LTRAG: Enhancing Autoformalization and Self-refinement for Logical Reasoning with Thought-Guided RAG

Ruikang Hu, Shaoyu Lin, Yeliang Xiu, Yongmei Liu[*]
Dept. of Computer Science, Sun Yat-sen University, Guangzhou 510006, China
{hurk3,linshy57,xiuyliang}@mail2.sysu.edu.cn, ymliu@mail.sysu.edu.cn

Abstract

Logical reasoning is fundamental to intelligent systems. Large language models (LLMs) have demonstrated promise in natural language (NL) reasoning, especially with techniques like chain-of-thought (CoT) prompting. Neuro-symbolic methods like Logic-LM and LINC further enhance performance on challenging datasets FOLIO and AR-LSAT by integrating formalization with LLMs and symbolic solvers, and possibly refinement with LLMs. However, these methods still struggle with the accurate formalization of complex NL problems. In this paper, we introduce LTRAG, a framework to enhance autoformalization and self-refinement for logical reasoning with Retrieval-Augmented Generation (RAG), by building knowledge bases of thought-guided examples[1]. Experimental results on FOLIO and AR-LSAT show that LTRAG consistently outperforms Logic-LM and LINC across different models. On GPT-4 and AR-LSAT, it achieves an accuracy gain of 13% over Logic-LM.

1 Introduction

Logical reasoning is a fundamental aspect of human intelligence and is essential for complex tasks such as problem solving. Recently, logical reasoning over natural language (NL) exploiting large language models (LLMs) has received much attention. Many datasets have been proposed, including synthetic ProofWriter for rule reasoning (Tafjord et al., 2021), human-crafted FOLIO for complex first-order logic (FOL) reasoning (Han et al., 2024), and AR-LSAT extracted from LSAT exams (Zhong et al., 2021). Various methods have been explored, including prompting (Wei et al., 2022; Kojima et al., 2022), fine-tuning (Zelikman et al., 2022), and neuro-symbolic methods based on search (Kazemi et al., 2023; Hao et al., 2023).

A recent endeavor for logical reasoning over NL is neuro-symbolic methods based on autoformalization, by combining translation with LLMs from NLs to formal languages and rigorous reasoning of symbolic solvers. As long as the translations are correct, resorting to symbolic solvers can guarantee faithfulness of reasoning, which cannot be ensured by reasoning in LLMs, because of their fundamental nature of black-box probabilistic models. Typical works are Logic-LM (Pan et al., 2023) which introduces self-refinement to use the symbolic solver's error messages to refine the formalization, and LINC (Olausson et al., 2023), which uses majority-vote to decide the result from multiple formulations. On GPT-4, LINC achieves an accuracy of 98% on ProofWriter, and Logic-LM reaches 79% on FOLIO.

However, research on autoformalization still faces significant challenges in guaranteeing correctness of translation, especially for more complex NL inputs. For example, with GPT-4, on AR-LSAT, Logic-LM only achieves an accuracy of 43%. In particular, these methods rely on a fixed set of examples for autoformalization and self-refinement, thus struggling with diverse and complex inputs, limiting their generalizability. On the other hand, Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) enhances generation by dynamically retrieving relevant information from an external knowledge base (KB).

In this paper, we propose a framework LTRAG to enhance autoformalization and self-refinement for logical reasoning with thought-guided RAG, by building KBs of thought-guided examples for formalization and refinement. Experimental results on FOLIO and AR-LSAT show that LTRAG consistently outperforms Logic-LM and LINC across different models. In particular, LTRAG achieves an accuracy gain of 8% over Logic-LM and LINC on GPT-3.5-turbo and FOLIO, and 13% on GPT-4 and AR-LSAT.

[*]Corresponding author
[1]https://github.com/sysulic/LTRAG

2 Related Work

Translating a NL into a formal language is challenging due to its ambiguity and implicit information. Nguyen et al. (2022) proposed a method combining manual translation with deep learning. Yang et al. (2024) introduced LogicLLaMA, combining supervised fine-tuning and reinforcement learning with human feedback. Chen et al. (2023) developed a framework using LLMs for translation between NL and temporal logic using intermediate languages.

Autoformalization-based neuro-symbolic methods for logical reasoning have recently gained much attention. Pan et al. (2023) proposed Logic-LM, including a **Problem Formulator** to formalize the problem, a **Symbolic Reasoner**, and a **Self-Refiner** module for error correction. Olausson et al. (2023) introduced LINC, which also combines autoformalization and symbolic solving, but uses majority voting to aggregate results. Ye et al. (2023b) proposed SATLM, which uses LLMs to formalize NL input into satisfiability (SAT) problems and uses a SAT solver for reasoning. Jiang et al. (2024) introduced LeanReasoner, which fine-tunes with data in the Lean theorem-proving environment, formalizes problems into Lean theorems and solves them with a tactic generator and proof search. Xu et al. (2024) proposed SymbCoT, which does autoformalization but reasons with CoT prompting based on both NL and FOL inputs.

Lewis et al. (2020) proposed RAG, using document retrieval to improve output precision with external knowledge. Fan et al. (2024) showed RAG reduces hallucinations and improves generation quality. Jiang et al. (2023) introduced FLARE, enabling efficient retrieval during generation.

Example selection is key to in-context learning. Liu et al. (2022) used a sentence encoder to select top-k similar examples for given problems, showing dynamic selection improves LLM performance. Levy et al. (2023) studied compositional generalization in semantic parsing, selecting diverse examples via coverage and diversity based methods. Ye et al. (2023a) proposes CEIL, an example selection method using Determinantal Point Processes and contrastive learning.

3 Framework

The structure of LTRAG is depicted in Figure 1. It comprises four key components: a Retrieval Module, a Translator LLM, a Solver, and a Fixer LLM. The Translator LLM (similar to Logic-LM's Problem Formulator) converts NL problems into formal representations, while the Fixer LLM (akin to Logic-LM's Self-Refine) corrects translation errors. The Retrieval Module, built upon FastGPT[2], dynamically retrieves similar examples from the RAG KBs. Specifically, we store the embeddings of examples from the KBs as vectors. When a new task of translation or fixing is introduced, its text is embedded into a vector of the same dimension. We then compute cosine similarity between the task vector and all vectors in the KB, selecting the top-k most similar examples to provide context for the LLMs. These retrieved examples serve as guiding references for both Translator and Fixer LLMs, enhancing the accuracy of formalization and error correction. The detailed method for constructing the KBs can be found in Section 4.2. Once the problem is formalized, the Solver takes over to perform logical reasoning. The Solver is based on Microsoft's Z3 solver[3] (de Moura and Bjørner, 2008), and is capable of handling FOL expressions (in FOLIO) and constraint satisfaction problems (in AR-LSAT). If errors are detected, they are reported to the Fixer LLM for another formalization, and the above process is repeated.

Here is an example (full version in Appendix A.1): One of the input premises is "There are four seasons in a year: Spring, Summer, Fall, and Winter". The full problem is used to retrieve the translation KB, and both the problem and the retrieved examples are provided to the Translator LLM, resulting in an initial formalization:

$$\forall x(Season(x) \to (x = Spring \lor \\ x = Summer \lor x = Fall \lor x = Winter)).$$

The solver will return an error, indicating that using "=" is not allowed. The error is used to retrieve the fixer KB, and both the error and the retrieved examples are provided to the Fixer LLM, resulting in the final correct formalization:

$$\forall x(Season(x) \to (IsSpring(x) \lor \\ IsSummer(x) \lor IsFall(x) \lor IsWinter(x))).$$

4 Experiments

4.1 Experimental Setup

We evaluate LTRAG on FOLIO and AR-LSAT, comparing it against baselines such as Standard prompting, CoT prompting (using 2 examples on

[2]https://github.com/labring/FastGPT

[3]https://github.com/Z3Prover/z3

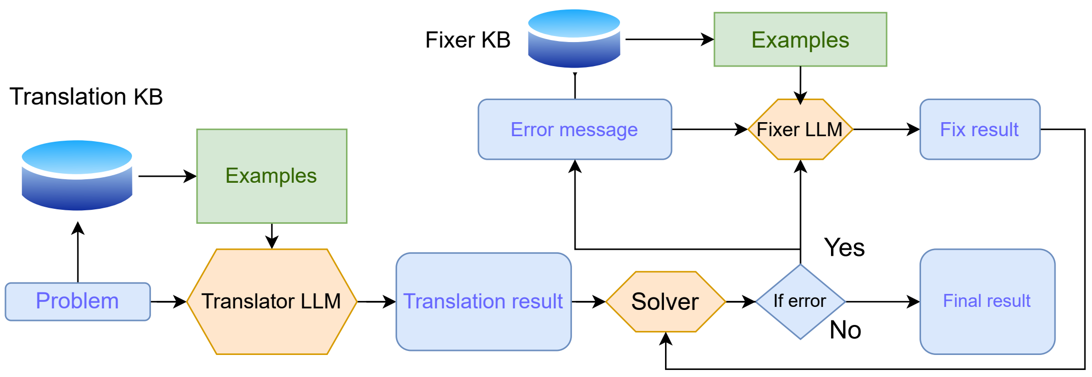

Figure 1: The framework of LTRAG.

FOLIO and 1 on AR-LSAT), LINC, and Logic-LM, across models including GPT-4o (OpenAI, 2024), DeepSeek v2.5 (DeepSeek-AI, 2024), Llama3.3-70b (Grattafiori et al., 2024), GPT-3.5-turbo, and Gemma2-27b (Rivière et al., 2024).

On FOLIO, as our test set, we use the 182 verified samples retained by LINC after filtering out the problematic ones from the original 204 samples. On AR-LSAT, as Logic-LM, we use the 231 samples from the dev set as our test set.

Note that AR-LSAT samples are single-choice questions. Even when the program generated for a sample is executable, the solver may return no or multiple answers. This is a type of semantic errors. When errors persist after repeated repairs, Logic-LM returns an answer using the CoT method, and we do the same.

4.2 Knowledge Base Construction

Our RAG KBs are semi-automatically constructed. Roughly, the translation KBs are automatically constructed from the training sets with programs or LLMs, while the fixer KBs are constructed by first manually fixing a small set of error cases and then using it to guide LLMs to generate the rest. Detailed examples are in Appendix A.2. In the following, we give details of our KB construction.

For FOLIO, the translation KB is built from the training set, with the 122 samples where DeepSeek fails to obtain the correct answer using the Translator LLM with an empty KB and the solver. We focus on errors made by powerful models like DeepSeek, as these are often more representative and challenging. Each such sample is provided with the annotated formulas and translation steps obtained with a program as follows: first, extract all predicates and constants from the problem; second, identify relevant predicates and constants for each sentence; and finally, translate the sentence into FOL. The fixer KB is built from the training set, using samples where DeepSeek fails to obtain an executable formalization. A subset is manually corrected for high-quality references, while the rest are automatically repaired as follows: for each such sample, using the annotated formulas and the manual repair subset as context, let DeepSeek return the thought process for repairing, and retain the sample if the solver returns the correct answer. The fixer KB ends up with 57 samples.

For AR-LSAT, the translation KB is built as follows: first, we pick 1,500 samples from the training set; second, for each of these samples, we use DeepSeek to formalize it with the RAG KB of Logic-LM's formalizations of the dev set; finally, we retain 538 executable samples. As to the second step, we focus on defining the formalization's syntax and precautions in the prompts; due to the length of AR-LSAT samples, we choose not to let LLMs generate the thought process because long outputs confuse LLMs and easily generate unexecutable formalizations. The fixer KB is built from the training set, using samples where DeepSeek fails to obtain an executable formalization or a unique answer. We first manually analyze the types of errors reported by the solver (e.g., syntax errors, multiple answers). For each error type, we craft high-quality examples demonstrating the error and its correction. The rest samples are semi-automatically repaired as follows: for each such sample, using the manual repair subset, let DeepSeek return the thought process for repairing the formulation, which is manually checked for correctness and decided for being retained or not. The fixer KB ends up with 41 samples.

Model	FOLIO (Accuracy %)					AR-LSAT (Accuracy %)			
	LTRAG	Standard	CoT	LINC	Logic-LM	LTRAG	Standard	CoT	Logic-LM
GPT-4o	**80.77**	73.63	78.02	72.50	78.92	**56.71**	40.26	43.72	43.04
DeepSeek v2.5	**78.57**	74.73	76.37	-	-	**68.40**	51.52	64.50	-
Llama3.3	**78.57**	72.53	71.43	-	-	**59.31**	40.26	39.83	-
GPT-3.5-turbo	**70.88**	56.59	59.34	62.60	61.27	**26.84**	24.24	19.48	26.41
Gemma2	**79.67**	59.89	62.09	-	-	**35.06**	25.97	24.67	-

Table 1: Performance comparison on FOLIO and AR-LSAT. The data for Logic-LM and LINC comes from their papers, and '-' denotes that they did not experiment on the model. LINC did not experiment on AR-LSAT. Best results in each row are bolded.

4.3 Results

The experimental results are summarized in Table 1. To handle longer contexts, we use GPT-4o, which has a larger context window (128K tokens) compared to GPT-4 (8K tokens) used in prior works. The two models show minimal performance differences on reasoning tasks.

On FOLIO, LTRAG consistently outperforms other methods across different models. For GPT-4o, it achieves 80.77%, surpassing Logic-LM's 78.92% and LINC's 72.50%. For GPT-3.5-turbo, LTRAG attains 70.88%, significantly outperforming Logic-LM (61.27%) and LINC (62.60%).

Model	Exe_rate	Exe_accuracy
GPT-4o	69.3	50.0
GPT-4(Logic-LM)	39.8	58.8
GPT-3.5-turbo	54.1	19.2
GPT-3.5(Logic-LM)	21.8	**60.3**
DeepSeek v2.5	**71.0**	44.5
Llama3.3	66.7	52.6
Gemma2	45.0	36.5

Table 2: Executable rate (Exe_rate) and Execution Accuracy (Exe_accuracy) on AR-LSAT.

On AR-LSAT, LTRAG also consistently improves over the baseline methods. For GPT-4o, it outperforms both Logic-LM and CoT by about 13%. LTRAG also enhances DeepSeek v2.5's performance, achieving 68.40% compared to 51.52% under Standard prompting. However, LTRAG attains limited improvements on GPT-3.5-turbo, with the performance gain being less than 3% compared to Standard Prompting and Logic-LM.

In Table 2, we analyze the executable rate and execution accuracy of LTRAG on AR-LSAT, in comparison to Logic-LM. In terms of executable rate, on both GPT-4o and GPT-3.5-turbo, LTRAG outperforms Logic-LM by about 30%, indicating its superior ability to generate executable programs. It is easy to notice that our execution accuracy is lower compared to Logic-LM. A possible reason is that we get much more executable programs, making the error rate in execution increase.

4.4 Ablation Experiments

In ablation studies, we investigate the effect of the Fixer LLM, i.e., we compare the performance with and without the Fixer LLM.

We test the performance with different temperatures ranging from 0.1 to 0.3 and different numbers of in-context examples. We test with 1, 2, and 3 examples for each model on FOLIO; and on AR-LSAT, with 3, 5, and 7 examples during the translation phase, and only one example during the repair phase due to the length of the examples. We show the results of each model at the optimal temperature for the translation process. For the repair process, we show the results at the optimal temperature, where the input is the result of the translation process at the optimal temperature for the optimal number of examples.

Model	w/o Fix			with Fix		
	E=1	E=2	E=3	E=1	E=2	E=3
DeepSeek V2.5	**76.9**	75.3	75.8	**78.6**	78.0	78.0
GPT-4o	74.7	74.7	**75.8**	**80.8**	**80.8**	**80.8**
Llama3.3	**74.2**	73.1	**74.2**	**78.0**	**78.0**	**78.0**
GPT-3.5-turbo	60.9	**64.3**	63.7	69.8	**70.3**	69.8
Gemma2	73.1	**76.4**	75.3	78.6	**79.1**	78.0

Table 3: Accuracy (%) comparison on FOLIO – Exact Match. E represents the number of in-context examples.

For FOLIO, we use two evaluation settings: **Exact Match**, and **Error as Unknown**. Exact Match considers correct only if the predicted label matches the ground truth label. In contrast, Error as Unknown treats grammatical errors as "Unknown" results, which may allow some outputs to be counted as correct by chance. Tables 3 and 4 present the accuracy comparison with and without

Model	w/o Fix			with Fix		
	E=1	E=2	E=3	E=1	E=2	E=3
DeepSeek V2.5	**78.6**	78.0	**78.6**	**78.6**	**78.6**	**78.6**
GPT-4o	75.8	77.5	**78.0**	**80.8**	**80.8**	**80.8**
Llama3.3	75.8	75.8	**78.0**	**78.6**	**78.6**	**78.6**
GPT-3.5-turbo	65.4	**68.7**	68.1	**70.9**	**70.9**	**70.9**
Gemma2	75.3	**78.6**	78.0	**79.7**	**79.7**	**79.7**

Table 4: Accuracy (%) comparison on FOLIO – Error as Unknown.

Fix in the two settings, respectively. The "w/o Fix" parts show that increasing the number of examples slightly improves accuracy, while in others it introduces noise. Most importantly, the two tables show that the Fixer LLM improves accuracy by 2–5% for large models, while the improvement is 3–6% for small models. The improvements are particularly notable in the exact match setting, where Fixer LLM corrects superficial grammatical errors.

For AR-LSAT, Table 5 presents the accuracy comparison with and without Fix. We observe that in the "w/o Fix" part, large models achieve their highest accuracy around 17%, with GPT-4o performing best at 19.5%, while small models reach a maximum accuracy of only 8.2%. The table shows Fixer LLM improves accuracy by 15–20% for large models, while the improvement is 2–8% for small models. The larger improvements on AR-LSAT compared to FOLIO can be attributed to the higher complexity and error rate in AR-LSAT.

Model	w/o Fix			with Fix
	E=1	E=2	E=3	E=1
DeepSeek V2.5	12.1	**16.0**	13.9	**31.6**
GPT-4o	**19.5**	15.6	16.0	**34.6**
Llama3.3	16.0	**16.9**	14.7	**35.1**
GPT-3.5-turbo	6.10	**8.20**	6.10	**10.4**
Gemma2	5.20	**7.80**	3.00	**16.5**

Table 5: Accuracy (%) comparison on AR-LSAT.

4.5 Discussion

We first analyze on FOLIO, why LTRAG achieves better improvements on small models than on large models. We think the thought-guided examples for the Translator LLM notably benefit small models by mitigating their inherent limitations. Large models produce fewer errors, and small models have limited repair capabilities, leading to limited improvements by the Fixer LLM.

We then analyze on AR-LSAT, why LTRAG achieves better improvements on large models than on small models. AR-LSAT samples are primarily constraint satisfaction problems, having unique answers, making it difficult to provide a systematic translation approach. The samples also involve complex long-text constraints, thus limiting the number of reference examples given to the Fixer LLM. As a result, while both large and small models face difficulties, large models can make effective corrections with limited assistance, whereas small models cannot.

5 Conclusion

In this paper, we propose the LTRAG framework to enhance autoformalization and self-refinement for logical reasoning with thought-guided RAG. The translation KBs are automatically constructed, and the fixer KBs are semi-automatically constructed where a small set of error cases are manually fixed and used to guide LLMs to generate more repairing examples. Empirical results on the challenging datasets FOLIO and AR-LSAT demonstrate that our approach significantly improves refinement capabilities of large models and formalization capabilities of small models. An outstanding advantage of our work is to improve formalization with less computational resources than approaches based on fine-tuning. In the future, we are interested in improving our framework by enhancing retrieval mechanisms, prioritizing error additions to the fixer KBs, handling more types of semantic errors, and more automated KB construction.

Acknowledgements

We acknowledge support from the Natural Science Foundation of China under Grant No. 62476295.

Limitations

Below, we outline some of the key challenges and constraints associated with our framework:

Difficulty in Constructing Thought Processes for Certain Tasks While structured reasoning steps can be effectively constructed for datasets like FOLIO with short context, other tasks such as AR-LSAT present challenges. AR-LSAT problems often involve complex constraints and relationships that are harder to break down into a thought process. This makes it difficult to provide the same level of guidance for small models, limiting their performance improvements.

Limited Impact of the Fixer LLM on Tasks with Few Syntax Errors The Fixer LLM, which corrects errors flagged by the Solver, shows limited improvement on tasks where syntax errors are rare, such as FOLIO. This is particularly true for large models like GPT-4o, DeepSeek, and Llama, which already produce fewer syntax errors due to their advanced reasoning capabilities. As a result, the Fixer LLM's contributions are marginal in such cases, and the primary benefits of LTRAG come from the structured formalization process. Conversely, the Fixer LLM proves more effective on complex tasks like AR-LSAT, where the error types are more varied. Large models, with their superior refinement capabilities, can leverage the Fixer LLM to achieve significant improvements. However, small models, that struggle with both autoformalization and self-refinement, gain less benefit from the Fixer LLM in these scenarios.

Limitations in the Fixer LLM on Semantic Errors The Fixer LLM is primarily designed to address surface-level syntax errors, such as incorrect predicate usage or invalid logical operators. It is not capable of resolving deeper semantic errors, where the logical formalization may be syntactically correct but semantically flawed. On FOLIO, where the solver provides unique answers, it's hard to detect semantic errors, while AR-LSAT provides extra feedback when the solver returns no or multiple answers. This limitation highlights the need for more advanced mechanisms that can handle both syntactic and semantic errors.

Dependency on Knowledge Base Quality The performance of LTRAG heavily relies on the quality and comprehensiveness of the KBs. In cases where the KB lacks sufficient examples or contains inaccuracies, the system's abilities may be compromised.

References

Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. 2023. NL2TL: transforming natural languages to temporal logics using large language models. In *EMNLP 2023, Singapore, December 6-10, 2023*, pages 15880–15903.

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: an efficient SMT solver. In *TACAS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Wenqi Fan, Ding, and et al. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pages 6491—6501.

Aaron Grattafiori, Abhimanyu Dubey, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Simeng Han, Hailey Schoelkopf, and et al. 2024. FOLIO: natural language reasoning with first-order logic. In *EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 22017–22031.

Shibo Hao, Yi Gu, and et al. 2023. Reasoning with language model is planning with world model. In *EMNLP 2023, Singapore, December 6-10, 2023*, pages 8154–8173.

Dongwei Jiang, Marcio Fonseca, and Shay B. Cohen. 2024. Leanreasoner: Boosting complex logical reasoning with lean. In *NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 7497–7510.

Zhengbao Jiang, Frank F. Xu, and et al. 2023. Active retrieval augmented generation. In *EMNLP 2023, Singapore, December 6-10, 2023*, pages 7969–7992. Association for Computational Linguistics.

Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. 2023. LAMBADA: backward chaining for automated reasoning in natural language. In *ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 6547–6568.

Takeshi Kojima, Shixiang Shane Gu, and et al. 2022. Large language models are zero-shot reasoners. In *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Itay Levy, Ben Bogin, and Jonathan Berant. 2023. Diverse demonstrations improve in-context compositional generalization. pages 1401–1422.

Patrick S. H. Lewis, Ethan Perez, and et al. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS 2020, December 6-12, 2020, virtual*.

Jiachang Liu, Dinghan Shen, and et al. 2022. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114.

Ha-Thanh Nguyen, Wachara Fungwacharakorn, Fumihito Nishino, and Ken Satoh. 2022. A multi-step approach in translating natural language into logical formula. In *JURIX 2022: The Thirty-fifth Annual Conference, Saarbrücken, Germany, 14-16 December 2022*, volume 362 of *Frontiers in Artificial Intelligence and Applications*, pages 103–112. IOS Press.

Theo Olausson, Alex Gu, and et al. 2023. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *EMNLP 2023, Singapore, December 6-10, 2023*, pages 5153–5176.

OpenAI. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 3806–3824.

Morgane Rivière, Shreya Pathak, and et al. 2024. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 3621–3634.

Jason Wei, Xuezhi Wang, and et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Jundong Xu, Hao Fei, and et al. 2024. Faithful logical reasoning via symbolic chain-of-thought. In *ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13326–13365.

Yuan Yang, Siheng Xiong, and et al. 2024. Harnessing the power of large language models for natural language to first-order logic translation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6942–6959, Bangkok, Thailand. Association for Computational Linguistics.

Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2023a. Compositional exemplars for in-context learning. In *ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 39818–39833. PMLR.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023b. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Wanjun Zhong, Siyuan Wang, and et al. 2021. AR-LSAT: investigating analytical reasoning of text. *CoRR*, abs/2104.06598.

A Appendix

A.1 An example of solving FOLIO problems

```
Premises:
1.[BG] There are four seasons in a
    ↪ year: Spring, Summer, Fall, and
    ↪ Winter.
2.All students who want to have a long
    ↪ vacation love summer the most.
3.Emma's favorite season is summer.
4.Mia's favorite season is not the same
    ↪ as Emma's.
5.James wants to have a long vacation.

Conclusion:
James's favorite season is summer.
```

Listing 1: Problem

Premises:
1. $\forall x(Season(x) \rightarrow (x = Spring \lor$
$x = Summer \lor x = Fall \lor x = Winter))$
2. $\forall x(WantsLongVacation(x) \rightarrow$
$FavoriteSeason(x, Summer))$
3. $FavoriteSeason(Emma, Summer)$
4. $\neg(FavoriteSeason(Mia, Emma) \land$
$FavoriteSeason(Emma, Mia))$
5. $WantsLongVacation(James)$

Conclusion:
$FavoriteSeason(James, Summer)$

Listing 2: Translated Formulas

The solver will return an error, indicating that using "=" is not allowed on premise 1. Instead, predicates should be used to represent relationships.

Premises:
1. $\forall x(Season(x) \rightarrow (IsSpring(x) \lor$
$IsSummer(x) \lor IsFall(x) \lor IsWinter(x)))$
2. $\forall x(WantsLongVacation(x) \rightarrow$
$FavoriteSeason(x, Summer))$
3. $FavoriteSeason(Emma, Summer)$
4. $\neg(FavoriteSeason(Mia, Emma) \land$
$FavoriteSeason(Emma, Mia))$
5. $WantsLongVacation(James)$

Conclusion:
$FavoriteSeason(James, Summer)$

Listing 3: Fixed Formulas

A.2 Examples from the RAG KBs

A.2.1 FOLIO

```
Input:
Premises:
1.All pets are animals.
2.Pets can be either a dog or a cat.
3.If a person has a pet, they care for
    ↪ that pet.
4.Dogs and Cats can be naughty.
5.Pets who are naughty are not liked as
    ↪ much.
6.Charlie has a naughty pet dog named
    ↪ Leo.

Conclusion:
Leo is an animal.
Output:
We can get these predicates:
Pet(x), Animal(x), Dog(x), Cat(x), \\
HasPet(x), Cares(x,y), Naughty(x),
    ↪ Liked(x,y)
We can get these constants:
charlie,leo
Let's translate one by one.
Premises:
1.Text:All pets are animals.
Predicates:
Animal(x),Pet(x)
```
Fol: $\forall x(Pet(x) \rightarrow Animal(x))$
```
2.Text:Pets can be either a dog or a
    ↪ cat.
Predicates:
Cat(x),Pet(x),Dog(x)
```
Fol: $\forall x(Pet(x) \rightarrow Dog(x) \oplus Cat(x))$
```
3.Text:If a person has a pet, they care
    ↪ for that pet.
Predicates:
Cares(x,y),HasPet(x)
```
Fol: $\forall x \forall y(HasPet(x) \rightarrow Cares(x, y))$
```
4.Text:Dogs and Cats can be naughty.
Predicates:
Cat(x),Naughty(x),Dog(x)
```
Fol: $\exists x((Cat(x) \land Naughty(x)) \lor$
$(Dog(x) \land Naughty(x)))$
```
5.Text:Pets who are naughty are not
    ↪ liked as much.
Predicates:
Naughty(x),Pet(x),Liked(x,y)
```
Fol: $\forall x \forall y(Pet(x) \land Naughty(x) \rightarrow \neg Liked(x, y))$
```
6.Text:Charlie has a naughty pet dog
    ↪ named Leo.
Predicates:
Pet(x),Naughty(x),HasPet(x),Dog(x)
Constants:
charlie,leo
```
Fol: $HasPet(charlie) \land Pet(leo)$
$\land Dog(leo) \land Naughty(leo)$
```
Conclusion:
Text:Leo is an animal.
Predicates:
Animal(x)
Constants:
leo
```
Fol: $Animal(leo)$

Listing 4: An Example from the Translation KB for FOLIO

```
Input:
Premises:
1.All commercial aircraft are produced
    ↪ by either Boeing or Airbus.
origin: $\forall x(CommercialAircraft(x)$
$\rightarrow (ProducedByBoeing(x) \lor ProducedByAirbus(x)))$
true answer: $\forall x(CommercialAircraft(x)$
$\rightarrow ProducedBy(x, boeing) \oplus ProducedBy(x, airbus))$
2.All American Airlines jets are
    ↪ commercial aircraft.
origin: $\forall x(AmericanAirlinesJet(x)$
$\rightarrow CommercialAircraft(x))$
true answer: $\forall x(AmericanAirlinesAircraft(x)$
$\rightarrow CommercialAircraft(x))$
3.Airbus made more in revenue than
    ↪ Boeing last year.
origin: $\exists y(Revenue(airbus, y) \land$
$\exists z(Revenue(boeing, z) \land y > z))$
true answer: $MoreinRevenue(airbus, boeing)$

Conclusion:
There does not exist a United Airlines
    ↪ plane produced by Boeing.
Error:
$\exists y(Revenue(airbus, y) \land$
$\exists z(Revenue(boeing, z) \land y > z))$ error :
    ↪ ("Illegal comparisons: Matched
    ↪ 'y > z' from position 50 to 55.
    ↪ You can't use x > something, try
    ↪ to use predicate(x, something)
    ↪ to replace it.",) Invalid
    ↪ symbols in `=<>>=<=` at position
    ↪ 52

Output:
Premises:

1. Original:
   $\forall x(CommercialAircraft(x)$
   $\rightarrow (ProducedByBoeing(x)$
   $\lor ProducedByAirbus(x)))$
   Explanation: The use of $\lor$ (logical
       ↪ disjunction) is incorrect
       ↪ here because it implies that
       ↪ an aircraft can be produced
       ↪ by both Boeing and Airbus,
       ↪ which is not the case. We
       ↪ should use $\oplus$ (logical
       ↪ exclusive disjunction) to
       ↪ indicate that an aircraft is
       ↪ produced by either Boeing or
       ↪ Airbus, but not both.
   Corrected:
   $\forall x(CommercialAircraft(x)$
   $\rightarrow (ProducedByBoeing(x)$
   $\oplus ProducedByAirbus(x)))$

2. Original:
   $\forall x(AmericanAirlinesJet(x)$
   $\rightarrow CommercialAircraft(x))$
   Explanation: There is no need to
       ↪ change.

3. Original:
   $\exists y(Revenue(airbus, y)$
   $\land \exists z(Revenue(boeing, z) \land y > z))$
   Explanation: The use of '>' (greater
       ↪ than) is invalid according to
       ↪ the grammar rules. We need to
       ↪ replace it with a predicate
       ↪ that represents the
       ↪ relationship between the
       ↪ revenues of Airbus and
       ↪ Boeing. We can use a
       ↪ predicate
       ↪ $MoreInRevenue(airbus, boeing)$ to
       ↪ indicate that Airbus made
       ↪ more revenue than Boeing.
   Corrected:
   $MoreInRevenue(airbus, boeing)$

Conclusion:

Original:
$\neg\exists x(UnitedAirlinesPlane(x)$
$\land ProducedByBoeing(x))$
Explanation: There is no need to change.

Final Corrected Formulas:

Premises:
1. $\forall x(CommercialAircraft(x)$
$\rightarrow (ProducedByBoeing(x) \oplus ProducedByAirbus(x)))$
    ↪
2. $\forall x(AmericanAirlinesJet(x)$
$\rightarrow CommercialAircraft(x))$
3. $MoreInRevenue(airbus, boeing)$

Conclusion:
$\neg\exists x(UnitedAirlinesPlane(x)$
$\land ProducedByBoeing(x))$
```

Listing 5: An Example from the Fixer KB for FOLIO

A.2.2 AR-LSAT

```
Input:
# Context:
Charlie makes a soup by adding exactly
    ↪ six kinds of foods---kale,
    ↪ lentils, mushrooms, onions,
    ↪ tomatoes, and zucchini---to a
    ↪ broth, one food at a time. No
    ↪ food is added more than once.
    ↪ The order in which Charlie adds
    ↪ the foods to the broth must be
    ↪ consistent with the following:
    ↪ If the mushrooms are added
    ↪ third, then the lentils are
    ↪ added last. If the zucchini is
    ↪ added first, then the lentils
    ↪ are added at some time before
    ↪ the onions. Neither the tomatoes
    ↪ nor the kale is added fifth. The
    ↪ mushrooms are added at some time
    ↪ before the tomatoes or the kale,
    ↪ but not before both.

# Question:
If the lentils are added last, then
    ↪ which one of the following must
    ↪ be true?

# Options:
A) At least one of the foods is added
    ↪ at some time before the zucchini.
B) At least two of the foods are added
    ↪ at some time before the kale.
C) The mushrooms are added third.
D) The zucchini is added third.
E) The tomatoes are added fourth.

Output:
# Declarations
foods = EnumSort([kale, lentils,
    ↪ mushrooms, onions, tomatoes,
    ↪ zucchini])
order = Function([foods] -> [int])
# Constraints
ForAll([f:foods], And(order(f) <= 6,
    ↪ order(f) >= 1)) ::: **Implied
    ↪ Information**: The order is
    ↪ defined between 1 and 6.
ForAll([f1:foods, f2:foods], Implies(f1
    ↪ != f2, order(f1) != order(f2)))
    ↪ ::: **Implied Information**: All
    ↪ food items have different
    ↪ positions.
Implies(order(mushrooms) == 3,
    ↪ order(lentils) == 6) ::: If the
    ↪ mushrooms are added third, then
    ↪ the lentils are added last
Implies(order(zucchini) == 1,
    ↪ order(lentils) < order(onions))
    ↪ ::: If the zucchini is added
    ↪ first, then the lentils are
    ↪ added at some time before the
    ↪ onions
And(order(tomatoes) != 5, order(kale)
    ↪ != 5) ::: Neither the tomatoes
    ↪ nor the kale is added fifth
Exists([f:foods],
    ↪ And(Or(order(mushrooms) <
    ↪ order(tomatoes),
    ↪ order(mushrooms) < order(kale)),
    ↪ Not(And(order(mushrooms) <
    ↪ order(tomatoes),
    ↪ order(mushrooms) <
    ↪ order(kale))))) ::: The
    ↪ mushrooms are added at some time
    ↪ before the tomatoes or the kale,
    ↪ but not before both
order(lentils) == 6 ::: If the lentils
    ↪ are added last
# Options
is_valid(Exists([f:foods], order(f) <
    ↪ order(zucchini))) ::: (A)
is_valid(Count([f:foods], order(f) <
    ↪ order(kale)) >= 2) ::: (B)
is_valid(order(mushrooms) == 3) ::: (C)
is_valid(order(zucchini) == 3) ::: (D)
is_valid(order(tomatoes) == 4) ::: (E)
```

Listing 6: An Example from the Translation KB for AR-LSAT

~~~
Input:
### Context:
Eight camp counselorsFran, George,
    ↪ Henry, Joan, Kathy, Lewis,
    ↪ Nathan, and Olgamust each be
    ↪ assigned to supervise exactly
    ↪ one of three activitiesswimming,
    ↪ tennis, and volleyball. The
    ↪ assignment of counselors must
    ↪ conform to the following
    ↪ conditions: Each activity is
    ↪ supervised by at least two, but
    ↪ not more than three, of the
    ↪ eight counselors. Henry
    ↪ supervises swimming. Neither
    ↪ Kathy nor Olga supervises
    ↪ tennis. Neither Kathy nor Nathan
    ↪ supervises the same activity as
    ↪ Joan. If George supervises
    ↪ swimming, both Nathan and Olga
    ↪ supervise volleyball.
### Question:
Which one of the following is a pair of
    ↪ counselors who could be two of
    ↪ three counselors assigned to
    ↪ supervise swimming?
### Options:
A) George and Nathan
B) George and Olga
C) Joan and Kathy
D) Joan and Nathan
E) Joan and Olga
### Logic program
```plaintext
# Declarations
counselors = EnumSort([Fran, George,
    ↪ Henry, Joan, Kathy, Lewis,
    ↪ Nathan, Olga])
activities = EnumSort([swimming,
    ↪ tennis, volleyball])
assigned = Function([counselors] ->
    ↪ [activities])

# Constraints
ForAll([a:activities],
    ↪ And(Count([c:counselors],
    ↪ assigned(c) == a) >= 2,
~~~

```
    ↪ Count([c:counselors],
    ↪ assigned(c) == a) <= 3)) :::
    ↪ Each activity is supervised by
    ↪ at least two, but not more than
    ↪ three, of the eight counselors
assigned(Henry) == swimming ::: Henry
    ↪ supervises swimming
assigned(Kathy) != tennis ::: Neither
    ↪ Kathy nor Olga supervises tennis
assigned(Olga) != tennis ::: Neither
    ↪ Kathy nor Olga supervises tennis
assigned(Kathy) != assigned(Joan) :::
    ↪ Neither Kathy nor Nathan
    ↪ supervises the same activity as
    ↪ Joan
assigned(Nathan) != assigned(Joan) :::
    ↪ Neither Kathy nor Nathan
    ↪ supervises the same activity as
    ↪ Joan
Implies(assigned(George) == swimming,
    ↪ And(assigned(Nathan) ==
    ↪ volleyball, assigned(Olga) ==
    ↪ volleyball)) ::: If George
    ↪ supervises swimming, both Nathan
    ↪ and Olga supervise volleyball

# Options
is_accurate_list([assigned(George) ==
    ↪ swimming, assigned(Nathan) ==
    ↪ swimming]) ::: (A)
is_accurate_list([assigned(George) ==
    ↪ swimming, assigned(Olga) ==
    ↪ swimming]) ::: (B)
is_accurate_list([assigned(Joan) ==
    ↪ swimming, assigned(Kathy) ==
    ↪ swimming]) ::: (C)
is_accurate_list([assigned(Joan) ==
    ↪ swimming, assigned(Nathan) ==
    ↪ swimming]) ::: (D)
is_accurate_list([assigned(Joan) ==
    ↪ swimming, assigned(Olga) ==
    ↪ swimming]) ::: (E)
```
### Error
No correct option problem. All options
    ↪ in this program are judged as
    ↪ incorrect, but this question is
    ↪ a single-choice question. It
    ↪ seems that some information has
    ↪ been overlooked, and the program
    ↪ needs to be re-examined.

Output:
### Analysis
1. There is no statement that each
    ↪ counselor can only be assigned
    ↪ to one activity; this constraint
    ↪ needs to be added.
2. The question asks which two people
    ↪ can simultaneously be assigned
    ↪ to swimming, so `is_sat` should
    ↪ be used.
The final program is modified as
    ↪ follows:
```plaintext
# Declarations
counselors = EnumSort([Fran, George,
    ↪ Henry, Joan, Kathy, Lewis,
    ↪ Nathan, Olga])
activities = EnumSort([swimming,
    ↪ tennis, volleyball])
assigned = Function([counselors] ->
    ↪ [activities])

# Constraints
ForAll([a:activities],
    ↪ And(Count([c:counselors],
    ↪ assigned(c) == a) >= 2,
    ↪ Count([c:counselors],
    ↪ assigned(c) == a) <= 3)) :::
    ↪ Each activity is supervised by
    ↪ at least two, but not more than
    ↪ three, of the eight counselors
ForAll([c:counselors],
    ↪ Count([a:activities],
    ↪ assigned(c) == a) == 1) ::: A
    ↪ person can only participate in
    ↪ one activity.
assigned(Henry) == swimming ::: Henry
    ↪ supervises swimming
assigned(Kathy) != tennis ::: Neither
    ↪ Kathy nor Olga supervises tennis
assigned(Olga) != tennis ::: Neither
    ↪ Kathy nor Olga supervises tennis
assigned(Kathy) != assigned(Joan) :::
    ↪ Neither Kathy nor Nathan
    ↪ supervises the same activity as
    ↪ Joan
assigned(Nathan) != assigned(Joan) :::
    ↪ Neither Kathy nor Nathan
    ↪ supervises the same activity as
    ↪ Joan
Implies(assigned(George) == swimming,
    ↪ And(assigned(Nathan) ==
    ↪ volleyball, assigned(Olga) ==
    ↪ volleyball)) ::: If George
    ↪ supervises swimming, both Nathan
    ↪ and Olga supervise volleyball

# Options
is_sat(And(assigned(George) ==
    ↪ swimming, assigned(Nathan) ==
    ↪ swimming)) ::: (A)
is_sat(And(assigned(George) ==
    ↪ swimming, assigned(Olga) ==
    ↪ swimming)) ::: (B)
is_sat(And(assigned(Joan) == swimming,
    ↪ assigned(Kathy) == swimming))
    ↪ ::: (C)
is_sat(And(assigned(Joan) == swimming,
    ↪ assigned(Nathan) == swimming))
    ↪ ::: (D)
is_sat(And(assigned(Joan) == swimming,
    ↪ assigned(Olga) == swimming)) :::
    ↪ (E)
```

Listing 7: An Example from the Fixer KB for AR-LSAT