Measuring Systematic Generalization in Neural Proof Generation with Transformers

Nicolas Gontier
Quebec Artifical Intelligence Institute (Mila),
Polytechnique Montréal
gontiern@mila.quebec

Koustuv Sinha
Quebec Artifical Intelligence Institute (Mila),
McGill University
Facebook AI Research

Siva Reddy
Quebec Artifical Intelligence Institute (Mila),
McGill University
Facebook CIFAR AI Chair

Christopher Pal
Quebec Artifical Intelligence Institute (Mila),
Polytechnique Montréal
ElementAI
Canada CIFAR AI Chair

Abstract

We are interested in understanding how well Transformer language models (TLMs) can perform reasoning tasks when trained on knowledge encoded in the form of natural language. We investigate systematic generalization abilities on an inductive logical reasoning task in natural language, which involves reasoning over relationships between entities grounded in first-order logical proofs. Specifically, we perform soft theorem-proving by leveraging TLMs to generate logical proofs represented in natural language. We systematically test proof generation capabilities, along with inference capabilities leveraging the generated proofs. We observe length-generalization issues in proof generation and inference when evaluated on longer-than-trained sequences. However, we observe TLMs improve their generalization performance after being exposed to longer, exhaustive proofs. In addition, we discover that TLMs are able to generalize better using backward-chaining proofs compared to their forward-chaining counterparts, while they find it easier to generate forward chaining proofs. We observe that models that are not trained to generate proofs are better at generalizing to problems based on longer proofs. This result suggests that Transformers have efficient, yet not interpretable reasoning strategies internally. These results also highlight the systematic generalization issues in TLMs in the context of logical reasoning, and we believe this work will motivate deeper inspection of their underlying reasoning strategies.

1 Introduction

Systematic Generalization is the capacity to understand and produce a potentially infinite number of novel combinations from known components (Chomsky, 1957; Montague, 1970). For example, in Figure 1, a model could be exposed to a set of facts (e.g., "Nat is the granddaughter of Betty", "Greg is the brother of Nat", "Flo is the sister of Greg"), but not to all the possible facts that can be inferred by combination of the known components (e.g., "Flo is the granddaughter of Betty"). If a model is able to perfectly accomplish a task by leveraging existing facts to infer new ones, we deem the model is generalizing systematically.

Recent development in natural language processing (NLP) showed that Transformer (Vaswani et al., 2017) language models are able to capture linguistic knowledge (Peters et al., 2018; Goldberg, 2019; Tenney et al., 2019), and yield state-of-the-art performances in many NLP tasks (Radford et al., 2018;

To appear at NeurIPS 2020, submitted for peer review on June 5th 2020.

Devlin et al., 2019; Dai et al., 2019; Yang et al., 2019), including but not limited to answering reading comprehension questions (Radford et al., 2019; Brown et al., 2020) and generating factual knowledge (Petroni et al., 2019) with little to no task supervision. These models are optimized on large corpora to predict the next word in a sentence or to predict masked words in a sentence. While yielding impressive results, it is not clear if Transformer models rely on many superficial patterns in the data or if they actually learn re-usable skills, enabling them to generalize to new tasks by leveraging compositionality of those skills (Lake and Baroni, 2018; Liška et al., 2018). Training on massive data can give certain advantages with respect to understanding the meanings of words, but we conjecture that such data gives models much less experience with reasoning over long inference chains.

In our work, we study the less understood issues related to how well Transformer Language Models (TLMs) are able to perform long chains of reasoning. In particular, we study TLMs in the task of theorem proving, where facts and proofs are specified in natural language. Theorem provers are effective and interpretable solutions for systematically composing known facts into novel ones (De Raedt et al., 2007; Rocktäschel and Riedel, 2017). Using thereom proving, we test if TLMs can generate interpretable proofs with language modeling as their main objective. In particular, we study their *behavior* as logical reasoners on text through analyzing the proofs generated in natural language and the final answer. This setup allows us to evaluate the reasoning and generalization capabilities of TLMs. Recent work such as Petroni et al. (2019); Raffel et al. (2020); Brown et al. (2020) suggest that language models can be treated as knowledge bases. This directly motivates us to investigate if language models can also learn certain reasoning strategies. Studying these abilities would enable future research in using these models as dynamic knowledge bases that could infer new knowledge even when it is not "stored" directly (i.e. seen during pre-training).

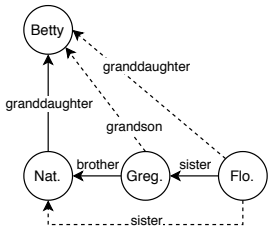


Figure 1: Example of a CLUTRR graph with known facts (solid lines) and unknown facts to infer (dotted lines).

For natural language theorem proving, we use the question answering CLUTRR benchmark suite (Sinha et al., 2019) to perform controlled studies. This dataset is of interest because (i) of the compositional nature of tasks involved, making it ideal to evaluate systematic generalization and (ii) each question–answer pair is accompanied by a proof that can be used to explain how to arrive at the answer. We use the dataset as a medium to design targeted experiments to understand the reasoning capacity of TLMs.

Our experiments reveal the following:

1. TLMs suffer from length generalization, i.e., they cannot *extrapolate* to proofs which require more proof steps than seen during training time.
2. They generalize better when trained to generate long proofs compared to short proofs.
3. They generalize better when trained to generate backward-chaining proofs rather than forward-chaining.
4. Surprisingly, they generalize better when they are trained to directly generate the answer instead of learning to generate the proof and then the answer.

To the best of our knowledge, we are the first to use a language modeling objective to do interpretable theorem proving with a Transformer. We hope that this work can shed some light on the reasoning capacity of Transformers and inspire future research directions for the community to design models with greater reasoning capacity.

2 Related Work

Systematic generalization has recently been in spotlight due to its importance in understanding the strengths and weaknesses of neural networks. Bahdanau et al. (2019a,b) identify and evaluate the generalization capacity of visual question answering models. We however focus this study on a fully natural language domain. There have been several recent studies on explicitly evaluating systematic generalization capabilities of natural language understanding and generation. Dasgupta et al. (2019) introduce a natural language inference (NLI) dataset which proves to be challenging for language understanding models for compositional generalization. Goodwin et al. (2020) also evaluate systematic generalization in NLI setting with procedurally generated controlled test cases to observe the failures of neural architectures.

	raw	facts	amt
story	[(Natasha, granddaughter, Betty), (Florence, sister, Gregorio), (Gregorio, brother, Natasha)]	<STORY> Natasha is a granddaughter to Betty. Florence is Gregorio 's sister. Gregorio is a brother of Natasha.	<STORY> Betty likes picking berries with her son 's daughter. Her name is Natasha. Gregorio took his sister, Florence, to a baseball game. Gregorio and his sister Natasha love it when their grandmother visits because she spoils them. She is coming this week to watch them while their parents are out of town.
query	(Florence, _, Betty)	<QUERY> Who is Florence for Betty ?	
proof	[{(Florence, granddaughter, Betty): [(Florence, sister, Gregorio), (Gregorio, grandson, Betty)]}, {(Gregorio, grandson, Betty): [(Gregorio, brother, Natasha), (Natasha, granddaughter, Betty)]}]	<PROOF> since Florence is a sister of Gregorio, and Gregorio is a grandson to Betty, then Florence is a granddaughter to Betty. since Gregorio is a brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is a grandson of Betty.	
answer	granddaughter	<ANSWER> Florence is the granddaughter of Betty	

Table 1: CLUTRR example of level 3 (ie: 4 entities, 3 relations, 2 proof steps). The proof follows the `short-proof-rev` strategy. We refer the reader to Figure 1 to visualize the corresponding graph in which solid lines refer to the facts given in the story and dotted lines refer to the new facts inferred in each proof step.

Our work studies the systematic generation and reasoning capabilities on Transformer-based (Vaswani et al., 2017) language model using CLUTRR (Sinha et al., 2019), a dataset for logical question answering which provides access to the underlying logical proofs. Similar datasets include SCAN (Lake and Baroni, 2018) which has been instrumental to test systematic generalization (Lake, 2019; Baroni, 2020) and CFQ (Keysers et al., 2020) which measures systematicity of language understanding via a question answering setup. Sinha et al. (2019) propose a series of baseline models with the CLUTRR dataset but none of them took advantage of the provided proof attached with each example. In addition, their Transformer baselines were not fine-tuned on the task. Unlike them, we focus on learning and generating proofs for studying systematic generalization.

Neural proof generation (Sekiyama et al., 2017) and neural theorem proving (Rocktäschel and Riedel, 2017; Weber et al., 2019; Minervini et al., 2020) have been explored in previous work. They tend to combine symbolic and statistical approaches to leverage the compositionality of symbolic systems and the flexibility of statistical systems. Nevertheless, these combined systems all assume some predefined set of atoms and rules making up the environment. We instead use natural language text to define our environment and measure the limits of a purely statistical approach.

Clark et al. (2020) is the most relevant work to ours. However their system is not generative, rather they predict a true/false binary label on candidate answers. We instead explore the generalization capacity of Transformer decoders and use the generated proof as an interpretable explanation for the final answer.

3 Evaluating systematic generalization through interpretable reasoning

3.1 The task

Background. We use the family relation CLUTRR benchmark suite (Sinha et al., 2019) to generate our dataset. Each example is composed of (i) a family graph $G = (V, E)$ (referred as *story*) with entities as nodes ($v \in V$) and relationships as edges ($e \in E$), (ii) a *query* about the relationship between two entities separated by more than one hop in the family graph, or (u, v) (iii) a reasoning path (referred as *proof*) expressed as a list of $(node, edge, node)$ tuples, referred to as *facts* and (iv) the target relationship between the two queried entities (referred to as the *answer*). The dataset contains 272 distinct entities and 20 relationship types, ordering to $\sim$ 1.5M possible facts. Each $(node, edge, node)$ fact can be expressed in natural language using either one of 5 factual sentences (referred to as `facts` template), or by using one of $6,000$ noisy but more natural sentences written by mechanical turkers (refered as `amt` template). Family graphs are expressed using either the `facts` template or the `amt` template, while queries, proofs and answers are always expressed with the `facts` template. A CLUTRR example can be seen in Table 1 and Figure 1.

Terminology. In order to evaluate systematic generalization, we define the following building blocks that constitute a *proof*:

- `entity`: one node. *e.g. "Anna"*.
- `relation`: one edge. *e.g. "mother"*.

sp	since Gregorio is a brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is a grandson of Betty. since Florence is a sister of Gregorio, and Gregorio is a grandson to Betty, then Florence is a granddaughter to Betty.
spr	since Florence is a sister of Gregorio, and Gregorio is a grandson to Betty, then Florence is a granddaughter to Betty. since Gregorio is a brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is a grandson of Betty.
lp	since Gregorio is the brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is the grandson of Betty. since Florence is the sister of Gregorio, and Gregorio is the brother of Natasha, then Florence is the sister of Natasha. since Florence is the sister of Natasha, and Natasha is the granddaughter of Betty, then Florence is the granddaughter of Betty.
lpr	since Florence is the sister of Natasha, and Natasha is the granddaughter of Betty, then Florence is the granddaughter of Betty. since Florence is the sister of Gregorio, and Gregorio is the brother of Natasha, then Florence is the sister of Natasha. since Gregorio is the brother of Natasha, and Natasha is the granddaughter of Betty, then Gregorio is the grandson of Betty.

Table 2: Proof resolution types for an example of level 3. We refer the reader to Figure 1 for the kinship graph corresponding to this example. **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed.

- `fact`: Single factual sentence representing a $(node, edge, node)$ tuple using `facts` template. *e.g. "Anna is the mother of Bob."*
- `proof_step`: Single inference step combining two `facts` to get a new one. *e.g. "since (A, mother, B) and (B, brother, C) then (A, mother, C)."*
- `proof`: The entire *resolution chain*, consisting of multiple `proof_steps`.

Following the setup of CLUTRR, we define the relative *difficulty* of individual tasks according to the number of edges present in the family graph. For instance, Figure 1 shows a level-3 example because it has 3 solid edges (known facts) between 4 entities. In general, a **level** k **task consists of** k **edges** (corresponding to k sentences in the story) **between** $k+1$ **nodes and** $k-1$ **hidden edges to infer** (corresponding to $k-1$ proof steps to solve the task). As the levels increase, so does the number of sentences in the story and the number of proof steps in the proof.

Problem Setup. We trigger a model to: (1) given a story and query, generate a proof followed by an answer, and (2) given a story, query, and a proof, generate an answer. In particular, we train a Transformer-based decoder (Liu et al., 2018) with the language modeling objective on entire sequences of "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER> [answer]":

$$L(\theta) = \sum_i \log P(w_i | w_1, \ldots, w_{i-1}; \theta)$$

This setup enables us to generate both the answer to a query and the proof to arrive at this answer, given as input the family graph, expressed as a story and a question. Concretely, we inject sequences of the story and query having delimiters "<STORY>" and "<QUERY>" to the language model and trigger it to generate the corresponding proof and answer with tokens "<PROOF>" and "<ANSWER>" respectively.

3.2 Proof resolution strategies

In our task, we turn language models into approximate proof generators. Specifically, we train TLMs to generate `proofs` (as defined in Section 3.1). We do not explicitly perform inference on the generated `proofs`, but reformulate the language generation objective to generate the inferred answer after the `proof` sequence. Our task allows us to leverage TLMs to generate *forward* and *backward* chaining resolution paths used in Inductive Logic Programming (ILP) (Evans and Grefenstette, 2018). In our case, these resolution paths are expressed in natural language. To simulate approximate

ANON TEST	lvl.**2**	lvl.**3**	lvl.**4**	lvl.**5**	lvl.**6**	lvl.**7**	lvl.**8**	lvl.**9**	lvl.**10**
proofs (*many proof steps*)	16.28%	0%	0%	0%	0%	0%	0%	0%	0%
proof steps ("*since A-r1-B and B-r2-C then A-r3-C*")	73.08%	58.06%	52.75%	54.28%	50.93%	59.04%	56.92%	53.55%	52.17%
facts (*A-r-B*)	100%	100%	100%	100%	100%	100%	100%	100%	100%
entities (*A*)	100%	100%	100%	100%	100%	100%	100%	100%	100%
relations (*r*)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 3: Percentage of the test proof's building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value of 100% to better visualize which building blocks were entirely contained in the training set.

theorem generation, we introduce four different types of `proof` that can be used to derive the answer given a story and query. An example of each of these can be seen in Table 2 and we describe each setting below:

short-proof-rev (spr). In this setting, we use the backward-chaining resolution path provided by the CLUTRR dataset, which is generated by recursive application of the kinship logical rules. This proof strategy can be viewed as an *explain-why* scenario, where the first sentence in the proof contains the answer (target relationship) and the subsequent sentences contain the intermediate steps required to explain that answer. We refer the reader to Sinha et al. (2019) for further details on the generation of this proof setting.

short-proof (sp). In this setup, we reverse the resolution chain provided by the CLUTRR dataset by swapping all sentences from the `short-proof-rev` setting. Doing so, we arrive at a forward-chaining inference path, in which the final proof step consists of the target relationship. Specifically, the first sentence in the proof combines two facts from the given story to infer a new fact. In subsequent proof steps, the inferred fact from the previous step is combined with a fact from the story, to infer a new fact until the answer is found.

long-proof (lp). Forward-chaining inference technique in ILP consists of generating all possible new facts from the starting facts, and evaluate each of them for the resolution of the target answer (Russell and Norvig, 2020). In this setup, we extend the `short-proof` setup where we attempt to infer all possible facts given the ones present in the input story. Each proof step combines any two previously known facts to infer a new fact until the answer is found. Pseudo-code for generating this proof can be found in Appendix 6.2.

long-proof-rev (lpr). This setting is the same as the previous one, but in reverse. It starts from the answer and goes back to the facts originally given in the story. This resolution strategy can be viewed as a backward-chaining strategy where all possible paths are considered. This proof strategy is obtained by swapping all sentences from the long-proof setting.

We compare each strategy in our experiments to understand which form of logical resolution is easier to learn for TLMs. In particular, we note that the reversed proof strategies (`spr` and `lpr`) fall in the backward-chaining family of logical resolution, while the non-reversed strategies (`sp` and `lp`) represent the forward-chaining resolutions. Backward-chaining family features the proof step containing the answer at the beginning of the proof. On the other hand, forward-chaining type proofs (`sp` and `lp`) feature the proof step containing the answer at the end of the proof.

3.3 Systematic generalization in proof generation

Now that we have defined the task and various proof generation strategies available in our setup, we proceed to define the aspects of generalization we aim to test. Our original CLUTRR formulation tests the generalization capacity of a model to new `facts` hence new `proof_steps` and new `proofs`, after being trained on all `entities` and `relations`. Initial experiments on this setup showed that Transformer language models fail to generalize to unseen `facts`. Indeed, due to the presence of a large number of entities in CLUTRR, we end up with combinatorially large number of possible facts. The model may thus not be able to learn how to represent each entity effectively, hence reducing its chances to learn higher-order structures such as unseen `facts`. Experimental results on this original setting are provided in Appendix 6.1.

We instead slightly simplify the generalization evaluation and allow the model to also be exposed to all possible `facts`. This formulation tests a model capacity to generalize to new `proof_steps` hence new `proofs`, after being trained on all `entities`, `relations` and `facts`. Since providing a training corpus covering all possible facts would significantly increase the training data, we instead reduce the number of entities in CLUTRR by replacing every entity by one of k[1] randomly sampled entity tokens, resulting in significantly fewer possible facts, and thus all facts being contained in the training set (Table 3).

Interpolation and Extrapolation. Having access to the levels of difficulty of each test examples, we evaluate both how Transformers can generalize to unseen proofs of the same difficulty as seen during training (*inductive* generalization); and how they can generalize to unseen proofs of *unseen difficulty levels*. In particular, we test *interpolation* in which the testing difficulty levels are not seen during training and less than training levels; and *extrapolation* in which the test difficulty levels are not seen during training and higher than training levels. This setup systematically tests the length generalization capabilities of Transformer-based language model in logical theorem proving.

4 Experiments and Analysis

We aim to answer the following questions to analyze the proof generation capabilities of Transformer-based language models (TLMs):

1. Are TLMs able to reason better after being trained to generate interpretable proofs expressed in natural language?
2. Which type of proofs are easier to learn and generate for TLMs?
3. Which type of proofs are more useful for TLMs to generate accurate answers?

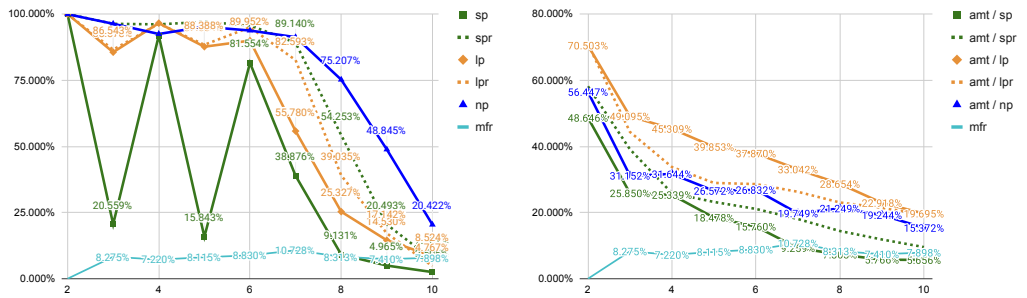
Setup. In all our experiments we used a Transformer decoder architecture (Liu et al., 2018) with 2.5M and 3.5M parameters with a vocabulary size of 90 and $1,800$ tokens for stories expressed with the `facts` and `amt` template respectively. Detailed parameter settings for our models are given in Appendix 6.3. We also ran preliminary experiments with a larger model (145M parameters) (Appendix 6.4), with a GPT2 model (Radford et al., 2019) (Appendix 6.5), and with a more complex network (an encoder-decoder transformer) (Appendix 6.6) but found similar conclusions or further investigation being required. We generate $390,000$ CLUTRR examples of level 2 to 10. We train the models on $300,000$ examples of levels $2, 4$ and 6 and evaluate the model on a test set of $10,000$ examples for all levels from 2 to 10. Specifically, we test levels 3 and 5 for interpolation, levels $2, 4$ and 6 for inductive generalization and levels $7, 8, 9$ and 10 for extrapolation.

Evaluation Metrics. In the following experiments, we evaluate both the generated *proof validity* and *answer accuracy*. The answer is defined as the first sentence after the "<ANSWER>" tag in the generated sequence. Since all answers during training were expressed using the `facts` template, we reverse this template to extract the $(entity, relation, entity)$ triple from the generated answer. If the extraction fails, we consider the generated answer wrong. We then compare the extracted triple to the ground truth provided in the CLUTRR dataset. For comparison, in all experiments, we also report the accuracy of the naive most-frequent-relation (MFR) baseline consisting of predicting the relation that is the most frequent in the training set for the queried entity pair.

A proof is defined as the ordered sequence of all sentences generated between the "<PROOF>" and "<ANSWER>" tokens. For validating a proof, since all proofs during training were expressed using the `facts` template, we reverse this template to extract all $(entity, relation, entity)$ triples from the generated proof sentences. If the extraction process fails at any point, the entire proof is considered invalid. The ordered sequence of each proof step is then evaluated against the transitivity rules defined by the CLUTRR environment. In addition, we also check that all the facts necessary for the proof are either given in the input story, or inferred from a previous proof step. If any of these condition fail, we consider the proof invalid.

No proof setup. In addition to the four proof strategies defined in Section 3.2, we also compare in all our experiments with a model that is trained to directly generate the answer after the story and query. In particular, this *no-proof* model is trained on sequences of "<STORY> [story] <QUERY> [query] <PROOF> none . <ANSWER> [answer]". This allows us to estimate how important is the proof for our models to be able to generalize.

[1] $k = 20$ in our case because we know that the maximum number of entities in a story is less than 20.



(a) stories expressed with the `facts` template.

(b) stories expressed with the `amt` template.

Figure 2: Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed, **np**=no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

4.1 Answer Accuracy

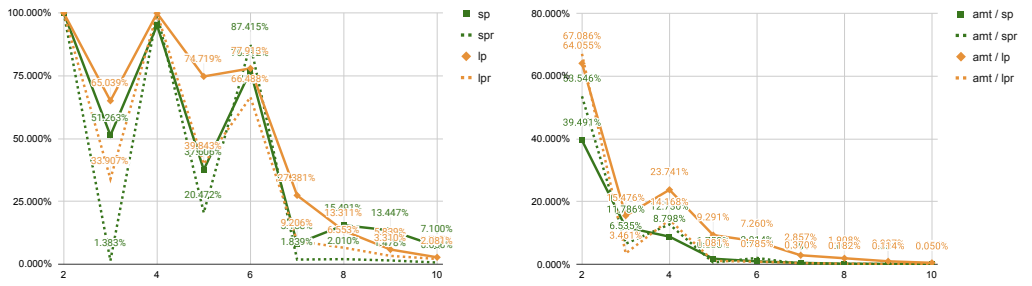
We evaluate the answer accuracy of models trained with different proof settings on the test set described earlier by Table 3. Each model is given as input a story, query, and the proof trigger token ("<STORY> [story] <QUERY> [query] <PROOF>"), and we allow them to decode the next tokens, that is, the proof followed by the answer.

Q: *Are TLMs able to generalize to unseen proof steps?* **A**: *For simple language, yes in interpolation and no in extrapolation. For complex language, No in both cases.*

In Figure 2a we evaluate models trained wit stories expressed with the `facts` template. We observe that in all proof setups, with the exception of short-proofs, TLMs are able to systematically generalize to predict the correct answer inferred from unseen `proof_steps` and `proofs`, both in inductive (levels 2, 4, 6) and for interpolation setup (levels 3 and 5). However, in all proof setups TLMs still have difficulties to extrapolate to longer problems requiring a larger number of reasoning steps, conforming to length generalization issues discovered in related tasks (Lake, 2019).
In Figure 2b we note that models trained on noisy `amt` stories fail to systematically generalize to predict the correct answer. In addition, we can see a linear decrease in accuracy with the level of difficulty. Having to de-noise the input stories to extract relevant kinship relations, in addition to running logical inference, makes the task much more challenging for our network. We conjecture that generalizing in this harder setting may require additional capacity added to the model, either in terms of model size, model architecture, training data, or a combination of all the above. For instance, we explore the benefit of fine-tuning GPT2 (Radford et al., 2019) in Section 6.5 as an initial step, but leave room for further improvement in future work.

Q: *Which reasoning strategy generalizes better?* **A**: *Backward-chaining is better than forward-chaining, but no-proof can be better than both. Long-proofs are better than short-proofs.*

We observe that backward proof strategies (spr, lpr) better help the model to answer accurately than their respective forward strategies (sp, lp) (Figure 2), with the exception of long proofs in the `amt` story template. This suggests that backward chaining is easier to learn, or easier to use, or both than forward chaining for TLMs. We believe this effect is due to the position-dependent exploitation of TLMs, where the answer is usually in the first generated proof-step in case of backward-chaining proofs. In addition, we note in Figure 2 that long-proofs (lp, lpr) yield better generalization performance than short-proofs (sp, spr) with the exception of reversed strategies in the `facts` story template.
It is also interesting to see that models trained to go directly to the answer by generating the "*none*" token as a proof tend to perform better than all other models required to generate the proof in `facts` stories (Figure 2a). One hypothesis is that the generated proof may be invalid most of the time and hence the extra information given by the proof is actually deteriorating the model's performance. To see if that may be the case, we next look at the validity of the generated proofs for all models (except the trivial no-proof).



(a) stories expressed with the `facts` template.

(b) stories expressed with the `amt` template.

Figure 3: Proof validity for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed, **np**=no-proof.

4.2 Proof Validity

We evaluate the proof validity of models trained with different proof settings on the test set (previously described by Table 3) in Figure 3. Similarly as above, each model is given as input a story and query and we trigger the model to decode the proof and answer with the trigger tokens "<PROOF>" and "<ANSWER>" respectively.

Q: *Which reasoning strategy is easier to generate?* **A**: *forward-chaining is easier than backward-chaining and long-proofs are easier than short-proofs.*

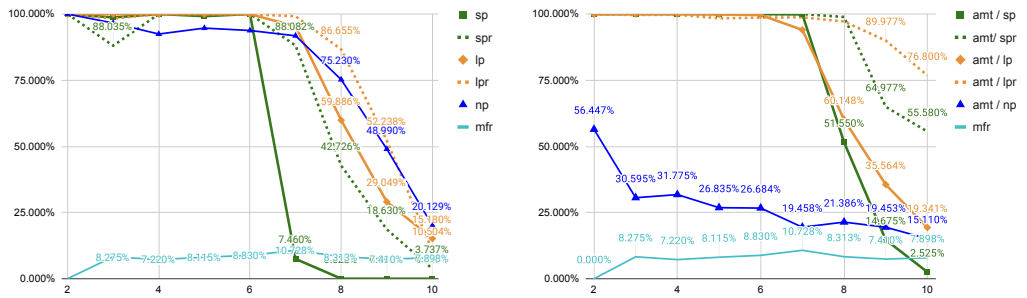
From Figure 3a we observe that forward-chaining strategies (sp, lp) tend to be easier to generate than their respective reversed strategies (spr, lpr). This is contrary to the previous observation where backward-chaining strategies were easier for the models to understand. We believe that this is due to the fact that the model has a higher chance of generating the first proof step correctly than the final proof step. Since backward chaining proofs contain the answer in the first proof step, when re-using that information to predict the answer, there is a higher chance that the answer will be correct. This explains why the answer accuracy of such model is relatively high while their proof validity is relatively low.
In addition, we observe that in both `facts` and `amt` stories (Figure 3), long proof strategies are easier to generate than shorter ones. This was not expected at first since long sequences are usually harder to model in language models. One hypothesis is that since long-proofs come from a systematic construction (see Appendix 6.2) they are easier to generate than the more arbitrary short proofs.

Q: *Are TLMs able to generate valid proofs of unseen lengths?* **A**: *No.*

We observe that valid proofs are difficult to generate for TLMs in unseen difficulty levels, both in interpolation and extrapolation setting (Figure 3a). This partially explains why the no-proof setting in the previous section yielded better generalization performances. In addition, we note in Figure3b that the generated proofs from models trained on noisy `amt` stories are mostly invalid. We believe that this is due to the fact that models need to denoise the information from the input story in addition to generating a valid proof, making the task much harder. To understand if models rely on the validity of the proof, we next evaluate their answer accuracy when given the real proof as input rather than the generated one.

4.3 Proof is given

To understand if models rely on the validity of the proof, we again evaluate the answer accuracy as in Section 4.1, but this time the models are given as input the story, the query and the real proof followed by the answer trigger token: "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>". We then let the language model decode the next tokens making up the answer. Note that the no-proof model is given "none" as its "[proof]" so we don't expect this model performance to change from Section 4.1.



(a) stories expressed with the `facts` template.

(b) stories expressed with the `amt` template.

Figure 4: Answer accuracy for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>" and they generate the answer. The models are trained on levels 2, 4, 6 only. Different proof settings are evaluated: **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed, **np**=no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

Q: *Are ground-truth proofs useful for TLMs to generalize systematically?* **A**: *Yes.*

When the proof is provided in the input, all models outperform the no-proof model in inductive and interpolation test cases (Figure 4). In extrapolation test cases, models trained on `facts` stories (Figure 4a) benefit from the proof compared to Section 4.1, and models trained with `amt` stories outperform the no-proof model (Figure 4b). This suggests that models do learn to use the correct proof to better generalize during inference.
However, as the difficulty of the examples increase, the generalization performance of all models decreases. Even when given the proof containing the correct answer, TLMs fail to copy the correct information from sequences of greater length than seen during training. Our hypothesis for this is that Transformers strongly rely on the position of the answer and have trouble learning simple tasks – such as copying the answer from the proof – if the information for this task happens at unseen positions.

Q: *Which reasoning strategy is easier to use when generating answers?* **A**: *backward-chaining is easier to use than forward-chaining and long-proofs are easier to use than short-proofs.*

Another interesting observation is that, in general, the reversed proofs (dotted lines in Figure 4) tend to be more useful than forward strategies for our model in generating the correct answer, aligning with our findings in Section 4.1. Similarly as above, we believe that this is due to the facts that Transformers strongly rely on the position of the answer. Indeed, in reversed proofs (**spr**, **lpr**), the answer is always in the first proof step, for which the position depends only on the story length; whereas in **sp** and **lp** the answer is always in the last proof step, for which the position depends both on the story length and on the proof length.
We also see that long, exhaustive proofs are easier to be used when generating the final answer, compared to short-proof strategies. This suggests that while being a longer sequence of tokens to encode, if a model was able to generate such proofs, it would ease its generalization capacities.

5 Conclusion

TLMs are state of the art models for a wide variety of natural language processing tasks. Given their widespread use, it is important to understand the limits of their ability to reason on knowledge expressed in natural language and to extrapolate learned inference procedures to unseen problem instances. Our explorations reveal multiple insights. Firstly, TLMs suffer from length-generalization issues in generating proofs. Secondly, TLMs get better at reasoning when trained with longer, exhaustive proofs. In addition, the fact that backward-chaining proof models perform better than forward-chaining ones makes us believe that backward-chaining strategies are easier to use albeit being harder to generate. Moreover, we find that no-proof models perform better than those trained to produce proofs. We conjecture that benefiting from naturally stated logical proof statements requires more complex internal representations. Recent work on developing position-agnostic attention mechanisms for Transformers (Dubois et al., 2020) can be useful as a future direction to develop

generalizable models. Furthermore, our results motivates the use of neuro-symbolic methods such as Neural Theorem Provers (Rocktäschel and Riedel, 2017) as an alternative avenue to achieving systems that systematically generalize on logical and compositional reasoning tasks. Combining these approaches with large pre-trained language models is left as future research. We hope that this work will inspire research on the systematic generalization capacity of language models and motivate further study and the creation of neural models with greater reasoning capacity.

Broader Impact

Transformer based models have been very effective for various language understanding and generation tasks. Thus, it is crucial to understand their limitations and issues in generalization to unseen data. In this work, we rely on systematic tests to trigger Transformer-based models to generate an interpretable proof in natural language, and then evaluate the robustness properties of that proof. This research can help us understand the reasoning strategies employed by Transformer-based models for both inference and generation. Using first-order logic based datasets, we explicitly test the logical validity of such systems in practice, which can shed some light into developing more robust and systematic models in the future.

Due to the recent success of Transformer-based models and large-scale pre-training, there is significant interest in the applications of these models to real world scenarios such as: Dialogue, Question Answering and text-classification. Failure of such systems could produce nonsensical, wrong or racially-biased results (Henderson et al., 2018). As such, logical analysis of the generation capabilities of Transformer-based models, such as in this work, could have an impact on building safer, more robust and interpretable systems in these domains. However, the fact that proof free inference works so well, may also imply that models which generate proofs, do so in a decoupled way from the computations yielding the final answer. This could give a user a false sense of explainability, as proofs, or explanations may be generated by a separate and decoupled computation within the Transformer model.

Acknowledgments and Disclosure of Funding

The authors would like to acknowledge support from Element AI for providing computational resources which were used to run the experiments in this work. We also acknowledge the help provided by Sandeep Subramanian in sharing part of his experimental code. Nicolas is partially funded by a scholarship from the Fonds de Recherche Quebec Nature et Technologie. We thank CIFAR for their support through the CIFAR AI Chairs program. We also thank NSERC and PROMPT for their support.

References

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. 2019a. Systematic generalization: What is required and can it be learned? In *Proceedings of the 2019 International Conference on Learning Representations*.

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

Marco Baroni. 2020. Linguistic generalization and compositionality in modern artificial neural networks. *Philosophical Transactions of the Royal Society*, 375(1791).

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Noam Chomsky. 1957. Logical structures in language. *American Documentation*, 8(4):284–291.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence,*

IJCAI-20, pages 3882–3890. International Joint Conferences on Artificial Intelligence Organization. Main track.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Ishita Dasgupta, Demi Guo, Samuel J. Gershman, and Noah D. Goodman. 2019. Analyzing machine-learned representations: A natural language case study. *arXiv preprint arXiv:1909.05885*.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, IJCAI-07*, pages 2462–2467. International Joint Conferences on Artificial Intelligence Organization.

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*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yann Dubois, Gautier Dagan, Dieuwke Hupkes, and Elia Bruni. 2020. Location Attention for Extrapolation to Longer Sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 403–413, Online. Association for Computational Linguistics.

Richard Evans and Edward Grefenstette. 2018. Learning explanatory rules from noisy data (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5598–5602. International Joint Conferences on Artificial Intelligence Organization.

Yoav Goldberg. 2019. Assessing bert's syntactic abilities. *arXiv preprint arXiv:1901.05287*.

Emily Goodwin, Koustuv Sinha, and Timothy J. O'Donnell. 2020. Probing linguistic systematicity. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1958–1969, Online. Association for Computational Linguistics.

Peter Henderson, Koustuv Sinha, Nicolas Angelard-Gontier, Nan Rosemary Ke, Genevieve Fried, Ryan Lowe, and Joelle Pineau. 2018. Ethical challenges in data-driven dialogue systems. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 123–129.

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 *Proceedings of the 2020 International Conference on Learning Representations*.

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

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*, pages 2879–2888.

Adam Liška, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In *Proceedings of the 2018 International Conference on Learning Representations*.

Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. 2020. Differentiable reasoning on large knowledge bases and natural language. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, volume 34 no.4: AAAI Technical Tracks Machine Learning*, pages 5182–5190.

Richard Montague. 1970. Universal grammar. *Theoria*, 36(3):373–398.

Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium. 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)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

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(1).

Tim Rocktäschel and Sebastian Riedel. 2017. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems 30*, pages 3788–3800. Curran Associates, Inc.

Stuart Russell and Peter Norvig. 2020. *Artificial intelligence: a modern approach*, fourth edition. Pearson.

Taro Sekiyama, Akifumi Imanishi, and Kohei Suenaga. 2017. Towards proof synthesis guided by neural machine translation for intuitionistic propositional logic. *arXiv preprint arXiv:1706.06462*.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. 2019. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4506–4515, Hong Kong, China. Association for Computational Linguistics.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. In *Proceedings of the 2019 International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Leon Weber, Pasquale Minervini, Jannes Münchmeyer, Ulf Leser, and Tim Rocktäschel. 2019. NLProlog: Reasoning with weak unification for question answering in natural language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6151–6161, Florence, Italy. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc.

6 Supplementary Material

6.1 Original CLUTRR evaluation

ORIGINAL TEST	lvl.**2**	lvl.**3**	lvl.**4**	lvl.**5**	lvl.**6**	lvl.**7**	lvl.**8**	lvl.**9**	lvl.**10**
proofs (*many proof steps*)	99.62%	0%	0%	0%	0%	0%	0%	0%	0%
proof steps ("*since A-r1-B and B-r2-C then A-r3-C*")	99.62%	0%	99.96%	0%	100%	0%	0%	0%	0%
facts (*A-r-B*)	100%	0.47%	100%	0.83%	100%	0.20%	0.20%	0.10%	0.42%
entities (*A*)	100%	23.81%	100%	35.72%	100%	26.19%	21.43%	30.95%	30.95%
relations (*r*)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4: Percentage of the **original test** proof building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value close to 100% to better visualize which building blocks were entirely contained in the training set.

The original CLUTRR data generation framework made sure that each test `proof` is not in the training set in order to test whether a model is able to generalize to unseen proofs. Initial results on the original CLUTRR test sets resulted in strong model performance ($\sim 99\%$) on levels seen during training ($2, 4, 6$) but no generalization at all ($\sim 0\%$) to other levels. After further analysis, we noticed that due to the cloze style nature of CLUTRR tasks, the first names representing entities were chosen arbitrarily. This resulted in level-k test set's `proof_steps` and `facts` also being in the level-k training set. In addition, level-k test set's `entities` were mostly seen *only* in level-k training set. This resulted in a big overlap between training and test sets for examples of the same level, but a weak overlap on other levels as we can see in Table 4.

NAMED TEST	lvl.**2**	lvl.**3**	lvl.**4**	lvl.**5**	lvl.**6**	lvl.**7**	lvl.**8**	lvl.**9**	lvl.**10**
proofs (*many proof steps*)	2.13%	0%	0%	0%	0%	0%	0%	0%	0%
proof steps ("*since A-r1-B and B-r2-C then A-r3-C*")	2.13%	0%	1.33%	1.74%	1.42%	1.80%	1.38%	0.99%	1.40%
facts (*A-r-B*)	15.48%	5.52%	6.77%	10.92%	6.38%	9.63%	10.51%	10.33%	8.33%
entities (*A*)	100%	100%	100%	100%	100%	100%	100%	100%	100%
relations (*r*)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 5: Percentage of the **Named test** proof's building blocks also present in the training set (composed of levels 2, 4, 6) for all levels. We colored all cells with a value of 100% to better visualize which building blocks were entirely contained in the training set

In our case, the entity names are important to evaluate systematic generalization. We want to evaluate the capacity of a model to generalize to new `facts`, `proof_steps`, and `proofs`, but keeping the `entities` and `relations` the same. We thus modified the original CLUTRR dataset to select test entities according to entities present in the training set. We devise a test set that uses all `relations` and `entities` from the training set but new `facts`, `proof_steps` and `proofs` for all levels. We call this dataset the *Named* data: all entities are referred by their original first name. Train and test overlap percentages between all building blocks are in Table 5.

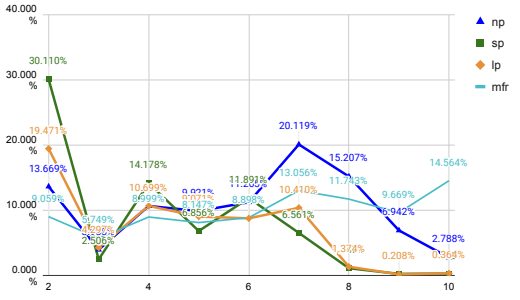
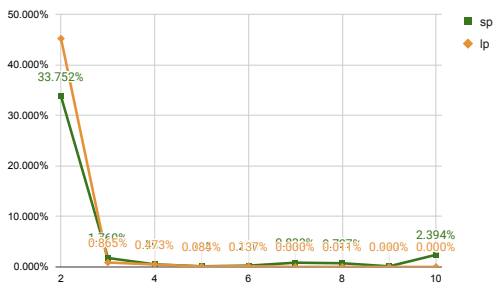
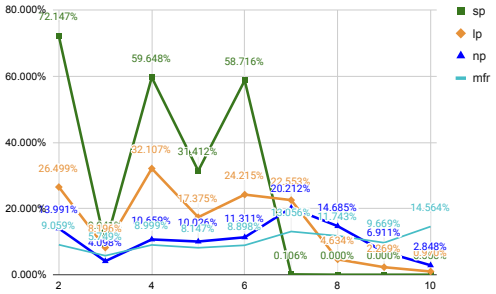


Figure 5: Answer accuracy on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and asked to generate the proof and answer. Models are trained on levels $2, 4, 6$ only. Different proof settings are evaluated: **sp**=short-proof, **lp**=long-proof, **np**=no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

Given as input the story and the query followed by the proof trigger token ("<STORY> [story] <QUERY> [query] <PROOF>") the model generated the corresponding proof ans answer. We report



(a) Proof validity on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and asked to generate the proof and answer.



(b) Answer accuracy on the Named test for all levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>" and asked to generate the answer.

Figure 6: Evaluation of models trained on levels 2, 4, 6 only.

in Figure 5 the answer accuracy and in Figure 6a the proof validity of all our models. Similarly, in Figure 6b we report the answer accuracy of our models when they are given as input the story, the query and the real proof, followed by the answer trigger token ("<STORY> [story] <QUERY> [query] <PROOF> [proof] <ANSWER>").

Experiments on this setup show that Transformer language models fail to generalize to unseen `facts`. Indeed, due to the presence of a large number of entities in CLUTRR, we end up with combinatorially large number of possible facts. The model may thus not be able to learn how to represent each entity effectively, hence reducing its chances to learn higher-order structures such as unseen `facts`.

6.2 Long Proof pseudo-code

```
def get_long_proof(story_facts, rules, query):
    """
    :params story_facts: list of (e_1, r, e_2) facts
    :params rules: list of composition rules. each rule is a dict
                   of the form {r1--r2: r3}
    :params query: tuple of entities for which we must find a relation (src, tgt)
    """
    proof = []  # list of proof steps to return

    # get all known relations (original, and reversed)
    all_facts = []
    for (e1, r, e2) in story_facts:
        inv_r = reverse_fact(e1, r, e2)
        all_facts.append((e1, r, e2))
        all_facts.append((e2, inv_r, e1))

    # go through every possible pair of facts
    for f1, f2 in itertools.combinations(all_facts, 2):
        e11, r1, e12 = f1
        e21, r2, e22 = f2
        inv_r1 = reverse_fact(e11, r1, e12)
        inv_r2 = reverse_fact(e21, r2, e22)

        # find the possible AB+BC combination.
        # there are 4 possible ways to combine 2 sentences with 2 entities each (1 in common):
        if e11 == e21 and e12 != e11 and e12 != e22:
            # AB+BC <=> inv_f1+f2
            A, new_r1, B = e12, inv_r1, e11
            B, new_r2, C = e21, r2, e22
            inv_r1 = r1
        elif e11 == e22 and e12 != e11 and e12 != e21:
            # AB+BC <=> f2+f1
```

```
            A, new_r1, B = e21, r2, e22
            B, new_r2, C = e11, r1, e12
            # swap inv_r1 and inv_r2
            inv_r1, inv_r2 = inv_r2, inv_r1
        elif e12 == e21 and e11 != e12 and e11 != e22:
            # AB+BC <=> f1+f2
            A, new_r1, B = e11, r1, e12
            B, new_r2, C = e21, r2, e22
        elif e12 == e22 and e11 != e12 and e11 != e21:
            # AB+BC <=> f1+inv_f2
            A, new_r1, B = e11, r1, e12
            B, new_r2, C = e22, inv_r2, e21
            inv_r2 = r2
        else:
            # invalid pair of facts
            continue

        # try to combine AB+BC
        if new_r1--new_r2 in rules:
            r3 = rules[new_r1--new_r2]
            inv_r3 = reverse_fact(A, r3, C)
            all_facts.append((A, r3, C))
            all_facts.append((C, inv_r3, A))
            proof.append(since A new_r1 B and B new_r2 C then A r3 C)
        # try to combine CB+BA
        elif inv_r2--inv_r1 in rules:
            r3 = rules[inv_r2--inv_r1]
            inv_r3 = reverse_fact(C, r3, A)
            all_facts.append((C, r3, A))
            all_facts.append((A, inv_r3, C))
            proof.append(since C inv_r2 B and B inv_r1 A then C r3 A)
        else:
            # invalid pair of facts
            continue

        # check if we found the link between the two queried entities
        (A, r, B) = all_facts[-1]
        if A==query[0] and B==query[1]:
            break
        if A==query[1] and B==query[0]:
            break

    return proof
```

6.3 Experiments parameter settings

	small	**large**
patience	20	20
batch size	512	256
float precision	16	16
embedding dimension	192	768
number of layers	5	20
dropout	0.1	0.1
transformer mlp hidden size	768	3072
attention heads	3	12
max length	$1,024$	512
activation	gelu	gelu
number of warmup steps	$20,000$	$20,000$
optimizer	adam	adam
total parameters	$\sim 3,000,000$	$\sim 145,000,000$

Table 6: Parameter settings.

All experiments in the main section of the paper were run with the **small** model size.

Additional experiments in Section 6.4 were run with the **large** model size.

6.4 More parameters

In this section we report the answer accuracy of a model trained with ∼145M parameters and compare its generalization performance with our initial smaller network (∼2.5M parameters). Models are trained on levels 2, 4 and 6. Each model is given the story and query as input, and triggered to generate the proof and answer with the "<PROOF>" and "<ANSWER>" tokens respectively.

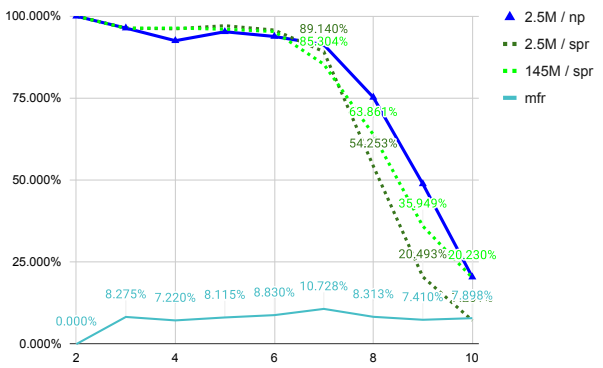


Figure 7: Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. Models are trained on levels 2, 4, 6 only. Stories are expressed with the `facts` template. Different proof settings are evaluated: **np**=no-proof and **spr**=short-proof-reversed. We also report the naive most-frequent-relation (**mfr**) baseline. Results on other proof settings with the 2.5M parameter network can be found in Figure 2a.

We observe in Figure 7 that the generalization capacity of the larger 145M network is almost identical to the smaller 2.5M parameter network trained on the same data (`facts` stories and short-proof-reversed). In addition, we also observe that the 145M model trained on reversed short proofs (145M / spr) is not better than the 2.5M model trained without any proof (2.5M / np). Overall, results show that model size improves only marginally the generalization capacity in our task.

6.5 Fine-tuning GPT2

In this section we report the answer accuracy of GPT2 models (Radford et al., 2019) trained from-scratch (`gpt2FS-`) on the CLUTRR dataset and of pre-trained GPT2 models fine-tuned (`gpt2FT-`) on the CLUTRR dataset. We leverage the GPT2 implementation from the huggingface library (Wolf et al., 2019). The resulting models have ∼125M parameters. In all experiments the models are trained on stories expressed in the `amt` template. Models are fine-tuned on levels 2, 4 and 6. Each model is given the story and query as input, and triggered to generate the proof and answer with the "<PROOF>" and "<ANSWER>" tokens respectively.

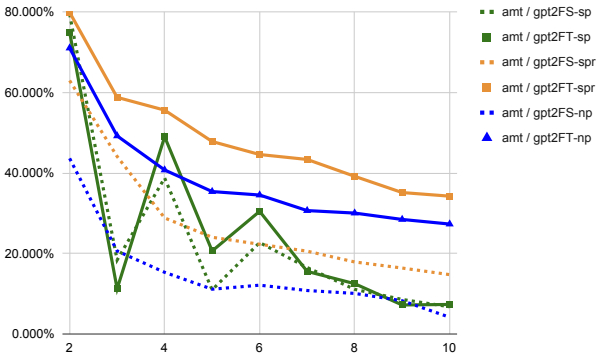


Figure 8: Answer accuracy for all test levels from 2 to 10. The models are given as input "<STORY> [story] <QUERY> [query] <PROOF>" and they generate the proof and answer. Models are fine-tuned on levels 2, 4, 6 only. Stories are expressed with the `amt` template. Different proof settings are evaluated: **sp**=short-proof, **spr**=short-proof-reversed, **np**=no-proof. We compare the performance of models trained from scratch (dotted lines; `gtp2FS-`) and of fine-tuned models (solid lines; `gpt2FT-`).

In Figure 8 we observe that in general, fine-tuned models perform better than the ones trained from scratch. We can also see that reversed-proof strategies are better than their forward proof counterpart, which is in accordance with what we discussed in Section 4.1. Although fine-tuning seems to improve the generalization capacity of GPT2, it is also interesting to note that the benefit of fine-tuning GPT2 on short-proofs (sp) is negligible compared to the benefits of fine-tuning GPT2 on short-proofs-reversed (spr) or no-proof (np). This suggests that fine-tuning alone is not enough to yield strong generalization performance, but the choice of proof strategy also influences greatly the answer accuracy.

6.6 Encoder-Decoder Network

In this section we evaluate the answer accuracy of sequence-to-sequence models trained on `facts` templated stories of level 2, 4 and 6. These models consist of a 5-layer Transformer encoder and a 5-layer Transformer decoder, each of them following the same parameter settings than what is described in the '**small**' column of Table 6. This resulted in 5.22M parameter models. Sequence-to-sequence models are trained to encode the story and question with the encoder, and generate the proof and answer with the decoder. Models trained on levels 2, 4 and 6. Each model is given the story and query as input, and triggered to generate the proof and answer with the "<PROOF>" and "<ANSWER>" tokens respectively.

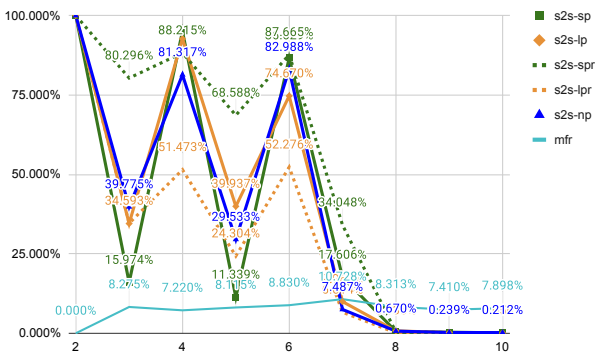


Figure 9: Answer accuracy for all test levels from 2 to 10. The models encodes the input "<STORY> [story] <QUERY> [query]" and they decode the proof and answer. Models are trained on levels 2, 4, 6 only. Stories are expressed with the `facts` template. Different proof settings are evaluated: **sp**=short-proof, **spr**=short-proof-reversed, **lp**=long-proof, **lpr**=long-proof-reversed, **np**=no-proof. We also report the naive most-frequent-relation (**mfr**) baseline.

In the results shown in Figure 9, we see that sequence-to-sequence models do not generalize well to unseen difficulty levels, both in extrapolation settings (levels 7–10) but also in interpolation settings (levels 3 and 5). This suggests that encoder-decoder architectures are more sensible to the sequence length seen during training. On the other hand, it is important to note that the encoder network was trained with the auto-regressive language modeling objective back-propagated from the decoder. It would be interesting to see if pre-training the encoder with a more traditional objective, that is masked language modeling (Devlin et al., 2019), would improve the generalization performance. We leave this exercise as future work. In addition, we plan to explore pre-trained models such as T5 (Raffel et al., 2020) in future work in order to improve performance with this type of architecture.