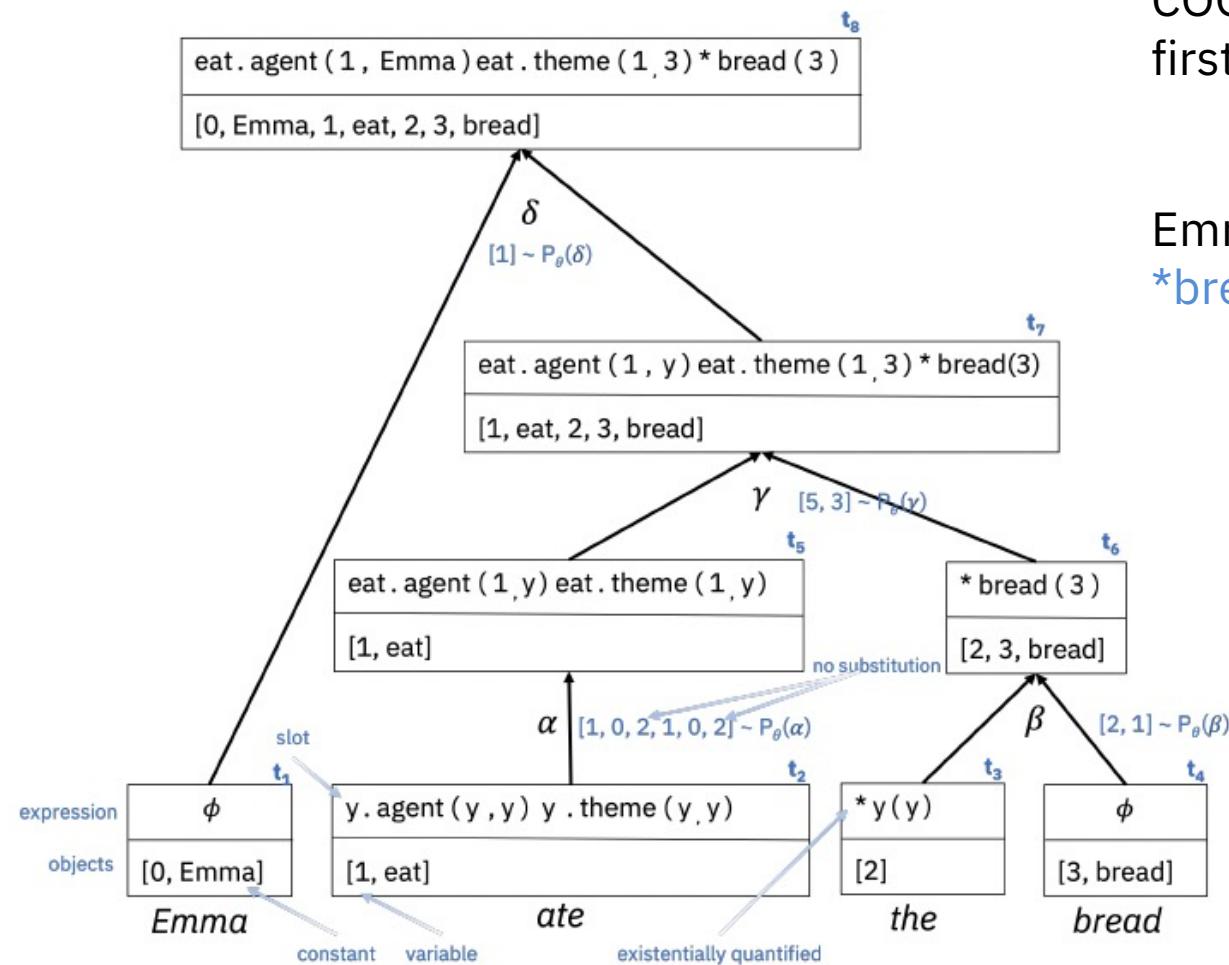


$r(\delta)(\text{args}) = \text{copy}(\text{map}_{\theta}(\delta), \text{concat}(\text{args}))$
implements a probabilistic version of `translate_directional_action()`

Example: SCAN inference Step 3



Example: COGS semantic parsing



COGS requires semantic parsing (translating English to a first-order logical form which captures its meaning)

Emma ate the bread:

*bread(3); eat.agent(1,Emma) AND eat.theme(1,3)

Datasets: SCAN and COGS

| Training set | Number of instances |
|------------------------------|---------------------|
| SCAN length split | 16,990 |
| SCAN length few shot split | 14 |
| SCAN add-jump split | 14,670 |
| SCAN add-jump few shot split | 14 |
| COGS split | 24,155 |
| COGS few shot | 22 |

1000x
reduction

To create the few-shot data:

loop through the dataset in length order and throw away any instances which don't introduce new types

SCAN and COGS: extreme few shot datasets

IN: walk OUT: I_WALK
IN: jump OUT: I_JUMP
IN: look OUT: I_LOOK
IN: run OUT: I_RUN
IN: turn right OUT: I_TURN_RIGHT
IN: walk right OUT: I_TURN_RIGHT I_WALK
IN: jump twice OUT: I_JUMP I_JUMP
IN: run thrice OUT: I_RUN I_RUN I_RUN
IN: walk after run OUT: I_RUN I_WALK
IN: turn opposite left OUT: I_TURN_LEFT I_TURN_LEFT
IN: run and walk OUT: I_RUN I_WALK
IN: walk opposite left OUT: I_TURN_LEFT I_TURN_LEFT I_WALK
IN: turn around right OUT: I_TURN_RIGHT I_TURN_RIGHT
I_TURN_RIGHT I_TURN_RIGHT
IN: run around left OUT: I_TURN_LEFT I_RUN I_TURN_LEFT I_RUN
I_TURN_LEFT I_RUN I_TURN_LEFT I_RUN

SCAN extreme few-shot instances

1. Oliver crumpled .
crumple . theme (x _ 1 ,
Oliver)
2. James investigated .
investigate . agent (x _ 1 ,
James)
3. Hazel cried .
cry . agent (x _ 1 , Hazel)
4. The captain ate .
* captain (x _ 1) ; eat .
agent (x _ 2 , x _ 1)
5. Emma floated Alexander .
float . agent (x _ 1 , Emma)
AND
float . theme (x _ 1 ,
Alexander)
6. Noah helped Levi .
help . agent (x _ 1 , Noah)
AND
help . theme (x _ 1 , Levi)
7. William drew Emma .
draw . agent (x _ 1 , William)
AND draw . theme (x _ 1 , Emma)
)
8. The donut was studied .
* donut (x _ 1) ; study .
theme (x _ 3 , x _ 1)
9. David expected to cook .
expect . agent (x _ 1 , David)
AND expect . xcomp (x _ 1 , x
_ 3)
AND cook . agent (x _ 3 ,
David)
10. The pen was helped .
* pen (x _ 1) ; help . theme
(x _ 3 , x _ 1)
11. A box was inflated .
box (x _ 1) AND inflate .
theme (x _ 3 , x _ 1)
12. Amelia gave Emma a strawberry .
give . agent (x _ 1 , Amelia) AND
give . recipient (x _ 1 , Emma) AND
give . theme (x _ 1 , x _ 4) AND strawberry (x _ 4)
13. Olivia was slipped a pencil .
slip . recipient (x _ 2 , Olivia) AND
slip . theme (x _ 2 , x _ 4) AND pencil (x _ 4)
14. Liam liked that Lucas cleaned .
like . agent (x _ 1 , Liam) AND
like . ccomp (x _ 1 , x _ 4) AND
clean . agent (x _ 4 , Lucas)
15. The cookie was passed to Emma .
* cookie (x _ 1) ; pass . theme (x _ 3 , x _ 1) AND
pass . recipient (x _ 3 , Emma)
16. A cake was stabbed by Scarlett .
cake (x _ 1) AND stab . theme (x _ 3 , x _ 1) AND
stab . agent (x _ 3 , Scarlett)
17. The cake was eaten by Olivia .
cake (x _ 1) ; eat . theme (x _ 3 , x _ 1) AND
eat . agent (x _ 3 , Olivia)
18. A bottle was rolled by Olivia .
bottle (x _ 1) AND roll . theme (x _ 3 , x _ 1) AND
roll . agent (x _ 3 , Olivia)
19. Natalie mailed the cake to Emma .
* cake (x _ 3) ; mail . agent (x _ 1 , Natalie) AND
mail . theme (x _ 1 , x _ 3) AND
mail . recipient (x _ 1 , Emma)
20. Emma ate the ring beside a bed .
* ring (x _ 3) ; eat . agent (x _ 1 , Emma) AND
eat . theme (x _ 1 , x _ 3) AND
ring . nmod . beside (x _ 3 , x _ 6) AND
bed (x _ 6)
21. Emma was posted the cake by Noah .
* cake (x _ 4) ; post . recipient (x _ 2 , Emma) AND
post . theme (x _ 2 , x _ 4) AND
post . agent (x _ 2 , Noah)
22. A cake was forwarded to Levi by Charlotte .
cake (x _ 1) AND
forward . theme (x _ 3 , x _ 1) AND
forward . recipient (x _ 3 , Levi) AND
forward . agent (x _ 3 , Charlotte)

COGS extreme few-shot instances

Model and Training

- The model is **end-to-end differentiable** and uses standard SGD training
- Training is **curricular** by length of the input
- The parameters it learns are for the dictionary (base function) and each module.
- All are learned with **small 1-2 layer FF networks with a (hard) Gumbel Softmax estimation.**
- **No embeddings** are used anywhere
- **Training time is comparable to neural networks** because it is not parallelized to use GPUs and just uses batch size 1
- More details in the paper and code

Open source code: <https://github.com/IBM/cpg>

Experiments

Compared against:

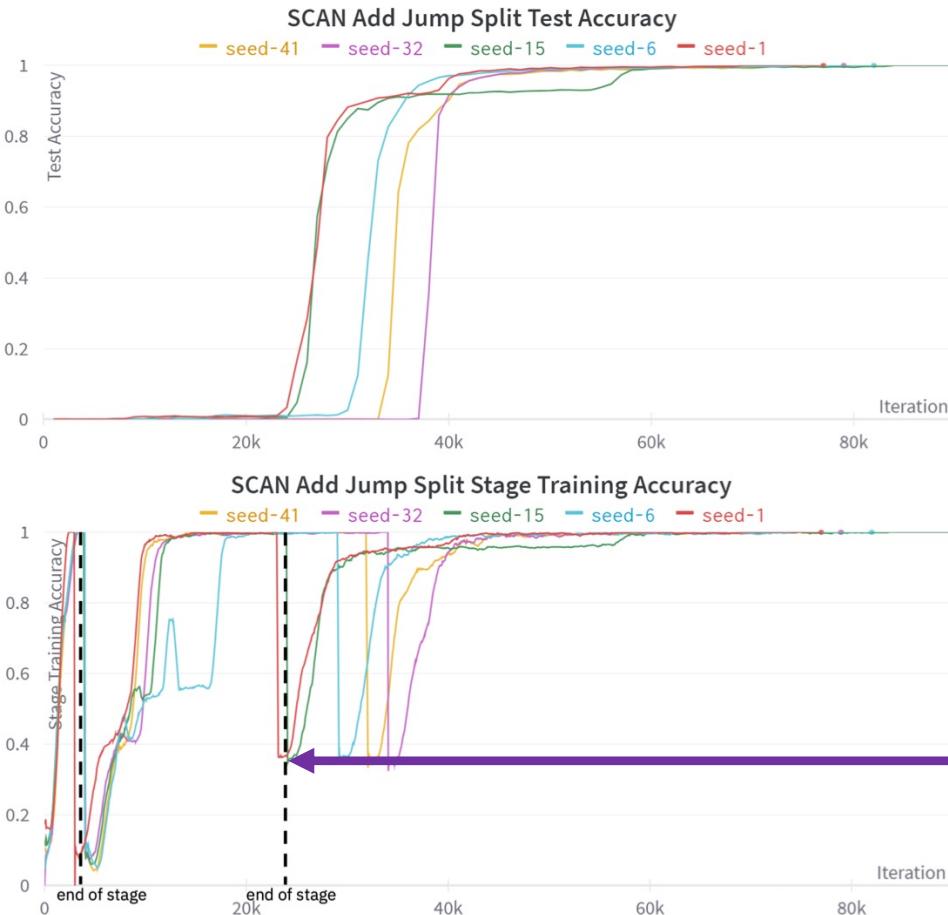
- Tuned Transformers
- Neuro-symbolic techniques (LANE, LEAR, NPS)
- Least-to-most prompted LLMs

Summary:

- **CPG achieves SOTA perfect few-shot generalization**
- Transformers use a lot of data and do quite poorly
- NS approaches perform much better than Transformers
- CPG outperforms a prompted LLM approach on accuracy and data efficiency (5x)

| Dataset | Split | Model | * | Accuracy (%) | Std. Dev (+/-) | Num Samples |
|---------|----------------|---|----------|--------------|----------------|-------------|
| SCAN | Length | Transformer (rp, u) [Csordás et al] | | 20 | | All (17k) |
| | | LANE [Liu et al] | 1 | 100 | | All (17k) |
| | | NPS [Nye et al] | 2 | 100 | | 149 +/- 28 |
| | | LLM+LTMP [Zhou et al] | 4 | 100 | | 22 |
| | | CPG | 3 | 100 | | 14 |
| | Add Jump | Transformer (small-4s) [Ontañón et al] | | 3.9 | | All (15k) |
| | | LANE [Liu et al] | 1 | 100 | | All (15k) |
| | | NPS [Nye et al] | 2 | 100 | | 209 +/- 91 |
| | | LLM+LTMP [Zhou et al] | 4 | 97.7 | | 22 |
| | | CPG | 3 | 100 | | 14 |
| COGS | Generalization | Transformer(rp, u) | 1 | 77 | .01 | All (24k) |
| | | LANE [Liu et al] | 1 | - | - | - |
| | | NPS [Nye et al] | 2 | - | - | - |
| | | LLM+LTMP [Drozdov et al] | 4, 6 | 99.2 | | 100 |
| | | LEAR [Liu et al] | 6 | 97.7 | .7 | All (24k) |
| | | CPG | 3 | 100 | | 22 |

Incremental training without forgetting



maps for types learned in earlier iterations are frozen after training and re-used in later stages
note that accuracy does not go to 0 even though the model is seeing new types for the first time

Figure 6: SCAN add-jump split. Top: test curve, Bottom: train curve. Results shown for 5 seeds. Vertical dash line indicates the end of the 1st and 2nd curriculum stages for seed 1.

Limitations and Next Steps

- Requires a context-free grammar for the input language
 - working on learning the grammar and parser now
 - LLMs could be helpful to simplify and de-noise idiomatic sentences
- May need different simple modules for different classes of tasks
 - using more general modules can help but at a cost of increased learning complexity
- Does not take advantage of GPUs and scaling
 - on the other hand it doesn't need much data to learn...

Conclusion

- LLMs are not systematic in several ways:
 - they don't recognize and respect important abstractions
 - similar things are treated differently
 - different things are treated similarly
 - composition is patchy and poor
- Systematic models can do much better and require several orders of magnitude less data
- CPG is a systematic model that solves SCAN and COGS benchmarks
- Next steps: learn the underlying grammar of the input language concurrently with the task

THANK YOU!

BACKUP

Dictionaries

| | |
|------------------------|---------------|
| run | :I_RUN |
| look | :I_LOOK |
| walk | :I_WALK |
| jump | :I_JUMP |
| left | :I_TURN_LEFT |
| right | :I_TURN_RIGHT |
| <i>everything else</i> | : ϕ |

Figure 4: The SCAN dictionary

Non-verb expression mappings:

| | |
|--------|---------------------|
| the | :*y(y) |
| a | :y(y) |
| on | :y.nmod.on(y,y) |
| in | :y.nmod.in(y,y) |
| beside | :y.nmod.beside(y,y) |

Verb expression mappings

| | |
|-----------------------------|---|
| v_trans_omissible_p1 | :y.agent(y,y) |
| v_trans_omissible_p2 | :y.agent(y,y)y.theme(y,y) |
| v_trans_omissible_pp_p1 | :y.theme(y,y) |
| v_trans_omissible_pp_p2 | :y.theme(y,y)y.agent(y,y) |
| v_trans_not_omissible | :y.agent(y,y)y.theme(y,y) |
| v_trans_not_omissible_pp_p1 | :y.theme(y,y) |
| v_trans_not_omissible_pp_p2 | :y.theme(y,y)y.agent(y,y) |
| v_cp_taking | :y.agent(y,y)y.ccomp(y,y) |
| v_inf_taking | :y.agent(y,y)y.xcomp(y,y) |
| v_unacc_p1 | :y.agent(y,y)y.theme(y,y) |
| v_unacc_p2 | :y.theme(y,y) |
| v_unacc_pp_p1 | :y.theme(y,y) |
| v_unacc_pp_p2 | :y.theme(y,y)y.agent(y,y) |
| v_cp_taking | :y.agent(y,y) |
| v_inf_taking | :y.agent(y,y) |
| v_unacc_p1 | :y.agent(y,y)y.theme(y,y)y.recipient(y,y) |
| v_unacc_p2 | :y.agent(y,y)y.recipient(y,y)y.theme(y,y) |
| v_unacc_pp_p1 | :y.theme(y,y)y.recipient(y,y) |
| v_unacc_pp_p2 | :y.theme(y,y)y.recipient(y,y)y.agent(y,y) |
| v_unerg | :y.recipient(y,y)y.theme(y,y) |
| v_inf | :y.recipient(y,y)y.theme(y,y)y.agent(y,y) |

Figure 5: The COGS dictionary

Compositional functions

In normal usage it means that a function is composed of other functions: $f = g \circ h$

Here it refers to the the *relational* notion that the meaning of the whole should be a function of the meanings of its parts

This presupposes a domain in which it makes sense to talk about wholes and parts (like language)

Pagin defines wholes and parts in terms of abstract grammar rules α

$$\mu(\alpha(u_1, u_2, \dots, u_n)) = r(\alpha)(\mu(u_1), \mu(u_2), \dots, \mu(u_n))$$

The diagram illustrates the formula $\mu(\alpha(u_1, u_2, \dots, u_n)) = r(\alpha)(\mu(u_1), \mu(u_2), \dots, \mu(u_n))$. It uses blue curly braces to group parts of the expression. The first brace groups $\mu(u_1), \mu(u_2), \dots, \mu(u_n)$ and is labeled "the meaning of the parts". The second brace groups $\mu(u_1), \mu(u_2), \dots, \mu(u_n)$ and is labeled "is a function". The third brace groups (u_1, u_2, \dots, u_n) and is labeled "of the whole". The fourth brace groups $\alpha(u_1, u_2, \dots, u_n)$ and is labeled "the meaning".

Pagin, P.; and Westerståhl, D. 2010. Compositionality I: Definitions and Variants. *Philosophy Compass*, 5(3): 250–264.