

---

# SATLM: Satisfiability-Aided Language Models Using Declarative Prompting

---

Xi Ye    Qiaochu Chen    Isil Dillig    Greg Durrett  
Department of Computer Science  
The University of Texas at Austin  
{xiye,qchen,isil,gdurrett}@cs.utexas.edu

## Abstract

Prior work has combined chain-of-thought prompting in large language models (LLMs) with programmatic representations to perform reasoning. While such an approach works well for tasks that only require forward reasoning (e.g., straightforward arithmetic), it is less effective for problems that require more sophisticated planning and search. In this paper, we propose a new *satisfiability-aided language modeling* (SATLM) approach for improving the reasoning capabilities of LLMs. We use an LLM to generate a *declarative task specification* rather than an imperative program and leverage an off-the-shelf automated theorem prover to derive the final answer. By offloading the actual reasoning task to an automated theorem prover, our approach can guarantee the correctness of the answer with respect to the parsed specification and avoid planning errors in the solving process. We evaluate SATLM on 6 datasets and show that it consistently outperforms program-aided LMs in an imperative paradigm. In particular, SATLM outperforms program-aided LMs by more than 20% on a challenging subset of the GSM arithmetic reasoning dataset; SATLM also achieves a new SoTA on LSAT and BOARDGAMEQA.

## 1 Introduction

Using large language models (LLMs) to perform complex reasoning has been a central thrust of recent research on LLMs [1, 4, 26, 38]. Solving a complex reasoning problem involves three conceptual components: parsing the problem to solve out of its natural language description, deriving a plan for solving the problem, and executing that plan to obtain an answer. Recent work on improving CoT prompting focuses on fixing *execution errors* by augmenting LLMs with symbolic executors such as a Python interpreter, which leads to improved performance on arithmetic and symbolic reasoning tasks [10, 3, 20]. However, CoT prompting [34, 23] and its executor-augmented successors [10, 3, 20] are oriented towards *imperative* solving procedures: a CoT or a program specifies the reasoning procedure as chained steps [34, 10] in the order of execution. While this is effective for problems whose natural language already provides a suitably clear “plan” for the reasoning, it only leads to limited success for reasoning problems like in Figure 1 that do not outline such a plan [28].

Our work tackles both execution errors and, more importantly, *planning errors*. We propose SATisfiability-aided Language Modeling (SATLM) using declarative prompting. The core idea is to cast a natural language (NL) reasoning problem as a satisfiability (SAT for short) problem specified by declarative logical formulas. As shown in Figure 1 (right), given a problem in NL, we prompt an LLM to parse it into a SAT problem which consists of a set of logical formulas, then obtain the solution by invoking a SAT solver.<sup>1</sup> The LLM can be prompted to accurately understand the preconditions stated in the problem while leveraging the solver for planning out the solving

---

<sup>1</sup>We use the term SAT solver to refer to any automated reasoning tool for checking the satisfiability in formal logic. Hence, “SAT solver” in this paper includes first-order theorem provers as well as SMT solvers.

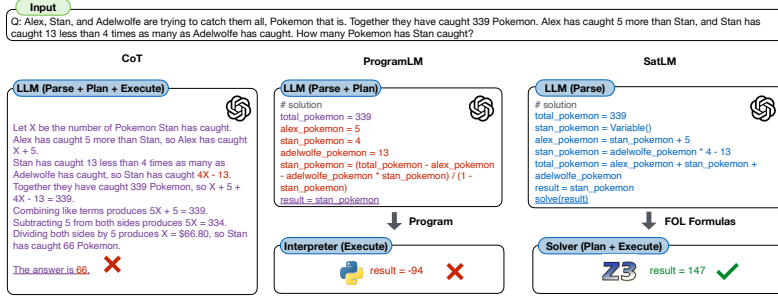


Figure 1: Illustration of our SATisfiability-aided Language Modeling approach (right). We first parse an NL input into a task specification (a set of logic constraints) using declarative prompting (Section 2.1) and then employ a SAT solver to solve the problem (Section 2.2). By only using the LLMs to generate declarative specifications and relying on a solver to handle the reasoning, SATLM generates the correct answer. By contrast, CoT makes errors when parsing an equation; PROGLM produces an incorrect reasoning chain (both errors are highlighted in red).

procedure. In addition, the SAT solver also can also guarantee the correctness of execution, similar to the interpreter used in program-aided language models (PROGLM).

We evaluate our approach on 6 datasets on arithmetic reasoning and logical reasoning. Our SATLM consistently outperforms CoT and PROGLM across all datasets, usually by a large margin. On GSM-SYS and ALGEBRA, SATLM outperforms PROGLM by more than 20%; on GSM, SATLM achieves 84.8% with self-consistency decoding using few-shot prompting, equaling against past work that uses the full training set and the same LLM [18, 22]. SATLM also sets a new SoTA on LSAT [39] and BOARDGAMEQA [14].

## 2 SAT-Aided Language Models using Declarative Prompting

**Overview** This work studies using LLMs to solve NL reasoning tasks. At a high level, an NL reasoning task is a NL description of a collection of facts  $\Phi$  (such as propositions or constraints) about some objects and a question  $Q$  related to these objects. The goal of the reasoning task is to find an answer to  $Q$  that can be deduced from the information provided in  $\Phi$ .

We conceptualize the general procedure for solving NL reasoning tasks in three steps: *parsing*, *planning*, and *execution*. We are given natural language input  $x_{test} = (NL(\Phi), NL(Q))$  which describes both  $\Phi$  and  $Q$ . Our first step is to parse this natural language into a predicted *task specification*  $(\hat{\Phi}, \hat{Q})$ , which is a *formal* description of the facts and the query.

Given  $(\hat{\Phi}, \hat{Q})$ , the planning step then involves determining a sequence of reasoning steps  $[r_1, \dots, r_n]$  beginning with the task specification and ending with the answer to the question. Each steps involves invoking a function (e.g., arithmetic operator or logical operator) that produces intermediate results which can be utilized in subsequent steps. A plan can be formulated by an LLM with CoT prompting or by a symbolic solver as in our work here. Finally, we execute the plan systematically with either a symbolic executor or an LLM, returning the output of the last step,  $r_n$ , as the final answer.

Our solution, SAT-aided LLMs approaches the problem using exactly these three steps. Prior work, CoT and PROGLM, can also be framed in the parse-plan-execute framework proposed above. In particular, CoT [23, 34] uses LLMs to perform each of the three steps. PROGLM [10, 3, 20] combine the parsing and planning steps to use an LLM to derive a program that corresponds to the plan. Please refer to Figure 1 for concrete examples.

### 2.1 Declarative Prompting

We use few-shot prompting to generate the specification  $s_{test}$  for the test input  $x_{test}$ . Specifically, we include few-shot input-specifications pairs  $(x_i, s_i)_{i=1}^k$  in the prompt, append test input  $x_{test}$  after the prompt, and let the LLM complete the specification for  $x_{test}$ , i.e.,  $s_{test} = p(x_{test} \mid x_1, s_1, \dots, x_k, s_k)$ . We show an example specification for an arithmetic reasoning task in Figure 1.

Table 1: Comparison of our approach (SATLM) against standard prompting, CoT and PROGLM. With greedy decoding, SATLM outperforms CoT and PROGLM on all datasets except for GSM by a substantial margin, and is on par with PROGLM on GSM. With self-consistency decoding, SATLM is consistently better than PROGLM, giving SoTA accuracy on LSAT and BOARDGAMEQA.

	GSM-SYS	GSM	ALGEBRA	LSAT	BOARD	CLUTRR	PROOF
<i>code-davinci-002 (greedy decoding)</i>							
STANDARD	21.0	22.2	45.9	22.0	44.6	41.2	76.6
CoT	46.5	62.7	53.6	23.5	60.7	40.8	80.1
PROGLM	43.4	<b>72.7</b>	52.3	—	—	58.9	83.7
SATLM	<b>69.4</b>	71.8	<b>77.5</b>	<b>35.0</b>	<b>79.4</b>	<b>68.3</b>	<b>99.7</b>
<i>code-davinci-002 (self-consistency decoding)</i>							
CoT	56.1	77.3	64.9	23.1	62.8	45.7	88.7
PROGLM	53.4	82.4	57.7	—	—	71.9	91.2
SATLM	<b>80.9</b>	<b>84.8</b>	<b>90.9</b>	<b>37.4</b>	<b>80.7</b>	<b>80.1</b>	<b>99.7</b>

Observe that in both examples, our SAT formulas (i.e., the logical formulas of  $[z_1, \dots, z_n]$  in  $\Sigma_{LF}$ ) are written as code following Python syntax and the natural language parts in  $\Sigma_{NL}$  are written using comment syntax.

## 2.2 Solving with a SAT Solver

**SAT problem** A SAT problem, denoted as  $\mathcal{P}$ , is a triple  $(\Phi, \mathcal{T}, Q)$  where  $\Phi$  is a set of first-order logic formulas in some theory  $\mathcal{T}$  and  $Q$  is the query of interest. We use  $\text{Variable}(\mathcal{P})$  to denote the free variables in  $\Phi$ .  $Q$  contains only variables in  $\text{Variable}(\mathcal{P})$ . An example SAT problem is  $\mathcal{P} = (\{x + y = 3, x - y = 1\}, \mathcal{T}_E \cup \mathcal{T}_Z, x - 2)$ , where  $\mathcal{T}_E \cup \mathcal{T}_Z$  indicates that only equality and linear arithmetic operations on integers are allowed in the formulas.

Many NL reasoning tasks can be formulated as SAT problems and solved using an off-the-shelf solver. For **arithmetic reasoning**, the SAT formulas  $\Phi$  are equations encoding the relationships between variables, and  $t$  specifies the target variable asked in the question (see Figure 1). For **logical reasoning**,  $\Phi$  encodes preconditions and  $t$  specifies the target statement posed by the question.

**Parsing NL to a SAT problem** Recall that we obtain a specification  $s_{test}$  from a test NL task  $x_{test}$ . To derive the SAT problem  $\mathcal{P}_{test} = (\hat{\Phi}_{test}, \mathcal{T}_{test}, \hat{Q}_{test})$  from  $s_{test}$ , we extract the constraints  $\hat{\Phi}_{test}$  and the target expression  $\hat{Q}_{test}$  (marked by solve in our prompt) by taking all the  $z_i$  in  $\Sigma_{LF}$  of  $s_{test}$ . We identify the theory  $\mathcal{T}_{test}$  by analyzing the formulas in  $\hat{\Phi}_{test}$ .

**Solving the SAT problem** Given the SAT problem  $\mathcal{P}$ , we invoke an automated theorem prover (such as the Z3 SMT solver [7] used in our implementation) to obtain a model  $M$  that maps each free variable  $v \in \text{Variable}(\mathcal{P})$  to a concrete value under theory  $\mathcal{T}$ . The final answer is obtained by substituting each free variable  $v_i$  in  $\hat{Q}$  with  $M[v_i]$ . For example, given the problem  $(\{x + y = 3, x - y = 1\}, \mathcal{T}_E \cup \mathcal{T}_Z, x - 2)$ , we ask the solver to find a solution to the constraint  $x + y = 3 \wedge x - y = 1$  in the theory  $\mathcal{T}_E \cup \mathcal{T}_Z$ , which yields  $x = 2$  and  $y = 1$ . Then, to obtain the final answer, we substitute  $x$  by 2 in the target expression  $x - 2$  to obtain the result  $2 - 2 = 0$ .

## 3 Experiments

**Setup** Our work mainly investigates 2 arithmetic reasoning datasets and 4 logical reasoning tasks. We compare SATLM against 3 baselines, including standard prompting (directly giving the answer), CoT, and executor-augmented LLMs (PROGLM). We conduct our main experiments and analysis on code-davinci-002 [2]. We evaluate the performance with both greedy decoding and self-consistency decoding [33]. We list all dataset statistics in Appendix A and detailed setup in Appendix G.

**Main Results** Table 1 shows the performance of our approach compared to the baselines. In general, our SAT-aided approach outperforms both CoT and PROGLM by a substantial margin except on GSM with greedy decoding.

The first two columns show the performance on the GSM dataset. CoT and PROGLM achieve much worse performance on GSM-SYS than on GSM, indicating that GSM-SYS is a challenging subset. On this subset, SATLM achieves 69.4% and 80.9% with greedy decoding and self-consistency decoding, surpassing both PROGLM and CoT more than by 20%. On the original GSM dataset, the SATLM model has a slightly lower accuracy than PROGLM with greedy decoding, but outperforms it with self-consistency decoding by 2.4%. This self-consistency accuracy of 84.8% even exceeds recent work that uses full training set on the same LLM (82.3% in DIVERSE [18]; 84.5% in LEVER [22]).

On LSAT, BOARDGAMEQA, CLUTRR, and PROOFWRITER, SATLM consistently achieves the best performance with either greedy decoding or self-consistency decoding. In particular, SATLM also sets the new SoTA on both LSAT and BOARDGAMEQA, surpassing previous models that are trained on the full training set. Specifically, SATLM boots the SoTA from 30.9% [39] to 37.4% on LSAT and from 73.9% [14] to 80.7% on BOARDGAMEQA.

### 3.1 Analysis

**Ablation: Impact of Symbolic Solver** We test a variant of our approach that still uses declarative prompting but then solves the equations in NL with CoT rather than using the symbolic solver (see Figure 15 in Appendix for concrete examples). Essentially, the LLM itself carries out planning and execution. As shown in Table 2, SAT<sub>COTSOLVER</sub> can solve many more SAT problems than NO-SOLVER. This partially reflects the effectiveness of CoT and partially reflects the fact that many dataset instances require relatively simple planning and execution, allowing pure forward reasoning to solve them. However, using a symbolic solver (SAT<sub>SYMSOLVER</sub>), which guarantees correct planning and execution, leads to further improvements.

#### Results Across Different Language Models

In addition to the main LLM used in our work, code-davinci-002, we further test whether SATLM can generalize to other LLMs. We choose gpt-3.5-turbo (0613 version), text-davinci-003, and code-davinci-001. gpt-3.5-turbo is optimized for chat. text-davinci-003 is an LLM pretrained on NL, and tuned to align with human feedback [24]. code-davinci-001 is also an LLM pretrained on code, but less capable compared to 002. As shown in Table 3, SATLM is better than PROGLM on the arithmetic reasoning and logical reasoning datasets except for GSM across these three LLMs. The trend is congruent with the results on code-davinci-002 (Table 1), which suggests the approach’s general applicability across different LLMs, regardless of their varying capabilities.

#### Sensitivity to Different Exemplar Sets

We test whether the advantages of SATLM is sensitive to different sets of exemplars. We experiment with 3 sets of exemplars on code-davinci-002. As shown in Table 4, SATLM consistently outperforms PROGLM by a large margin on GSM-SYS and CLUTRR, and achieves comparable performance on GSM. The results suggest the effectiveness of our approach is insensitive to varying the choice of exemplars.

#### LLMs Can Perform Commonsense Reasoning While Parsing

There are many problems that do not state premises or constraints in a completely explicit way. Figure 2) shows two examples

Table 2: The performance of variants of our approach that use CoT Solver. Using declarative prompting with CoT solver is more effective than imperative CoT prompting.

	GSM-SYS	GSM	CLUTRR
STANDARD	21.0	22.2	41.2
CoT	46.5	62.7	40.8
PAL	43.4	72.8	58.9
SAT <sub>SYMSOLVER</sub>	69.4	71.7	68.3
SAT <sub>COTSOLVER</sub>	54.5	63.2	48.9

Table 3: Results on gpt-3.5-turbo, text-davinci-003, and code-davinci-001. The effectiveness of SATLM can generalize across LLMs.

	GSM-SYS	GSM	LSAT	CLUTRR	PROOF
<i>gpt-3.5-turbo (greedy decoding)</i>					
CoT	44.8	74.4	23.9	41.2	82.3
PROGLM	51.2	<b>77.9</b>	—	45.9	76.4
SATLM	<b>63.4</b>	76.4	<b>30.0</b>	<b>50.6</b>	<b>96.4</b>
<i>text-davinci-003 (greedy decoding)</i>					
CoT	42.8	62.5	21.7	34.5	83.5
PROGLM	40.4	<b>71.7</b>	—	41.2	83.7
SATLM	<b>63.6</b>	70.3	<b>30.4</b>	<b>58.2</b>	<b>99.7</b>
<i>code-davinci-001 (greedy decoding)</i>					
PROGLM	15.5	<b>35.6</b>	—	22.2	63.8
SATLM	<b>16.5</b>	34.2	19.6	<b>30.2</b>	<b>86.6</b>

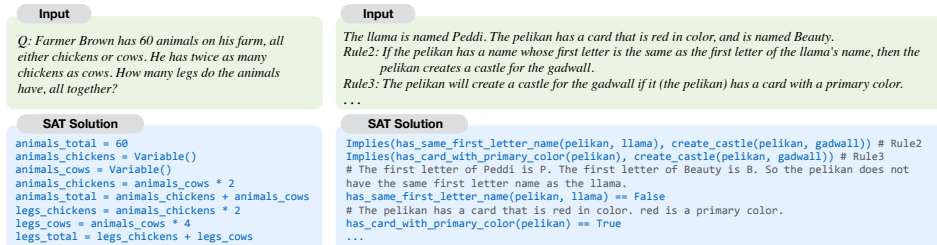


Figure 2: Examples outputs from GSM (left) and BOARDGAMEQA (right) show that LLMs can perform commonsense reasoning while parsing.

where commonsense inferences are required during parsing. For example, on the left, the model must recognize that *animals* refers to the chickens and cows collectively. Similarly, knowing that red is a primary color is needed to successfully apply rules on BOARDGAMEQA (right). We observe from the outputs in both cases that LLMs are capable of implicitly performing commonsense reasoning and produce correct logical formulas in the parsing step. As shown in Table 1, SATLM exhibits strong performance on BOARDGAMEQA, a dataset which requires this implicit background knowledge.

## 4 Related Work

Our work focus on improving LLMs on reasoning tasks, which are challenging for language models even with recent developments [21, 11]. Various techniques have been proposed for improving reasoning abilities [23, 40, 16, 15, 9, 32, 17, 20, 35]. They largely follow a chain-of-thought [34] or scratchpad [23] paradigm. Among them, our work is the most related to the line of work that generates imperative programs to be executed by a symbolic executor, such as a Python interpreter [10, 3] or domain-specific executors [20]. In this work, we propose a different paradigm that parses NL problems into declarative SAT problems and offloads the solving procedure to a SAT solver.

Table 4: The performance of PROGLM and SATLM with varying exemplar sets. SATLM consistently outperforms PROGLM on GSM-SYS and CLUTRR.

		GSM-SYS	GSM	CLUTRR
Set1	PROG	43.4	<b>72.7</b>	58.9
	SAT	<b>69.4</b>	71.8	<b>68.3</b>
Set2	PROG	41.4	<b>72.5</b>	59.0
	SAT	<b>71.8</b>	71.3	<b>67.9</b>
Set3	PROG	37.1	<b>70.3</b>	57.2
	SAT	<b>66.7</b>	70.0	<b>68.0</b>

Previous work has also explored equipping LLMs with other tools, including search engines [36, 29], calculators [5, 4], or domain-specific special modules [29, 8]. A line of work focuses on using program-related tools such as program executors [25], program analysis tools [13], and synthesis tools [27] to enhance the quality of the generated code. Our works further explores improving LLMs with SAT solvers.

## 5 Conclusion

We have presented a framework for satisfiability-aided language models for arithmetic reasoning and logical reasoning. We use an LLM to parse an NL query into a declarative specification and leverages a SAT solver to derive the final answer. Evaluation results on 6 datasets demonstrate the effectiveness of our approach over program-aided language models.

## References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS)*.

- [2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.
- [3] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.
- [4] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Baidoor Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor Lewkowycz, Erica Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. *ArXiv*, abs/2204.02311.
- [5] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- [6] Antonia Creswell, Murray Shanahan, and Irina Higgins. 2023. Selection-inference: Exploiting large language models for interpretable logical reasoning. In *The Eleventh International Conference on Learning Representations*.
- [7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08*, page 337–340, Berlin, Heidelberg. Springer-Verlag.
- [8] David Demeter and Doug Downey. 2020. Just add functions: A neural-symbolic language model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7634–7642.
- [9] Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- [11] Artur d’Avila Garcez and Luis C Lamb. 2023. Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, pages 1–20.
- [12] Joy He-Yueya, Gabriel Poesia, Rose E. Wang, and Noah D. Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *ArXiv*, abs/2304.09102.

- [13] Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, and Rahul Sharma. 2022. Jigsaw: Large language models meet program synthesis. *ICSE*.
- [14] Mehran Kazemi, Quan Yuan, Deepti Bhatia, Najoung Kim, Xin Xu, Vaiva Imbrasaitė, and Deepak Ramachandran. 2023. Boardgameqa: A dataset for natural language reasoning with contradictory information. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS)*.
- [15] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [16] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- [17] Shiyang Li, Jianshu Chen, Yelong Shen, Zhiyu Chen, Xinlu Zhang, Zekun Li, Hong Wang, Jing Qian, Baolin Peng, Yi Mao, et al. 2022. Explanations from large language models make small reasoners better. *arXiv preprint arXiv:2210.06726*.
- [18] Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*.
- [19] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher R’e, Diana Acosta-Navas, Drew A. Hudson, E. Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan S. Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas F. Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2022. Holistic evaluation of language models. *ArXiv*, abs/2211.09110.
- [20] Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*.
- [21] Gary Marcus. 2020. The next decade in AI: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*.
- [22] Ansong Ni, Srinu Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I Wang, and Xi Victoria Lin. 2023. LEVER: Learning to Verify Language-to-Code Generation with Execution. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- [23] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *ArXiv*, abs/2112.00114.
- [24] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback.
- [25] Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchronesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*.

- [26] Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John F. J. Mellor, Irina Higgins, Antonia Creswell, Nathan McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, L. Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, N. K. Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Tobias Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d’Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew G. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem W. Ayoub, Jeff Stanway, L. L. Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *ArXiv*, abs/2112.11446.
- [27] Kia Rahmani, Mohammad Raza, Sumit Gulwani, Vu Le, Daniel Morris, Arjun Radhakrishna, Gustavo Soares, and Ashish Tiwari. 2021. Multi-modal program inference: A marriage of pre-trained language models and component-based synthesis. *Proc. ACM Program. Lang.*, 5(OOPSLA).
- [28] Danilo Neves Ribeiro, Shen Wang, Xiaofei Ma, Henghui Zhu, Rui Dong, Deguang Kong, Juliette Burger, Anjelica Ramos, zhiheng huang, William Yang Wang, George Karypis, Bing Xiang, and Dan Roth. 2023. STREET: A multi-task structured reasoning and explanation benchmark. In *The Eleventh International Conference on Learning Representations*.
- [29] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv*.
- [30] 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)*.
- [31] 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 (ACL Findings)*.
- [32] Peifeng Wang, Aaron Chan, Filip Ilievski, Muhao Chen, and Xiang Ren. 2022. Pinto: Faithful language reasoning using prompt-generated rationales. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [33] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai hsin Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- [34] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NeurIPS)*.
- [35] Xi Ye and Greg Durrett. 2023. Explanation selection using unlabeled data for in-context learning. In *arXiv*.
- [36] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2023. Generate rather than retrieve: Large language models are strong context generators. In *International Conference for Learning Representation (ICLR)*.



- [37] Hanlin Zhang, Ziyang Li, Jiani Huang, Mayur Naik, and Eric Xing. 2022. Improved logical reasoning of language models via differentiable symbolic programming. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*.
- [38] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *ArXiv*, abs/2205.01068.
- [39] Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Yining Chen, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. 2022. Analytical reasoning of text. In *Findings of the Association for Computational Linguistics: NAACL (NAACL Findings)*.
- [40] Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. 2022. Least-to-most prompting enables complex reasoning in large language models. *ArXiv*, abs/2205.10625.

## A Detailed Statistics of Datasets

We show the statistics of all the datasets used in our paper in Table 5.

For CLUTRR, we follow the setting in FAITHFULCOT [20] and construct the prompt using exemplars requiring 2-3 reasoning steps and test whether the model can generalize to examples requiring up to 10 steps. We used the pre-processed test data consisting of 1,042 test examples from past work [20].

For PROOFWRITER, we use the closed world assumption setting, following past work [6]. We construct our test set by randomly sampling a subset of 1,000 examples (out of 10,000) from the test split of depth-5 setting, the most challenging setting.

Table 5: Number of few-shot exemplars, number of test examples and license for the datasets used in our paper.

	# Shot	# Test	License
GSM [5]	8	1,319	MIT license
GSM-SYS	8	547	MIT license
ALGEBRA [12]	8	222	Creative Commons Attribution Share Alike 4.0
LSAT [39]	8	230	MIT license
BOARDGAMEQA [14]	5	3,000	CC BY 4.0
CLUTRR [30]	8	1,042	Attribution-NonCommercial 4.0
PROOFWRITER [31]	4	1,000	CC BY 4.0.

**GSM-SYS Dataset** We construct GSM-SYS, a special subset consisting of 547 examples extracted from GSM. Specifically, we filter the entire GSM dataset (train split + test split) to find examples whose human-annotated explanations involve a system of equations, using patterns like “*let [letter] be*”, “*assume [letter] be*” and “*[number][letter]*”. We manually inspected 10% of the examples and found 80% of those samples did involve systems of equations in the explanation. We refer to this more challenging dataset as GSM-SYS.

## B Details of the Setup

**Details of Decoding** We evaluate the performance with both greedy decoding and self-consistency decoding [33]. Owing to the high computation cost, we use 5 samples for LSAT, BOARDGAMEQA, and PROOFWRITER, which involves long prompts, and use 40 samples for all other datasets.

**Details of Prompts** In general, we leverage CoT prompts and PROGLM prompts from existing work whenever available, and manually write SATLM prompts for the **same exemplar sets**. Prompt examples for all datasets can be found in Appendix G.

For arithmetic reasoning datasets, **GSM, GSM-SYS, and ALGEBRA**, we adapt the original CoT prompt and PROGLM prompt used in program-aided language models [10]. Specifically, we replace one random exemplar in the original prompt with another exemplar sampled from GSM-SYS. This is to improve the performance of CoT and PROGLM on GSM-SYS, as the original exemplar set achieves suboptimal performance for GSM-SYS. Our adapted CoT and PROGLM prompts achieve better performance compared to the original ones on both GSM and GSM-SYS (see Appendix C for details).

For **LSAT**, we randomly sample 8 exemplars and write prompts for CoT and SATLM. We note that LSAT is a particularly challenging task: we tried 3 CoT prompts written by 3 different authors of our paper, which all led to around 20% accuracy. Similar results are reported in other work [19, 28]. In addition, we only report CoT results, leaving out PROGLM. This decision is due to the fact that PROGLM uses Python as its program interpreter. While Python is a general-purpose programming language, it does not provide native support for formal logic reasoning, including essential components like logical inference rules and manipulation of logical formulas. Solving problems from LSAT requires strategies like proof by contradiction (see Appendix G for a detailed example), which we see no way to represent in the PROGLM framework and is not addressed in prior work.

**BOARDGAMEQA** contains problems requiring 1-3 depth of reasoning. We randomly sample 5 exemplars from the training set of depth 1 and depth 2 to construct the prompts for evaluation on the

test set of depth 1 and depth 2, respectively. We randomly sample 5 exemplars from the training set of depth 2 to construct the prompt for test set of depth 3, as using exemplars of depth 3 would lead to prohibitively long prompts to be consumed by LLMs. Similarly, we only report COT results as the baselines, leaving out PROGLM for BOARDGAMEQA. We use the proofs provided by the authors to construct the COT prompts and manually annotate the SAT specifications to construct the SATLM prompts.

For **CLUTRR**, we use the COT prompt and PROGLM prompt provided in FAITHFULCOT [20]. For **PROOFWRITER**, we use the COT prompt from SELECTION-INFERENCE [6], and adapt it to form the PROGLM prompt.

## C Performance of Original COT and PROGLM Prompts on Arithmetic Reasoning Datasets

Table 6: Performance of different approaches using our adapted exemplar set and the original exemplar set used in COT and PAL.

	ADAPTED (OURS)		ORIGINAL	
	GSM-SYS	GSM	GSM-SYS	GSM
COT	46.5	62.7	35.7	62.4
PROGLM	43.4	<b>72.7</b>	36.1	<b>71.7</b>
SATLM	<b>69.4</b>	71.8	<b>66.7</b>	70.9

Recall that we construct our arithmetic reasoning prompt used in Table 1 by replacing one random exemplar in the original prompt used in COT and PROGLM with an random example from GSM-SYS. We show the performance of COT, PROGLM, and our SATLM in Table 6 using our adapted exemplar set and original exemplar set in Table 6.

Our adaptation significantly improves the performance of COT and PROGLM on GSM-SYS, and slightly improves the performance on GSM. Furthermore, we still see that SATLM outperforms both COT and PROGLM by a large margin on GSM, using either our adapted set or the original set.

## D Extended Discussion on Concurrent Work

Table 7: Performance of different approaches on ALGEBRA.

	ALGEBRA	GSM
COT	53.6	62.4
PROGLM	52.3	72.7
SATLM (Ours)	77.5	71.8
MATHSYM [12]	76.3	69.4

Similar to our work, [12] proposes to solve arithmetic reasoning problems by parsing the problem into a set of variables and equations and using an external solver to derive the final answer. While their formalization is restricted to arithmetic problems, we use SAT problems encoded with first-order logical formulas that unifies a wide range of reasoning tasks.

In addition, we also evaluate our approach on the ALGEBRA dataset in [12], which consists of 222 examples from Algebra textbooks. We note that the results between ours and MATHSYM are not directly comparable, as MATHSYM picks a different exemplar set. As shown in Table 7, ALGEBRA is more challenging than GSM, and SATLM outperforms PROGLM and COT by more than 20%.

## E Examples of Error Cases of SATLM

### E.1 Examples of Unsatisfiable and Ambiguous Formulas

We show examples of formulas that are unsatisfiable or have ambiguous answers in Figure 3, Figure 4, and Figure 5.

---

#### UNSAT Formulas: GSM

---

Q: If a rectangle has a width of 42 inches and an area of 1638, how many rectangles of the same size would reach a length of 390 inches?

```
rectangle_width = 42
rectangle_area = 1638
rectangle_length = 390
rectangle_length = rectangle_area / rectangle_width
rectangles_needed = rectangle_length / rectangle_width
result = rectangles_needed
solve(result)
```

Q: Jason’s dog has a tail that’s half the length of its body, and a head that’s 1/6 the length of its body. If the dog is 30 inches long overall, how long is its tail?

```
dog_length = 30
tail_length = Variable()
head_length = Variable()
dog_length = tail_length + head_length
tail_length = dog_length * 1 / 2
head_length = dog_length * 1 / 6
result = tail_length
solver(result)
```

---

Figure 3: Examples of UNSAT formulas from GSM. Both examples are generated by code-davinci-002. UNSAT occurs when LLMs misunderstand the problem and output conflicting constraints (highlighted in red).

---

#### AMBIG Formulas: GSM

---

Q: **The sum of the three numbers is 500.** If the first number is 200, and the value of the second number is twice the value of the third number, find the value of the third number.

```
num1 = 200
num2 = Variable()
num3 = Variable()
num2 = num3 * 2
total = num1 + num2 + num3
result = num3
solve(result)
```

Q: **The total for the sum and product of Elvie’s age and Arielle’s age are 131.** If Elvie’s age is 10, how old is Arielle?

```
elvie_age = 10
arielle_age = Variable()
sum_age = elvie_age + arielle_age
product_age = elvie_age * arielle_age
total_age = sum_age + product_age
result = arielle_age
solve(result)
```

---

Figure 4: Examples of AMBIG solutions from GSM. Both examples are generated by code-davinci-002. The ambiguous formulas are under-constrained due to failure in encoding certain constraints specified in the problem (highlighted in red), resulting in multiple possible answers.

---

**UNSAT Formulas: CLUTRR**

---

Arlene and her husband Jeff went on a cruise. They had a wonderful time. Stephanie s father Jason loves his little princess even though she gets into a lot of trouble at school. Gloria’s mother Ruth and brother Jeff were working in the kitchen, preparing soup. Stephanie, Ruth’s daughter, was working outside in the garden.  
Q: How is [Jason] related to [Arlene]?

```
# [Arlene] and her husband [Jeff] went on a cruise. They had a wonderful time.
relation(Arlene, Jeff) = (wife, husband)
# [Stephanie]’s father [Jason] loves his little princess even though she gets into a lot of trouble at school.
relation(Stephanie, Jason) = (daughter, father)
# [Gloria]’s mother [Ruth] and brother [Jeff] were working in the kitchen, preparing soup.
relation(Gloria, Ruth) = (daughter, mother)
relation(Gloria, Jeff) = (daughter, brother)
# [Stephanie], [Ruth]’s daughter, was working outside in the garden.
relation(Stephanie, Ruth) = (daughter, mother)
# How is [Jason] related to [Arlene]?
solve(relation(Jason, Arlene))
```

---

**AMBIG Formulas: CLUTRR**

---

Kirk loves talking to his grandfather Stanley on the phone. Paul’s brother, Kirk, recently made it into town. Paul and his son Michael went to look at cars. Paul ended up buying the Mustang.  
Q: How is [Stanley] related to [Michael]?

```
# [Kirk] loves talking to his grandfather [Stanley] on the phone.
relation(Kirk, Stanley) = (grandson, grandfather)
# [Paul]s brother, [Kirk], recently made it into town.
relation(Paul, Kirk) = (brother, brother)
# [Paul] and his son [Michael] went to look at cars.
relation(Paul, Michael) = (father, son) note: Michael is Kirk’s son, not Paul’s son
# How is [Stanley] related to [Michael]?
solve(relation(Stanley, Michael))
```

---

Figure 5: An example of UNSAT formulas (top) and an example of AMBIG formulas (bottom) from CLUTRR. Both happen when the LLM misparses a constraint (highlighted in red). In particular, ambiguous formulas usually occur when the language description itself is highly ambiguous (see the note in the ambiguous formulas for an example).

## E.2 Examples of Incorrect Answers

We show examples where the SAT solver successfully returns an answer, but the answer is incorrect in Figure 6.

---

### Examples of Incorrect Answers: GSM

---

Q: Martin rings the small bell 4 times more than  $1/3$  as often as the big bell. If he rings both of them a combined total of 52 times, how many times does he ring the big bell?

```
small_bell_rings = Variable()
big_bell_rings = Variable()
total_bell_rings = 52
small_bell_rings = big_bell_rings * 1 / 3 * 4
total_bell_rings = small_bell_rings + big_bell_rings
result = big_bell_rings
solve(result)
```

---

Q: Jonathan was sad to learn he needed 2 more toys than he already had to have enough toys to make a sorted play set five times larger than James’s play set, which had 80 toys. How many toys does Jonathan currently have?

```
james_toys = 80
jonathan_toys = Variable()
jonathan_toys_needed = james_toys * 5 + 2
result = jonathan_toys_needed - 2
solve(result)
```

---

### Examples of Incorrect Answers: CLUTRR

---

Donald and Russell are brothers. Jason is father of their father.

Q: How is [Russell] related to [Jason]?

```
# [Donald] and [Russell] are brothers.
relation(Donald, Russell) = (brother, brother)
# [Jason] is father of their father
relation Jason, Donald = (father, son)
relation Jason, Russell = (father, son)
# How is [Russell] related to [Jason]?
solve(relation(Russell, Jason))
```

---

Figure 6: Two examples of incorrect answers from GSM (top) and an example of incorrect answer from CLUTRR (bottom). This occurs when LLMs output incorrect constraints (highlighted in red) but the incorrect formulas can still derive a non-ambiguous answer.

## F Examples of Execution Errors and Planning Errors Made by COTSOLVER

We show examples of execution errors and planning errors produced by COTSOLVER in Figure 7, Figure 8, and Figure 9.

As shown in Figure 7, LLMs can make errors when executing arithmetic operators, resulting in execution errors. While introducing an interpreter (PROGLM) can help fix such execution errors, it is not effective for handling planning errors in Figure 8, which involves manipulating symbolic variables to determine a chain of arithmetic equations leading to the final answer.

We note that for CLUTRR, the atomic function is applying a set of transitivity rules that infer the relation between A and C based on the relation between A and B and the relation between B and C (e.g., A's son is B, B's daughter is C. So A's granddaughter is C). So the an execution error in CLUTRR refers to incorrect outputs when applying transitivity rules (see the top section of Figure 9). The set of transitivity rules we use are provided in [20, 37].

---

### Execution Errors of COTSOLVER: GSM

---

Q: Carly is thinking about buying a wallet that costs 22 and a purse that costs 3 less than four times the cost of the wallet. What is the combined cost of both items?

```
wallet_cost = 22
purse_cost = Variable()
purse_cost = wallet_cost * 4 - 3
total_cost = wallet_cost + purse_cost
result = total_cost
# We know purse_cost = wallet_cost * 4 - 3
# So total_cost = wallet_cost + purse_cost = 22 + (22 * 4 - 3) = 22 + 87 = 109
# result = total_cost = 109
# The answer is 109
```

---

Q: A person borrowed an amount of money for a year at an interest rate of 12%. If the total interest is \$1500, what was the original borrowed amount?

```
interest_rate = 12
interest_total = 1500
borrowed_amount = Variable()
interest_total = borrowed_amount * interest_rate / 100 result = borrowed_amount
# We know interest_total = borrowed_amount * interest_rate / 100
# So borrowed_amount = interest_total / interest_rate * 100 = 1500 / 12 * 100 = 12500
/ 12 = 1041.67
# result = borrowed_amount = 1041.67
# The answer is 1041.67
```

---

Figure 7: Examples of execution errors made by COTSOLVER on arithmetic reasoning tasks. Both examples are generated by code-davinci-002, which makes mistakes when calculating the final value of equations.

---

### Planning Errors of CoTSOLVER: GSM

---

Q: Griffin had 24 french fries, but Kyle took 5 of them. Billy took twice as many as Kyle. Ginger gave Griffin a handful of her fries, and then Colby took from Griffin 3 less than the number of fries that Kyle had taken. If in the end Griffin had 27 fries, how many fries did Ginger give Griffin?

```
fries_initial = 24
kyle_took = 5
billy_took = kyle_took * 2
ginger_gave = Variable()
colby_took = kyle_took - 3
fries_after = 27
fries_after = fries_initial - kyle_took - billy_took + ginger_gave - colby_took
result = ginger_gave
# Solve the value of result
# We know fries_after = fries_initial - kyle_took - billy_took + ginger_gave - colby_took
# So ginger_gave = fries_after - fries_initial + kyle_took + billy_took - colby_took =
27 - 24 + 5 + 10 - 2 = 16
# result = ginger_gave = 16
# The answer is 16
```

---

Q: The combined age of Peter, Paul and Jean is 100 years old. Find the age of Peter knowing that Paul is 10 years older than John and that Peter's age is equal to the sum of Paul and John's age.

```
john_age = Variable()
total_age = 100
paul_age = john_age + 10
peter_age = paul_age + john_age
total_age = peter_age + paul_age + john_age
result = peter_age
# We know paul_age = john_age + 10, and peter_age = paul_age + john_age. So peter_age
= john_age + 10 + john_age = 2 * john_age + 10
# We also know total_age = peter_age + paul_age + john_age = 2 * john_age + 10 + john_age
+ 10 = 3 * john_age + 20
# Therefore, john_age = (total_age - 20) / 3 = (100 - 20) / 3 = 80 / 3 = 26.67
# peter_age = 2 * john_age + 10 = 2 * 26.67 + 10 = 53.33 + 10 = 63.33
# result = peter_age = 63.33
# The answer is 63.33
```

---

Figure 8: Examples of planning errors made by CoTSOLVER from GSM. Both examples are generated by code-davinci-002. In the first example, LLM fails to flip the sign of colby\_took. In the second example, the model drops an occurrence of john\_age when expanding out the formula. In both case, LLM fails to arrange the symbols to derive a chain of numeric equations that gives the correct answer.



---

**Execution Errors of COTSOLVER: CLUTRR**

---

```
# [James] was looking forward to dinner with his granddaughter, [Joanne].
relation(James, Joanne) = (grandfather, granddaughter)
# [Helen] went to her son [Leo]'s House
relation(Helen, Leo) = (mother, son)
# [Gene] bought gifts for his lovely sister, [Joanne]
relation(Gene, Joanne) = (brother, sister)
# [Kecia] wanted to visit an art museum, so she asked her father, [James] to take her.
relation(Kecia, James) = (daughter, father)
# [Sharon] and her sister, [Kecia], frequently bicker. However, they always make up quickly.
relation(Sharon, Kecia) = (sister, sister)
# [Leo] was so happy for his brother, [Gene], getting a promotion.
relation(Leo, Gene) = (brother, brother)
# How is [Sharon] related to [Helen]?
answer = relation(Sharon, Helen)
# Solve the problem
# Helen's son is Leo, and Leo's brother is Gene, so Helen's son is Gene.
# Helen's son is Gene, and Gene's sister is Joanne, so Helen's daughter is Joanne.
# Helen's daughter is Joanne, and Joanne's father is James, so Helen's son is James.
# Helen's son is James, and James's daughter is Kecia, so Helen's granddaughter is Kecia.
# Helen's granddaughter is Kecia, and Kecia's sister is Sharon, so Helen's granddaughter is Sharon.
# The answer is granddaughter.
```

---

**Planning Errors of COTSOLVER: CLUTRR**

---

```
# [Rosa] took her daughter [Melissa] to the park.
relation(Rosa, Melissa) = (mother, daughter)
# [Melissa] was sad her sister [Annie] could not join.
relation(Melissa, Annie) = (sister, sister)
# [Lisa] is having a great day at the playground with her mother [Rosa] and [Rosa]'s sister [Ruth].
relation(Lisa, Rosa) = (daughter, mother)
relation(Rosa, Ruth) = (sister, sister)
# [Rosa] asked her daughter, [Lisa], if she would like to go to a movie with her on Saturday night.
relation(Rosa, Lisa) = (mother, daughter)
# How is [Ruth] related to [Annie]?
answer = relation(Ruth, Annie)
# Solve the problem
# Rosa's daughter is Lisa, and Lisa's mother is Rosa, so Rosa's mother is Rosa.
# Rosa's mother is Rosa, and Rosa's sister is Ruth, so Rosa's aunt is Ruth.
# Rosa's aunt is Ruth, and Rosa's daughter is Melissa, so Rosa's niece is Melissa.
# Rosa's niece is Melissa, and Melissa's sister is Annie, so Rosa's niece is Annie.
# The answer is niece.
```

---

Figure 9: Examples of planning errors made by COTSOLVER on CLUTRR. We omit questions for brevity. Both examples are generated by code-davinci-002. In the first example, the model outputs an incorrect value when applying the transitivity rule marked in red (correct output should be husband). In the second example, the model comes up with an incorrect procedure.

## G Prompt Examples

We show one or two exemplars in the prompt for each dataset. We list prompts for PROGLM for comparison.

### Prompts for GSM and GSM-SYS

---

#### SATLM

---

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

```
jason_lollipops_initial = 20
lollipops_given = Variable()
jason_lollipops_after = 12
jason_lollipops_after = jason_lollipops_initial - lollipops_given
result = lollipops_given
solve(result)
```

Q: Jeff bought 6 pairs of shoes and 4 jerseys for \$560. Jerseys cost 1/4 price of one pair of shoes. Find the shoe's price total price.

```
shoes_num = 6
jerseys_num = 4
total_cost = 560
shoes_cost_each = Variable()
jerseys_cost_each = Variable()
shoes_cost_each * shoes_num + jerseys_cost_each * jerseys_num = total_cost
jerseys_cost_each = shoes_cost_each * 1 / 4
shoes_cost_total = shoes_cost_each * shoes_num
result = shoes_cost_total
solve(result)
```

---

#### PROGLM from [10]

---

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

```
jason_lollipops_initial = 20
jason_lollipops_after = 12
denny_lollipops = jason_lollipops_initial - jason_lollipops_after
result = denny_lollipops
return result
```

Q: Jeff bought 6 pairs of shoes and 4 jerseys for \$560. Jerseys cost 1/4 price of one pair of shoes. Find the shoe's price total price.

```
shoes_num = 6
jerseys_num = 4
total_cost = 560
jersey_shoes_cost_ratio = 1 / 4
shoes_cost_each = total_cost / (shoes_num + jerseys_num * jersey_shoes_cost_ratio)
shoes_cost_total = shoes_cost_each * shoes_num
result = shoes_cost_total
return result
```

---

Figure 10: Prompt (excerpt) used for GSM and GSM-SYS.

## Prompts for LSAT

---

### SATLM

---

Nine different treatments are available for a certain illness: three antibiotics—F, G, and H—three dietary regimens—M, N, and O—and three physical therapies—U, V, and W. For each case of the illness, a doctor will prescribe exactly five of the treatments, in accordance with the following conditions: If two of the antibiotics are prescribed, the remaining antibiotic cannot be prescribed. There must be exactly one dietary regimen prescribed. If O is not prescribed, F cannot be prescribed. If W is prescribed, F cannot be prescribed. G cannot be prescribed if both N and U are prescribed. V cannot be prescribed unless both H and M are prescribed.

Question: If O is prescribed for a given case, which one of the following is a pair of treatments both of which must also be prescribed for that case?

(A) F, M (B) G, V (C) N, U (D) U, V (E) U, W

```
treatments = [F, G, H, M, N, O, U, V, W]
antibiotics = [F, G, H]
dietary_regimens = [M, N, O]
physical_therapies = [U, V, W]
prescribed = Function(treatments, bool)
Count([t:treatments], prescribed(t)) == 5
Count([a:antibiotics], prescribed(a)) <= 2
Count([d:dietary_regimens], prescribed(d)) == 1
Implies(Not(prescribed(O)), Not(prescribed(F)))
Implies(prescribed(W), Not(prescribed(F)))
Implies(And(prescribed(N), prescribed(U)), Not(prescribed(G)))
Implies(prescribed(V), And(prescribed(H), prescribed(M)))

solve(Implies(prescribed(O), And(prescribed(U), prescribed(V)))) # (A)
solve(Implies(prescribed(O), And(prescribed(G), prescribed(V)))) # (B)
solve(Implies(prescribed(O), And(prescribed(N), prescribed(U)))) # (C)
solve(Implies(prescribed(O), And(prescribed(U), prescribed(V)))) # (D)
solve(Implies(prescribed(O), And(prescribed(U), prescribed(W)))) # (E)
```

---

### CoT (annotated by our authors)

---

Nine different treatments are available for a certain illness: three antibiotics—F, G, and H—three dietary regimens—M, N, and O—and three physical therapies—U, V, and W ..... (omitted, see above)

Question: If O is prescribed for a given case, which one of the following is a pair of treatments both of which must also be prescribed for that case?

(A) F, M (B) G, V (C) N, U (D) U, V (E) U, W

Let's first analyze the conditions. We know there must be exactly one dietary regimen prescribed. If O is a prescribed dietary regimen, then M and N must not be prescribed. We know V cannot be prescribed unless both H and M are prescribed, so V cannot be prescribed.

Let's consider option (A). M must not be prescribed. This option is False.

Let's consider option (B). V cannot be prescribed. This option is False.

Let's consider option (C). N must not be prescribed. This option is False.

Let's consider option (D). V cannot be prescribed. This option is False.

Let's consider option (E). Let's assume U is not prescribed. Because exactly 5 of the treatments will be prescribed, they must be F, G, H, O, and W. In this way both F and W are prescribed, which contradicts the condition that if W is prescribed, F cannot be prescribed. So U must be prescribed. We can prescribe G, H, O, U, W. This option is True.

So the answer is (E).

---

Figure 11: Prompt (excerpt) used for LSAT. Another example can be found in Figure ?? . Several of our authors authored different CoT prompts, leading to similar performance (ranging from 20% to 22%).

## Prompts for BOARDGAMEQA

---

### SATLM

---

A few players are playing a boardgame. The current state of the game is as follows. The gecko has 13 friends, and hates Chris Ronaldo. And the rules of the game are as follows. Rule1: If the gecko has more than 8 friends, then the gecko does not proceed to the spot that is right after the spot of the bat. Rule2: Regarding the gecko, if it is a fan of Chris Ronaldo, then we can conclude that it does not proceed to the spot that is right after the spot of the bat. Rule3: If something does not proceed to the spot right after the bat, then it does not give a magnifier to the swordfish.  
Q: Based on the game state and the rules and preferences, does the gecko give a magnifier to the swordfish?

```
# If the gecko has more than 8 friends, then the gecko does not proceed to the spot
that is right after the spot of the bat.
Implies(has_more_than_8_friends(gecko), Not(proceed_to_spot_right_after(gecko, bat)))
# Rule2: Regarding the gecko, if it is a fan of Chris Ronaldo, then we can conclude
that it does not proceed to the spot that is right after the spot of the bat.
Implies(is_fan_of_chris_ronaldo(gecko), Not(proceed_to_spot_right_after(gecko, bat)))
# Rule3: If something does not proceed to the spot right after the bat, then it does
not give a magnifier to the swordfish.
ForAll([x], Implies(Not(proceed_to_spot_right_after(x, bat)), Not(give_magnifier(x,
swordfish))))

# The current state of the game is as follows. The gecko has 13 friends, and hates
Chris Ronaldo.
# The gecko has 13 friends.
has_more_than_8_friends(gecko) == True
# The gecko hates Chris Ronaldo.
is_fan_of_chris_ronaldo(gecko) == False

# question: does the gecko give a magnifier to the swordfish?
solve(give_magnifier(gecko, swordfish))
```

---

### CoT from Kazemi et al. [14]

---

A few players are playing a boardgame. The current state of the game is as follows. The gecko has 13 friends, and hates Chris Ronaldo. And the rules of the game are as follows. Rule1: If the gecko has more than 8 friends, then the gecko does not proceed to the spot that is right after the spot of the bat. Rule2: Regarding the gecko, if it is a fan of Chris Ronaldo, then we can conclude that it does not proceed to the spot that is right after the spot of the bat. Rule3: If something does not proceed to the spot right after the bat, then it does not give a magnifier to the swordfish.  
Q: Based on the game state and the rules and preferences, does the gecko give a magnifier to the swordfish?

A: We know the gecko has 13 friends, 13 is more than 8, and according to Rule1 "if the gecko has more than 8 friends, then the gecko does not proceed to the spot right after the bat", so we can conclude "the gecko does not proceed to the spot right after the bat". We know the gecko does not proceed to the spot right after the bat, and according to Rule3 "if something does not proceed to the spot right after the bat, then it doesn't give a magnifier to the swordfish", so we can conclude "the gecko does not give a magnifier to the swordfish". So the statement "the gecko gives a magnifier to the swordfish" is disproved. The answer is no.

---

Figure 12: Prompt (excerpt) used for BOARDGAMEQA.

## Prompts for CLUTRR

---

### SATLM

---

Dorothy took her daughter Michelle and her mother Gabrielle car shopping.

Q: How is [Michelle] related to [Gabrielle]?

# [Dorothy] took her daughter [Michelle] and her mother [Gabrielle] car shopping.

relation(Dorothy, Michelle) = (mother, daughter)

relation(Dorothy, Gabrielle) = (daughter, mother)

# How is [Michelle] related to [Gabrielle]?

solve(relation(Michelle, Gabrielle))

Teresa and her brother Ellis were having a wonderful time at Disneyland. Ellis asked his grandmother, Molly, to read him a bedtime story. Molly read him Hansel & Gretel, which the boy always loved. Sandra is married to Thomas, the couple welcomed Teresa into the world.

Q: How is [Molly] related to [Sandra]?

# [Teresa] and her brother [Ellis] were having a wonderful time at Disneyland.

relation(Teresa, Ellis) = (sister, brother)

# [Ellis] asked his grandmother, [Molly], to read him a bedtime story.

relation(Ellis, Molly) = (grandson, grandmother)

# [Sandra] is married to Thomas, the couple welcomed [Teresa] into the world.

relation(Sandra, Teresa) = (mother, daughter)

# How is [Molly] related to [Sandra]?

solve (relation(Molly, Sandra))

---

### PROGLM from [20]

---

Dorothy took her daughter Michelle and her mother Gabrielle car shopping.

Q: How is [Michelle] related to [Gabrielle]?

# To answer this question, we write a program to answer the following subquestions:

# 1. How is [Michelle] related to [Dorothy]? (independent, support: "[Dorothy] took her daughter [Michelle] and her mother [Gabrielle] car shopping.")

relation(Michelle, Dorothy) = daughter

# 2. How is [Dorothy] related to [Gabrielle]? (independent, support: "[Dorothy] took her daughter [Michelle] and her mother [Gabrielle] car shopping.")

relation(Dorothy, Gabrielle) = daughter

# 3. Final answer: How is [Michelle] related to [Gabrielle]? (depends on 1, 2)

relation(Michelle, Gabrielle) = relation(Michelle, Dorothy) @ relation(Dorothy, Gabrielle)

Teresa and her brother Ellis were having a wonderful time at Disneyland..... (omitted, see above)

Q: How is [Molly] related to [Sandra]?

# To answer this question, we write a program to answer the following subquestions:

# 1. How is [Molly] related to [Ellis]? (independent, support: "[Ellis] asked his grandmother, [Molly], to read him a bedtime story.")

relation(Molly, Ellis) = grandmother

# 2. How is [Ellis] related to [Teresa]? (independent, support: "[Teresa] and her brother [Ellis] were having a wonderful time at Disneyland.")

relation(Ellis, Teresa) = brother

# 3. How is [Teresa] related to [Sandra]? (independent, support: "[Sandra] is married to Thomas, the couple welcomed [Teresa] into the world.")

relation(Teresa, Sandra) = daughter

# 4. Final answer: How is [Molly] related to [Sandra]? (depends on 1, 2, 3)

relation(Molly, Sandra) = relation(Molly, Ellis) @ relation(Ellis, Teresa) @ relation(Teresa, Sandra)

---

Figure 13: Prompt (excerpt) used for CLUTRR.

## Prompts for PROOFWRITER

---

### SATLM

---

Here are some facts and rules:

If someone visits the squirrel and the squirrel visits the rabbit then they are round. All round people are not kind. If someone is round then they chase the rabbit. If someone is red and they chase the rabbit then they visit the dog. If someone is red then they visit the squirrel. If someone visits the squirrel then the squirrel visits the rabbit. the rabbit visits the dog.

the squirrel chases the bald eagle. the squirrel chases the rabbit. the dog sees the bald eagle. the bald eagle does not chase the dog. the bald eagle is red. the squirrel is round. the rabbit does not see the dog. the rabbit sees the bald eagle. the rabbit sees the squirrel. the dog does not see the rabbit. the rabbit does not visit the bald eagle. the dog does not chase the bald eagle.

Q: The statement "The bald eagle visits the dog" is True or False?

```
ForAll([x], Implies(And(visit(x, squirrel), visit(squirrel, rabbit)), round(x)))
ForAll([x], Implies(round(x), Not(kind(x))))
ForAll([x], Implies(round(x), chase(x, rabbit)))
ForAll([x], Implies(And(red(x), chase(x, rabbit)), visit(x, dog)))
ForAll([x], Implies(red(x), visit(x, squirrel)))
ForAll([x], Implies(visit(x, squirrel), visit(squirrel, rabbit)))
chase(squirrel, rabbit)
see(dog, bald_eagle)
Not(chase(bald_eagle, dog))
red(bald_eagle)
round(squirrel)
Not(see(rabbit, dog))
see(rabbit, bald_eagle)
see(rabbit, squirrel)
Not(see(dog, rabbit))
Not(visit(rabbit, bald_eagle))
Not(chase(dog, bald_eagle))

solve(visit(bald_eagle, dog))
```

---

### PROGLM adapted from [6]

---

Here are some facts and rules:

If someone visits the squirrel and the squirrel visits the rabbit then they are round..... (omitted, see above)

Q: The statement "The bald eagle visits the dog" is True or False?

```
# the bald eagle is red.
bald_eagle_is_red = True
# If someone is red then they visit the squirrel.
bald_eagle_visits_squirrel = bald_eagle_is_red
# If someone visits the squirrel then the squirrel visits the rabbit.
squirrel_visits_rabbit = bald_eagle_visits_squirrel
# If someone visits the squirrel and the squirrel visits the rabbit then they are round.
bald_eagle_is_round = bald_eagle_visits_squirrel and squirrel_visits_rabbit
# If someone is round then they chase the rabbit.
bald_eagle_chases_rabbit = bald_eagle_is_round
# If someone is red and they chase the rabbit then they visit the dog.
bald_eagle_visits_dog = bald_eagle_is_red and bald_eagle_chases_rabbit
# Question: The statement "The bald eagle visits the dog" is True or False?
return bald_eagle_visits_dog
```

---

Figure 14: Prompt (excerpt) used for PROOFWRITER.

## Prompts for SAT<sub>COTSOLVER</sub>

---

### SAT<sub>COTSOLVER</sub> for GSM

---

Q: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

```
jason_lollipops_initial = 20
lollipops_given = Variable()
jason_lollipops_after = 12
jason_lollipops_after = jason_lollipops_initial - lollipops_given
result = lollipops_given
solve(result)
# Solve the value of result
# We know jason_lollipops_after = jason_lollipops_initial - lollipops_given
# So lollipops_given = jason_lollipops_initial - jason_lollipops_after = 20 - 12 = 8
# result = lollipops_given = 8
# The answer is 8
```

---

### SAT<sub>COTSOLVER</sub> for CLUTRR

---

Dorothy took her daughter Michelle and her mother Gabrielle car shopping.

Q: How is [Michelle] related to [Gabrielle]?

```
# [Dorothy] took her daughter [Michelle] and her mother [Gabrielle] car shopping.
```

```
relation(Dorothy, Michelle) = (mother, daughter)
```

```
relation(Dorothy, Gabrielle) = (daughter, mother)
```

```
# How is [Michelle] related to [Gabrielle]?
```

```
solve(relation(Michelle, Gabrielle))
```

```
# Solve the problem
```

```
# Gabrielle's daughter is Dorothy, and Dorothy's daughter is Michelle, so Gabrielle's granddaughter is Michelle.
```

```
# The answer is granddaughter.
```

---

Figure 15: Prompt (excerpt) used for SAT<sub>COTSOLVER</sub>.