Propositional Reasoning via Neural Transformer Language Models

Anthony Tomasic[1], Oscar J. Romero[1], John Zimmerman[1] and Aaron Steinfeld[1]

[1]*Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, 15213,USA*

Abstract

This paper presents exploratory work on how the key components of transformer-based neural language models (self-attention mechanism and large-scale pre-trained models) can be leveraged to perform context retrieval, symbol manipulation, and propositional reasoning. We fine-tuned off-the-shelf GPT-2 and GPT-3 language models to simulate the propositional logic resolution of non-recursive rules that combine conjunction, disjunction, and negation connectives. Next, we conducted a series of experiments and evaluated the performance of different syntactic representations. The promising results show that a transformer model can i) generalize an inference algorithm for propositional resolution, and ii) learn to solve simple instances of view updates (a well-studied problem in the field of databases) by leveraging the propositional reasoning framework.

Keywords

transformer neural language models, propositional reasoning, view updates problem

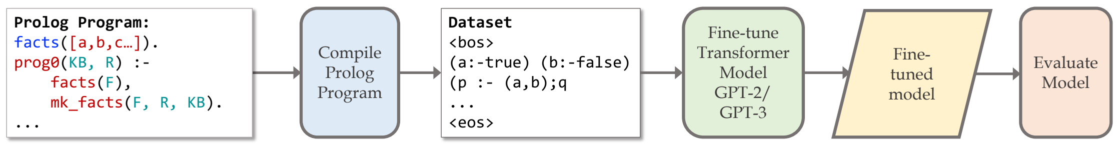
1. Introduction

Transformer models continue to be the preferred model for many natural language processing (NLP) tasks, e.g. machine translation, natural generation, summarization, etc. In general, the larger the transformer model, the larger the context that captures the reasoning of a task, resulting in better performance. This scaling relationship between the size of the model and performance had led to a race to construct ever larger models [1].

However, symbolic reasoning is not directly available in transformer models [2]. For example, a common NLP problem that end-to-end transformer-based applications have yet to deal with is abstraction over concepts. To circumvent this issue, delexicalization/lexicalization techniques (i.e., the replacement of specific slot values by their corresponding generic slot tokens to allow learning value-independent parameters) need to be used by a human designer [3, 4]. This manipulation of de/lexicalized terms can be considered a simple form of symbolic manipulation and reasoning. More generally, the lack of symbolic reasoning functionality has lead researchers

NeSy 2022, 16th International Workshop on Neural-Symbolic Learning and Reasoning, Cumberland Lodge, Windsor, UK

tomasic@andrew.cmu.edu (A. Tomasic); oscarr@andrew.cmu.edu (O. J. Romero); johnz@andrew.cmu.edu (J. Zimmerman); steinfeld@cmu.edu (A. Steinfeld)
https://www.cs.cmu.edu/~tomasic/ (A. Tomasic); http://www.cs.cmu.edu/~oscarr/ (O. J. Romero); https://design.cmu.edu/people/faculty/john-zimmerman/ (J. Zimmerman); http://www.cs.cmu.edu/~astein/ (A. Steinfeld)
0000-0001-7864-0364 (A. Tomasic); 0000-0002-4470-8209 (O. J. Romero); 0000-0001-5299-8157 (J. Zimmerman); 0000-0003-2274-0053 (A. Steinfeld)



```
Prolog Program:
facts([a,b,c…]).
prog0(KB, R) :-
    facts(F),
    mk_facts(F, R, KB).
...
```

```
Dataset
<bos>
(a:-true) (b:-false)
(p :- (a,b);q
...
<eos>
```

Figure 1: Pipeline to fine-tuning the transformer model. A Prolog program generates training data that is used to fine-tune both GPT-2 and GPT-3 transformer models for propositional resolution.

to extend the deep neural network models to directly represent symbolic reasoning [5, 6, 7, 8, 9].

In this paper, we consider a complementary approach where we demonstrate the capability of transformer models to directly simulate reasoning in propositional logic. We use this reasoning to implement a simple form of belief revision [10], namely propositional view updates [11].

The main contributions of this paper are: (i) A data representation of conjunction, disjunction and negation symbolic logic designed to represent propositional resolution; (ii) A methodology for generation of fine-tuning examples; (iii) A fine-tuned transformer model that learns the underlying symbolic inference algorithm for propositional resolution; and (iv) An experimental analysis of the performance of the fine-tuned model. In addition, we explore the implications of these contributions in two ways: (i) We study the impact of the transformer model size and structure on performance; and (ii) We apply this work to propositional view updates.

Transformer models have been demonstrated to generate semi-structured representations [12] that combine structured data, API calls, and natural language comprehension and synthesis. This paper points the way to extend transformer models to perform belief revision over these semi-structured representations.

To illustrate a direct application of our contribution, consider the following conversational interaction extracted from the logs of our prior work [12]:

```
User:   I want to take a bus to Carnegie Mellon University
System: I have 4 buses for you...
        The bus 41 leaves at 4:33pm and costs $5.50.
        There is 1 transfer. You'll need to take a connection with bus 61C...
```

The system response is only partially correct, the bus information and transfer are correct, but the price information is wrong. The system has calculated the price of the trip as the sum of each leg, but this sum is incorrect since transfers to another bus are free. Enabling a conversational system with view updates would permit, for example, the conversation to proceed as "User: No, the price information is wrong" and then start a (admittedly complex) dialog to correct the sum.

In this paper, we adopt conventional logic programming syntax, semantics, and terminology for programs, rules, terms, goals, and resolution for non-recursive propositional programs.

2. Methodology

Our objective is to fine-tune off-the-shelf transformer models GPT-2 [13] and GPT-3 [1] to simulate propositional logic resolution of rules (see Figure 1). For instance, consider the following example (written in Prolog syntax).

```
1)  p :- q,r.    % rule : p ← q ∧ r          5)  q,r.     % resolution...
2)  q :- true.   % fact : q ← true           6)  true,r.
3)  r :- true.   % fact : r ← true           7)  r.
4)  ?- p.        % query : is p true?        8)  true.
```

The example is arranged to state the program in terms of facts and rules (lines 1-3), then a goal (line 4), and then the resolution of the goal (lines 5-8). Resolution proceeds by alternating between two types of transformations: rule retrieval and goal rewriting. The logic proof above can be encoded as a sequence of tokens (that serve as the input to GPT models) as follows:

```
<bos> (q :- true) (r :- true) (p :- q,r) <soq> p
↳ <sop> (p :- q,r) (q :- true) true (r :- true) true <eos>
```

Where <bos> (beginning of sentence), <soq> (start of query), <sop> (start of proof), and <eos> (end of sentence) are special tokens, which serve as segment indicators. During fine-tuning, the entire sequence is presented as an example, while during testing, the sub-sequence <bos>. . . <sop> is used to prompt the model and the remaining sub-sequence serves as a gold label.

In addition to fine-tuning resolution, propositional programs contain some symmetries that must be accounted for when generating examples: (i) The ordering of terms and clauses is not relevant; and (ii) Clause bodies are composed with conjunction (**,**), disjunction (**;**), and negation (**not**) operators. These symmetries require that the examples randomize certain symbols so that the transformer learner does not incorrectly model sequential aspects of the problem.

In addition, the algorithm of propositional resolution must be reflected in the fine-tuning data: (i) The state of the resolution is represented by a sequence of goals; (ii) Selection Rule: the left most goal is selected and replaced with its definition; (iii) **true** and **false** goals are removed when they appear; and (iv) Resolution ends when no goals remain.

Three remaining issues must be settled to complete the specification of the representation: (i) A goal without a corresponding clause requires some interpretation. We eliminate this case by requiring all programs to have clauses for any goal. Thus, every chain of reasoning ends with either **true** or **false**; (ii) Disjunction can be represented in two ways: via multiple rules with the same head and by a semicolon disjunction operator. and (iii) Resolution is designed to determine the truth of a goal and thus the search terminates with the first **false** or all goals end with **true**. However, as discussed below, the resolution of *view update reasoning* requires the exploration of all proof paths. We explore this termination condition in the experiments.

In the context of relational and deductive databases, view update refers to the problem of translating update requests against a view into update requests against the base data [11]. View updates are a simple form of belief revision reasoning [10] that restricts the update to be the head of a rule and the *translation* of the update to be facts (i.e., the body of the rule). A translation is an update to the program that makes the view update true. So, given the program:

```
1) p :- q;r.              3) q :- false.              5) s :- false.
2) p :- s.                4) r :- true.
```

The view update `insert p` has two translations: `insert q` (i.e., replace `q :- false` with `q :- true`) and `insert s`. Note that `insert r` is redundant and thus not part of the translation. View update translations are naturally ambiguous since there are many ways to translate

Experiment	conj_{tr}	conj_{ts}	disj	neg	rules	expl	ctx	Experiment	conj_{tr}	conj_{ts}	disj	neg	rules	expl	ctx
$\text{Conj}_{\text{explore}}$	1-5	1-5	0	0	3	yes	no	$\text{Conj}_{\text{gpt3\_in}}$	10	1-10	0	0	3	no	no
$\text{Conj}_{\text{early}}$	1-5	1-5	0	0	3	no	no	$\text{Conj}_{\text{gpt3\_ex}}$	10	11-12	0	0	3	no	no
Conj_{ctx}	1-5	1-5	0	0	3	no	yes	$\text{Disj}_{\text{explore}}$	1-5	1-5	1-5	0	3	yes	no
$\text{Conj}_{\text{intrpl}}$	10	1-10	0	0	3	no	no	$\text{Disj}_{\text{early}}$	1-5	1-5	1-5	0	3	no	no
$\text{Conj}_{\text{extrpl}}$	10	11-12	0	0	3	no	no	$\text{Neg}_{\text{backtrack}}$	1-3	1-3	1-3	1-3	3	no	no
Conj_{10g}	1-10	1-10	0	0	3	no	no	$\text{View}_{\text{update}}$	1-3	1-3	1-3	1-3	3	yes	yes

Table 1
Parameters for experiments. conj_{tr}: # of conjuncts in train dataset. conj_{ts}: # of conjuncts in test dataset. disj: # of disjuncts. neg: # of negations. rules: # of rules per example. expl: "yes" = explore all branches, "no" = early termination. ctx: carry context.

an update. A simple method for reducing ambiguity annotates terms in a rule to indicate a preference for a particular update [14, 15]. Annotating the second rule implies that `insert s` is the preferred translation. To compute all possible view update translations, the approach we use here explores all solutions to the goal `p` without early termination and then reverse engineers the proof trees to generate translation by converting false facts to true ones, or visa versa. This same reasoning extends to deletion (each true proof must be falsified) and negation (flip between insertion and deletion). The experiments below reflect these conditions.

For each example, a generator program (written in Prolog – refer to Appendix A for further details) creates a logic program by randomly drawing the names of terms from a fixed vocabulary, randomly ordering the clauses, and randomly choosing among the operators in a syntactically valid way. The generator program then randomly chooses a rule head as the initial goal of the program. Finally, the generator program executes resolution on the generated logic program and the initial goal, and the resulting step-by-step execution is presented as a training example for the model to learn to perform resolution (Figure 1).

3. Experiment Framework

We used the open-source implementation of GPT-2 transformer architecture[1] (medium size) with default values for Adam learning rate (5e-5), epsilon for Adam optimizer (1e-8), and batch size (4). To generate coherent and syntactically valid outputs [16], we set parameters temperature (0.5), top-p nucleus sampling (0.95) and top-k sampling (50) using grid search.

We ran 12 experiments where the transformer model was fine-tuned and tested to perform resolution for conjunction, disjunction, and negation. For each experiment, we randomly generated 3 datasets of increasing size (1K, 10K, and 100K examples) with a 75/25 train/test split. We reported average accuracy (for an exact match with the test example) and an error analysis.

$\text{Conj}_{\text{explore}}$: in this experiment, the transformer model learns to explore all the paths of the search space defined by a rule clause, where conjunction is the only logical operator used to connect the rule's goals (see parameters on Table 1). The representation used is as follows:

```
<bos> (t :- true) (c :- false) (l :- false).. (e :- l,c,r,t)...  <soq> e
↳ <sop> (e :- l,c,r,t) (l :- false) false. (c :- false) false. (r :- false) false.
```

[1]https://github.com/huggingface/transformers

```
↳ (t :- true) true. <eos>
```

Note that the model must remember the context of the derivation. By "context" we mean the remaining goals to be resolved (e.g., goals in the sequence `l,c,r,t` is the derivation context).

Conj$_{\text{early}}$: this experiment is similar to Conj$_{\text{explore}}$, except that the reasoning process stops when a false term is found (early termination). That is, stopping at `(l :- false) false`.

Conj$_{\text{ctx}}$: similar to experiment Conj$_{\text{early}}$, except that the context of the derivation is carried at each step. We aimed to determine whether this representation improves the performance of the system by repeatedly presenting relevant stimuli to the transformer's attention mechanism. The following representation was used (the context is enclosed by squared brackets):

```
<bos>  ... <soq> e <sop> (e :- l,c,r,t).  ((l :- false),[c,r,t]) false,[c,r,t].
↳ ((c :- false),[r,t]) false,[r,t]. ((r :- false),[t]) false,[t]. ((t :- true)) <eos>
```

Conj$_{\text{10g}}$: a copy of experiment Conj$_{\text{early}}$ except that we increased the number of goals per rule (1-10) for both train and test datasets. This experiment serves as a baseline for comparison against experiments Conj$_{\text{intrpl}}$ and Conj$_{\text{extrpl}}$.

Conj$_{\text{intrpl}}$: We evaluated whether the transformer model could interpolate the resolution of clauses by training the model on 10 goals and evaluating it on tests with less than 10 goals.

Conj$_{\text{extrpl}}$: alike experiment Conj$_{\text{10g}}$ except that we trained the transformer model on rules with 10 goals and tested it on rules with 11-12 goals as a study for extrapolation.

Conj$_{\text{gpt3\_in}}$ and **Conj$_{\text{gpt3\_ex}}$**: We compared the performance of GPT-2 vs GPT-3 models by running experiments Conj$_{\text{intrpl}}$ and Conj$_{\text{extrpl}}$ on a GPT-3 fine-tuned model[2].

Disj$_{\text{explore}}$: in this experiment, the transformer model is fine-tuned to learn the procedural semantics for resolution given a specific arrangement of goals connected by conjunctions and/or disjunctions. For instance, given the facts `(g :- true) (j :- false) (q :- true)` and a rule `v`, the 4 possible derivations are:

```
1. (v :- g,j,q)                          3. (v :- g,(j;q))
        [g,j,q] [true,j,q]                       [g,(j;q)]  [true,(j;q)]
        [j,q] [false,q]   [q] [false]            [[j],[false]];[[q],[true]]
2. (v :- g;j,q)                          4. (v :- g;j;q)
        [[g,[true]]];[[j,q],[false,q]]           [[g],[true]];
        [[q],[true]]                             [[[j],[false]];[[q],[true]]]
```

Disj$_{\text{early}}$: this experiment is similar in nature to Disj$_{\text{explore}}$, except that an early termination is triggered when a false term is found on a conjunction branch.

Neg$_{\text{backtrack}}$: in this experiment, we added the negation operator to the training examples, so the model must learn backtracking, resolution, and negation. Consider the following example:

```
<bos> (r :- false) (z :- true) ...         [[not not r]]   [[not not (r :- false)]]
(q :- not r) (p :- not q;r,z) ...          [[not not false]]   [[not true]]
<soq> p <sop>                              [[false]]  [r,z]
[p :- not q;r,z]   [[not (q :- not r)]]    [false,z]  [fail] <eos>
```

[2]https://openai.com/api/

Experiment	Dataset size			Experiment	Dataset size			Experiment	Dataset size		
	1K	10K	100K		1K	10K	100K		1K	10K	100K
$\text{Conj}_{\text{explore}}$	0.55	0.88	1.00	$\text{Conj}_{\text{intrpl}}$	0.50	0.57	0.68	$\text{Disj}_{\text{explore}}$	0.27	0.54	0.98
$\text{Conj}_{\text{early}}$	0.66	0.93	1.00	$\text{Conj}_{\text{extrpl}}$	0.68	0.96	0.99	$\text{Disj}_{\text{early}}$	0.31	1.00	1.00
Conj_{ctx}	1.00	1.00	1.00	$\text{Conj}_{\text{gpt3\_in}}$	1.00	N/A	N/A	$\text{Neg}_{\text{backtrack}}$	0.33	1.00	1.00
Conj_{10g}	0.69	1.00	1.00	$\text{Conj}_{\text{gpt3\_ex}}$	1.00	N/A	N/A	$\text{View}_{\text{update}}$	0.17	0.90	1.00

Table 2
Average accuracy results for the experiments.

In the example above, the transformer model tries to satisfy first the left-most goal of p, that is, `not q`. Since q is a rule, then the model reads the rule `(q :- not r)` and retrieves its value `not (r :- false)`. Then, the double negation is applied to goal r, and since the goal sequence is not satisfied, the model "backtracks" to the next branch `[r,z]`. At this point, the model will fail to satisfy the goal given that r is false and no more branches remain. Note that the transformer model does not have any explicit notion of backtracking.

View$_{\textbf{update}}$: in this experiment, for each rule, we randomly marked one or more goals as the preferred derivation paths using the keywords *ins* (insertion) and *del* (deletion). We fine-tuned a model to understand this labeling and to explore each derivation path while carrying along a context (i.e., the results from previously resolved derivation paths). A simple case would be:

```
<bos> (q :- false) (g :- true) (s :- true)...
(d :- ins q,ins not g,s) ...
<soq> ins d <sop>
[ins [(d :- ins q,ins not g,s)]]
[ins [ins q,ins not g,s]]
[ins [ins (q :- false),ins not g,s]]
[ins [ins q false,ins not g,s]]
[ins q][ins [ins not g,s]]
[ins q][ins [del g,s]]
[ins q][ins [del (g :- true),s]]
[ins q][ins [del g true,s]]
[ins q,del g][ins [s]]
[insert q,delete g] <eos>
```

In this example, only marked goals q and `not g` are resolved while unmarked goal s is skipped. As mentioned on section 2, goals resolved to false can be inserted whereas goals resolved to true can be deleted, then those temporary results are added to the context (bold text enclosed in squared brackets) at the beginning of each line. Note that negation flips between deletion and insertion. Finally, the difference between `del` and `delete`, and `ins` and `insert` is to distinguish temporary reasoning during the derivation versus the final answer.

4. Results and Discussion

The results of the experiments are shown in Table 2. Also, we report the results of a post-hoc error analysis where three main types of errors were identified: (i) Syntactic error (e.g. incomplete sequences, misplaced tokens, etc.); (ii) Redundancy errors (e.g., the model generating unnecessary tokens); and (iii) Transcription errors (e.g., incorrectly flipping true/false and ins/del values during resolution, etc.) Some experiment results fall into multiple error types.

Exp.	syntactic			redundancy			transcription			Exp.	syntactic			redundancy			transcription		
	1K	10K	100K	1K	10K	100K	1K	10K	100K		1K	10K	100K	1K	10K	100K	1K	10K	100K
$\text{Conj}_{\text{explore}}$	66	45	0	60	38	0	50	13	0	$\text{Conj}_{\text{gpt3\_in}}$	97	–	–	84	–	–	4	–	–
$\text{Conj}_{\text{early}}$	0	0	0	100	100	0	0	0	0	$\text{Conj}_{\text{gpt3\_ex}}$	0	–	–	0	–	–	0	–	–
Conj_{ctx}	0	0	0	0	0	0	0	0	0	$\text{Disj}_{\text{explore}}$	73	69	56	47	35	32	56	14	17
Conj_{10g}	100	0	0	0	0	0	11	0	0	$\text{Disj}_{\text{early}}$	1	0	0	95	0	0	13	0	0
$\text{Conj}_{\text{intrpl}}$	100	100	100	1	1	1	16	19	14	$\text{Neg}_{\text{backtrack}}$	100	0	0	0	0	0	74	0	0
$\text{Conj}_{\text{extrpl}}$	99	100	100	0	0	0	7	0	0	$\text{View}_{\text{update}}$	99	99	0	0	1	0	55	41	0

Table 3
Per experiment error analysis expressed as a percentage of *incorrect* examples for the error category.

Considered together, experiments $\text{Conj}_{\text{explore}}$, $\text{Conj}_{\text{early}}$, and Conj_{ctx} show that the generation of specific training examples successfully trains the GPT-2 model to perform symbolic reasoning over conjunctive predicates. In particular, carrying along the context of resolution as it proceeds, greatly reduces the number of examples needed to fine-tune the model. The most common type of error in $\text{Conj}_{\text{explore}}$ is syntactic (55% avg.) followed by redundancy (48% avg.), whereas $\text{Conj}_{\text{early}}$ completely eliminates both syntactic and transcription errors (see Table 3).

Experiments Conj_{10g}, $\text{Conj}_{\text{intrpl}}$, and $\text{Conj}_{\text{extrpl}}$ explore the flexibility and brittleness of the transformer model. Comparing Conj_{10g} with $\text{Conj}_{\text{early}}$, the only difference is that the former dataset contains more goals, but the performance is nearly the same. One could interpret this result as the model is relatively insensitive to the number of goals, however, the error analysis reveals that the more goals to process the more syntactic errors and the less redundancy errors. After close examination, we realized that $\text{Conj}_{\text{early}}$ is more prone to redundancy errors when the last of the three rules is the one to be queried. Given that Conj_{10g} processes more rules, the probability of querying the last defined rule is much lower.

Contrasting $\text{Conj}_{\text{intrpl}}$ with Conj_{10g}, the results show that the transformer model cannot easily interpolate to resolution with less than 10 goals even with 100K examples, if a broader range of training data is not provided. Surprisingly, when comparing $\text{Conj}_{\text{extrpl}}$ with Conj_{10g}, the model performs better at extrapolating to a few more goals. Although syntactic errors are predominant in both $\text{Conj}_{\text{intrpl}}$ and $\text{Conj}_{\text{extrpl}}$, interpolation experiments show that the model mainly struggles with the correct insertion of the <eos> token and the placement of connectors.

The next two results, $\text{Conj}_{\text{gpt3\_in}}$ and $\text{Conj}_{\text{gpt3\_ex}}$ briefly explore the impact of GPT-3. Both models show much better performance than their GPT-2 counterparts, as is normally to be expected since scaling the number of parameters for transformer models generally leads to improved performance. For interpolation, note that GPT-3 (1K) performs better than GPT-2 (100K), which corresponds to a 32% improvement on performance using only a tiny fraction (1/100) of the amount of data used by GPT-2. On the other hand, $\text{Conj}_{\text{gpt3\_in}}$ achieved a perfect accuracy. These results demonstrate the capability of large pre-trained models such as GPT-3 to generalize an algorithm for symbolic reasoning using only a small amount of training data.

For disjunction, interpretation of $\text{Disj}_{\text{explore}}$ is a little tricky since the problem to solve is larger - i.e., there are more logical connectives involved. Because of the poor performance, clearly tracking disjunction state, even with the use of context, is a challenge. Our interpretation of the good performance of $\text{Disj}_{\text{early}}$ is due simply to the fact that most examples terminate early (67%), thus requiring the model to track less state, and because shorter (failed) proofs are probabilistically less likely to contain an error. While $\text{Disj}_{\text{explore}}$ contains errors of the three identified types, $\text{Disj}_{\text{early}}$ mostly produces redundancy errors (due to the tendency of the model of generating longer sequences after the early-termination step).

$\text{Neg}_{\text{backtrack}}$ provides evidence that a more complex logic can be learned with very little penalty. $\text{Neg}_{\text{backtrack}}$ mostly present syntactic and transcription errors (i.e., missing brackets and tokens during backtracking and incorrectly resolving nested negations, respectively).

Experiment $\text{View}_{\text{update}}$ demonstrates that simple brief revision reasoning is also possible within transformer models, although many additional examples are required. The error analysis reveals that the most common types of errors are syntactic errors (e.g., missing `ins` and `del` keywords from context and empty brackets at the end of the resolution – for instance, `[]` instead of `[insert q,delete g]`) and transcription errors (flipping `ins` to `del` and vice versa).

Finally, overall, the experiments demonstrate that a transformer model, which is fined-tuned on data that uses a proper representation, is able to learn and generalize the underlying inference algorithm to simulate propositional logic resolution regardless the length and level of complexity of the derivation proofs (see Appendix B for further details).

5. Related Work

Integration of symbolic and neural architectures is an active area of research. One strategy compiles symbolic reasoning into a neural network [17, 18, 19], where the network is constructed such that different predicates/rules can be semantically related to each other. While this approach requires that the construction of neural networks for each rule must be integrated with the application at the network level, in our work, reasoning is directly available at the model evaluation level. More generally, a classification of different systems focuses on the structure and reasoning of the underlying network [19]. For example, one category of the classification distinguishes symbolic from sub-symbolic systems. Transformer models are sub-symbolic since each lexical term is associated with an embedding. We use training data to essentially coerce the model into treating some sub-symbolic terms to behave as symbolic terms, i.e., propositions. Generically, the methodology of this paper injects symbolic reasoning by translating a logic program into training data for the transformer model. In effect, the lexical tokens of the logic program are blended with other lexical tokens provided in the training data, allowing for a mixing of different forms of reasoning. This representation trick is effectively a form of integration of symbolic language with inductive-bias [20]. Notably, our experiments show that transformers can effectively learn symbolic language-based representations and reasoning.

A neuro-symbolic approach that is similar in nature to ours, proposes an RNN-based iterative network that is trained end-to-end to perform reasoning over logic programs [21]. From the implementation perspective, we take advantage of transformers' capabilities over RNN networks, that is, transformers use positional embeddings and a self-attention mechanism, which allow our model to better capture the relationships between the different elements (symbols, logical connectives, and rules) of both the input and output sequences. This makes our model less prone to losing track during step-by-step resolution.

Logical Neural Networks [22] propose a model that performs logical reasoning by creating 1-to-1 mappings between neurons and the elements of logical formulae, where the weights of neurons are constrained to act as AND or OR gates. This work differs from ours mainly because it uses fixed-length representations that constrain the number of symbols/connectors to be used as well as the expressibility of the resolution. As we demonstrated, our transformer model can flexibly adapt to variable-length inputs (context and query) and outputs (resolution).

Another line of work proposes Logic-Integrated Neural Networks [23, 24], a framework that dynamically constructs its neural architecture based on an logical expression and then performs propositional reasoning. Although it supports compositionality, the computational cost exponentially increases as that number of expressions grow. In contrast, our model achieves high performance without resorting to restructuring the network architecture or training massive amounts of data (only fine-tuning was needed), resulting in much lower engineering costs.

Although somewhat tangential, [25] provide an experimental evaluation on how transformer models perform on mathematical reasoning. Despite this study did not cover logical reasoning, it highlights the capabilities of transformer models to interpret free-form questions, perform mathematical reasoning, and generate the correct answers. Unlike our approach, this work only consider direct responses, for instance, given the question: "Solve -42*r + 27*c = -1167 and 130*r + 4*c = 372 for r" the transformer model directly outputs "4" without intermediate steps. In contrast, our model uses step-by-step resolution, which can be further used for validation and, more importantly, for transparency, interpretability, and explainability purposes.

6. Conclusion and Future Work

In this paper we describe a method to train transformer models in the steps of propositional logic resolution through examples. We systematically explore different possible representations of propositional clauses as data and measure the performance impact. The end result is a reasoning system for conjunctions, disjunction, and negation. We then apply this system to computing view updates (a form of belief revision reasoning). For future work, one step is to extend the techniques described here to more powerful logic systems [26] - the principal extension is the addition of data that trains the steps involved in unification. Unification is a context-specific substitution of symbols, similar to the types of substitutions performed in this paper. Finally, we end this paper with a call for researchers to examine a neuro-symbolic reasoning system with respect to its expressive power to simulate other reasoning systems. As this paper shows, a (low-cost) simulation provides a powerful argument to prefer one system over another.

References

[1] T. Brown, B. Mann, N. Ryder, et al., Language models are few-shot learners, in: Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[2] J. W. Rae, S. Borgeaud, et al., Scaling language models: Methods, analysis & insights from training gopher, CoRR abs/2112.11446 (2021). URL: https://arxiv.org/abs/2112.11446. arXiv:2112.11446.

[3] D. Ham, J.-G. Lee, Y. Jang, et al., End-to-end neural pipeline for goal-oriented dialogue systems using GPT-2, in: Proceedings of the 58th ACL Conference, 2020, pp. 583–592. URL: https://aclanthology.org/2020.acl-main.54. doi:10.18653/v1/2020.acl-main.54.

[4] P. Budzianowski, I. Vulić, Hello, it's gpt-2-how can i help you? towards the use of

pretrained language models for task-oriented dialogue systems, in: Proceedings of the 3rd Workshop on Neural Generation and Translation, 2019, pp. 15–22.
[5] A. Santoro, A. Lampinen, K. Mathewson, T. Lillicrap, D. Raposo, Symbolic behaviour in artificial intelligence, arXiv preprint arXiv:2102.03406 (2021).
[6] A. d. Garcez, L. C. Lamb, Neurosymbolic ai: the 3rd wave, arXiv preprint arXiv:2012.05876 (2020).
[7] M. Garnelo, M. Shanahan, Reconciling deep learning with symbolic artificial intelligence: representing objects and relations, Current Opinion in Behavioral Sciences 29 (2019) 17–23.
[8] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, S. J. Gershman, Building machines that learn and think like people, Behavioral and brain sciences 40 (2017).
[9] G. Marcus, The next decade in ai: Four steps towards robust artificial intelligence, arXiv preprint arXiv:2002.06177 (2020).
[10] P. Gärdenfors, Belief Revision, Cambridge University Press, 2003.
[11] A. C. Kakas, P. Mancarella, Database updates through abduction., in: VLDB, volume 90, 1990, pp. 650–661.
[12] O. J. Romero, A. Wang, J. Zimmerman, A. Steinfeld, A. Tomasic, A task-oriented dialogue architecture via transformer neural language models and symbolic injection, in: SIGDIAL, 2021, pp. 438–444.
[13] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, Language models are unsupervised multitask learners, 2018. Https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.
[14] A. Tomasic, View update translation via deduction and annotation, in: International Conference on Database Theory, Springer, 1988, pp. 338–352.
[15] A. Tomasic, Determining correct view update translations via query containment, in: International Conference on Logic Programming Workshop on Deductive Databases, Stanford InfoLab, 1994, pp. 75–83. URL: http://ilpubs.stanford.edu:8090/53/.
[16] S. Welleck, I. Kulikov, J. Kim, et al., Consistency of a recurrent language model with respect to incomplete decoding, arXiv preprint arXiv:2002.02492 (2020).
[17] P. Minervini, M. Bošnjak, T. Rocktäschel, et al., Differentiable reasoning on large knowledge bases and natural language, in: Proceedings of the AAAI conference on artificial intelligence, volume 34, 2020, pp. 5182–5190.
[18] S. N. Tran, Compositional Neural Logic Programming, in: 30th International Joint Conference on Artificial Intelligence, 2021, pp. 3059–3066.
[19] L. De Raedt, S. Dumančić, R. Manhaeve, G. Marra, From Statistical Relational to Neural Symbolic Artificial Intelligence: a Survey, in: IJCAI-20, 2021, pp. 89–94.
[20] H. Geffner, Target languages (vs. inductive biases) for learning to act and plan, arXiv preprint arXiv:2109.07195 (2021).
[21] N. Cingillioglu, A. Russo, Deeplogic: Towards end-to-end differentiable logical reasoning, CEUR Workshop Proceedings, 2019. URL: http://hdl.handle.net/10044/1/71107.
[22] R. Riegel, A. G. Gray, F. P. S. Luus, et al., Logical neural networks, CoRR abs/2006.13155 (2020). URL: https://arxiv.org/abs/2006.13155. arXiv:2006.13155.
[23] S. Shi, H. Chen, W. Ma, J. Mao, M. Zhang, Y. Zhang, Neural Logic Reasoning, Association for Computing Machinery, New York, NY, USA, 2020, p. 1365–1374. URL: https://doi.org/

10.1145/3340531.3411949.
[24] S. Shi, H. Chen, M. Zhang, Y. Zhang, Neural logic networks, CoRR abs/1910.08629 (2019). URL: http://arxiv.org/abs/1910.08629. arXiv:1910.08629.
[25] D. Saxton, E. Grefenstette, F. Hill, P. Kohli, Analysing mathematical reasoning abilities of neural models, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 2019, pp. 127–132.
[26] W. Van Laer, L. De Raedt, How to upgrade propositional learners to first order logic: A case study, in: Relational data mining, Springer, 2001, pp. 235–261.

A. Dataset genereration using Prolog

The code snippet below denotes a Prolog logic program used for generating the examples. For simplicity, we only present the program for conjunction/disjunction generation.

```
facts([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).

% fact generation
mk_facts(0, Facts0, Facts0, []).
mk_facts(K, Facts0, Facts2, [NewRule0|NewRule1]) :-
    pick(Fact, Facts0, Facts1),
    mk_fact_rule(Fact, NewRule0),
    K1 is K-1,
    mk_facts(K1, Facts1, Facts2, NewRule1).

mk_fact_rule(Fact, (Fact :- Body)) :-
    random(X),
    (X > 0.7) -> Body = true; Body = false.

pick(X, L1, L2) :-
    length(L1, Length),
    Length1 is Length+1,
    random(1, Length1, N),
    nth(N, L1, X),
    delete(L1, X, L2).

prog0(KB, R) :- facts(F), mk_facts(12, F, R, KB).

% rule generation
mk_rules(0, _, _, R, R, []).
mk_rules(K, P, KB, R0, R2, [NewRule|NewKB]) :-
    pick(Head, R0, R1),
    mk_rule(P, Head, KB, NewRule),
    K1 is K-1,
    mk_rules(K1, P, KB, R1, R2, NewKB).

mk_rules_rand(0, _, R, R, []).
mk_rules_rand(K, KB, R0, R2, [NewRule|NewKB]) :-
    pick(Head, R0, R1),
    random(4, 6, RN),
    mk_rule(RN, Head, KB, NewRule),
    K1 is K-1,
    mk_rules_rand(K1, KB, R1, R2, NewKB).

mk_rule(P, Head, KB, (Head :- Body)) :-
    mk_rule_body(P, KB, Body).

mk_rule_body(1, KB, Term) :-
    pick((Term :- _), KB, _).
mk_rule_body(P, KB0, TermBody) :-
    pick((Term :- _), KB0, KB1),
    P1 is P-1,
    mk_rule_body(P1, KB1, Body),
    mk_rule_body_body(Term, Body, TermBody).

mk_rule_body_body(Term, Body, TermBody) :-
    random(X),
    (X < 0.5) -> TermBody = (Term, Body); TermBody = (Term; Body).

prog1(Query, KB, Rules) :-
    prog0(KB, R),
    mk_rules_rand(3, KB, R, _, Rules),
    pick((Query :- _), Rules, _).

%%% the meta-interpreter - mi - Thanks to Markus Triska (triska@metalevel.at)
clause_list((g(X)), [X]) :- X \= (_,_).
clause_list((g(H), T), [H|T1]) :- clause_list(T, T1).
clause_list((g(H); g(T)), [(H; T)]).

% mi_backtrace is the core prover
mi_defaulty_better(true, true).
mi_defaulty_better(false, false).
mi_defaulty_better((A,B), (BA,BB)) :-
        mi_defaulty_better(A, BA),
        mi_defaulty_better(B, BB).
mi_defaulty_better((A;B), (BA;BB)) :-
        mi_defaulty_better(A, BA),
        mi_defaulty_better(B, BB).
mi_defaulty_better(G, g(G)) :-
        G \= false,
        G \= true,
        G \= (_,_),
        G \= (_;_).

mi_clause(Goal, KB, Body0, Body) :-
        member((Goal :- Body0), KB),
        mi_defaulty_better(Body0, Body).

mi_tree(true, _, [true]).
mi_tree(false, _, [false]).
mi_tree((A,B), KB, T2) :-
        mi_tree(A, KB, T0),
        mi_tree(B, KB, T1),
        append(T0, T1, T2).
mi_tree((A;B), KB, T2) :-
        mi_tree(A, KB, T0),
        mi_tree(B, KB, T1),
        T2 = [(T0; T1)].
mi_tree(g(G), KB, T1) :-
        mi_clause(G, KB, Body, G_Body),
        mi_tree(G_Body, KB, T0),
        append([(G:-Body)], T0, T1).

new_conjunct_env(B, AEnv, ANewEnv) :-
    clause_list(B, BList),
    new_conjunct_env_(BList, AEnv, ANewEnv).
new_conjunct_env_(_, [], []).
new_conjunct_env_(BList, [H|T0], [ABList|T1]) :-
    append(H, BList, ABList),
    new_conjunct_env_(BList, T0, T1).

mi_backtrace(G, KB, Trail) :-
         nonvar(G), mi_tree(g(G), KB, Trail).

prog2(Q, NewKB, Trail) :- prog1(Q, KB, Rules), append(KB, Rules, NewKB),
 ↪  mi_backtrace(Q, NewKB, Trail).

strip_g(g(X),X).

dump_kb(_, []).
dump_kb(File, [H|T]) :-
    write_ds(File, H),
    dump_kb(File, T).

dump_query(File, Query) :-
    write_ds(File, Query).

dump_proof(File, []).
dump_proof(File, [[false|T0]|T]) :-
    write_ds(File, [false|T0]).
dump_proof(File, [H0|T]) :-
    write_ds(File, H0),
    dump_proof(File, T).

write_ds(File, Text) :-
    open(File, append, Stream),
    write(Stream, Text),  nl(Stream),
    close(Stream).

prog3 :- prog3(5).
prog3(K) :- prog3('data/disjunction.txt', K).
prog3(_, 0).
prog3(File, K) :-
    prog2(Query, KB, Trail),
    write_ds(File, '<bos>'),
    write_ds(File, KB),
    write_ds(File, '<soq>'),
    write_ds(File, Query),
    write_ds(File, '<sop>'),
    dump_pair(File, KB, Trail),
    write_ds(File, '<eos>'),
    K1 is K-1,
    prog3(File, K1).

dump_pair(_, _, []).
dump_pair(File, KB, [H|T]) :-
    write_ds(File, KB),
    write_ds(File, H),
    dump_pair(File, KB, T).
```

B. Generated Propositional Resolution for View-Updates Reasoning

Below we provide some propositional proofs for the view-updates reasoning experiment generated by the transformer model, which illustrates its capability to generalize the underlying inference algorithm that generates sequences of resolution steps with different lengths and different levels of complexity.

B.1. Proof 1

```
<bos>
(i :- true)
(f :- false)
(l :- false)
(u :- true)
(q :- false)
(a :- true)
(z :- false)
(m :- true)
(c :- true)
(n :- false)
(s :- false)
(b :- true)
(h :- n;f,m)
(j :- b,ins c,s)
(w :- not m,ins c,i)
<soq> ins w
<sop>
[ins [(w :- not m,ins c,i)]]
[ins [not m,ins c,i]]
[ins [ins c,i]]
[ins [ins (c :- true),i]]
[ins [ins c true,i]]
[ins [i]]
[]
<eos>
```

B.2. Proof 2

```
<bos>
(y :- false)
(o :- false)
(u :- true)
(p :- true)
(a :- true)
(x :- true)
(n :- true)
(k :- false)
(w :- true)
```

```
(m :- false)
(s :- false)
(l :- false)
(c :- ins y,(ins l;ins not p))
(d :- u;w,s)
(b :- s,(o;k))
<soq> ins c
<sop>
[ins [(c :- ins y,(ins l;ins not p))]]
[ins [ins y,(ins l;ins not p)]]
[ins [ins (y :- false),(ins l;ins not p)]]
[ins [ins y false,(ins l;ins not p)]]
[[ins y][ins [ins l;ins not p]]
[ins y][ins [ins l]]
[ins y][ins [ins (l :- false)]]
[ins y][ins [ins l false]]]
[ins y,ins l][ins [ins not p]]
[ins y,ins l][ins [del p]]
[ins y,ins l][ins [del (p :- true)]]
[ins y,ins l][ins [del p true]]
[ins y,ins l,del p]
[insert y,insert l,delete p]
<eos>
```