Conference Paper Title*

1st Federico Raspanti
Eindhoven University of Technology
Université Côte d'Azur
City, Country
f.raspanti@student.tue.nl

***Abstract*—**

***Index Terms*—**

I. Introduction

context

LLMs have shown remarkable capabilities for logical reasoning, especially when guided with prompting techniques such as chain-of-thought and few-shot examples

motivation

The autoregressive nature of LLMs prevents them from reasoning in a determistic and truly logical fashion, making them unreliable when it comes to logical reasoning task.

problem

Given a natural language reasoning problem
$P_{NL} = (R_{NL}, Q_{NL})$.
$R_{NL} =$ list of rules in natural language
$Q_{NL} =$ statement to be proven

We aim to extract
$P_{LOG} = (R_{LOG}, Q_{LOG})$
$R_{LOG} =$ list of logical rules
$Q_{LOG} =$ logical statement to be proven

research questions

- **R1**: Does the *accuracy* (F1 score) of the generated logical problems improve when providing *embedding-wise* similar few-shot examples?
- **R2**: By using Grammar Constrained Decoding (GCD) to guarantee *validity*, how much does the LLM's response degrade?

contributions

- we show the impact of semantic similarity in NL-Logic conversion.
- we provide a new approach to self-correcting of the LLM's mistakes by dynamicall generating grammar constraints

II. Related Work

A. Logic-LM

B. LoGiPT

C. LLM-R

III. Preliminaries

A. Grammar-Constrained Decoding

B. In-context Learning

C. Chain-of-thought Reasoning

IV. Methodology

A. Retrieving Relevant Examples

For each test problem, we retrieve the most similar problems from our datasets' training sets by ranking them according to *cosine score* of their embeddings.

B. Sketching

We use a powerful, black-box LLM to convert the NL problems to Logical Problems.

C. Grammar-Constrained Formulation

We use an open-source model with logit access to correct all *invalid* logical problems generated during Sketching

D. Solving the Problem

We use a symbolic solver to solve the generated logical problem.
First-Order-Logic: We use Prover9
Logic Programming: We use pyke

V. Experiments

A. Datasets

FOLIO:
PrOntoQA:
ProofWriter:

B. Baselines

C. Metrics

D. Implementaion Details

VI. Results

A. Main Results

B. Further Analysis

1) Impact of dynamic example retrieval:

Dataset	Accuracy (F1)	Formula Validity
Logic-LM	59.25—54.67	
+Refinement	58.69—58.57	
DynoReasoner	63.12—62.93	
+Refinement	62.16	
+Constrained Generation	63.55	
+Refinement & Constrained Generation		

TABLE I
ACCURACY OF EXECUTABLE SAMPLES (F1)

Dataset	Logic-LM	+ Refinement	+ Dynamic Examples	+ Both
FOLIOv2	61.57%	60.59%	**64.03%**	63.54%
PrOntoQA				
ProofWriter				

TABLE II
ACCURACY (F1) WITH FEW-SHOT COT BACKUP IF SAMPLES ARE NON-EXECUTABLE

Dataset	Direct Few-shot	CoT Few-shot
FOLIOv2	47.05%—42.85—41.37	66.17%—64.61%—66.12%—67.27%
PrOntoQA		
ProofWriter		

TABLE III
ACCURACY (F1) OF BACKUP ON NON-EXECUTABLE SAMPLES

Dataset	Direct Few-shot	CoT Few-shot
FOLIOv2	47.05%—42.85—41.37	66.17%—64.61%—66.12%—67.27%
PrOntoQA		
ProofWriter		

TABLE IV
ACCURACY (F1) OF BACKUP ON NON-EXECUTABLE SAMPLES

2) Impact of Constrained decoding:

3) Impact of self-verification loop:

VII. CONCLUSION AND FUTURE WORK

ACKNOWLEDGMENT

APPENDIX

A. Grammars

B. Prompts

C. Formulations

Dataset	Logic-LM	+ Refinement	+ Dynamic Examples	+ Both
FOLIOv2	66.50%—68.47%	67.98%—68.96%	69.45%—70.44%	**72.90%**
PrOntoQA				
ProofWriter				

TABLE V
FULLY EXECUTABLE SAMPLES RATE - SAMPLES THAT CAN BE BOTH PARSED AND EXECUTED (%)

Dataset	Logic-LM	+ Refinement	+ Dynamic Examples	+ Both
FOLIOv2	**13.79%**	13.79%—14.28%	15.27%—15.76%	14.77%
PrOntoQA				
ProofWriter				

TABLE VI
PARSING ERRORS RATE (%)

Dataset	Logic-LM	+ Refinement	+ Dynamic Examples	+ Both
FOLIOv2	19.70%	18.22%—16.74%	15.27%—12.80%	**12.31%**
PrOntoQA				
ProofWriter				

TABLE VII
EXECUTION ERRORS RATE (%)