PENCIL: LONG THOUGHTS WITH SHORT MEMORY

Chenxiao Yang, Nathan Srebro, David McAllester & Zhiyuan Li
Toyota Technological Institute at Chicago
{chenxiao,nati,mcallester,zhiyuanli}@ttic.edu

ABSTRACT

While recent works (e.g. o1, DeepSeek R1) have shown great promise of using long Chain-of-Thought (CoT) at test-time to improve reasoning capabilities of language models, it is challenging to scale up due to inefficient memory usage — intermediate computations accumulate indefinitely in the context even no longer needed for generating future thoughts. We propose PENCIL to address this limitation, which incorporates a reduction mechanism into the autoregressive generation process, allowing the model to recursively clean up intermediate thoughts in ways learned from training. With the reduction mechanism, the maximal context length during generation can decrease from the time complexity for solving the problem, which is often exponential for inherently hard tasks, to the actual space required, which is often polynomial. By using space efficiently, PENCIL can generate longer thoughts using small memory and thus solve larger-scale problems with more inference time. For example, we show PENCIL achieves almost perfect accuracy on the challenging Einstein's puzzle using a small 25M-transformer with 2048 context length. Theoretically, we show PENCIL can perform universal space-efficient computation by simulating Turing machines with optimal time and space complexity.

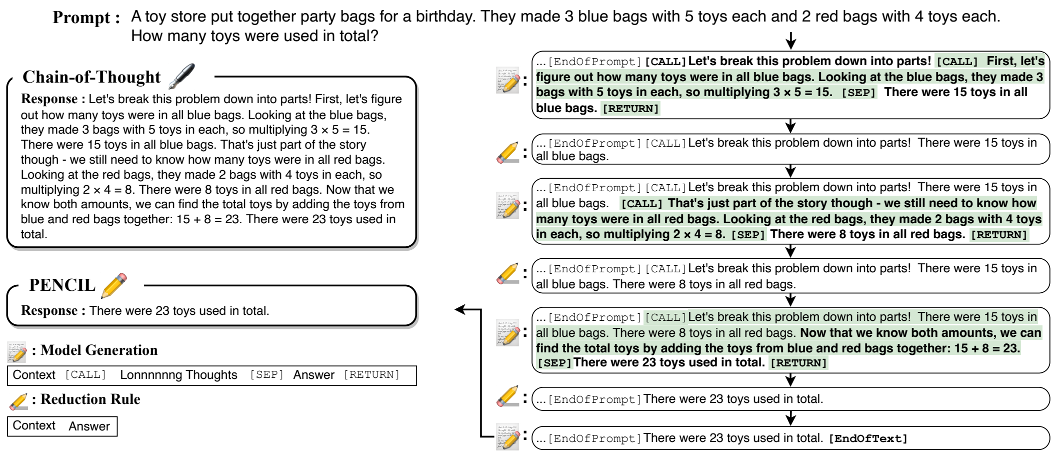
1 INTRODUCTION

Recently, there has been a surge of interest in reasoning with *Chain-of-Thought* (CoT) (Wei et al., 2022) and generating longer thoughts at test-time to tackle larger-scale and more complicated problems (OpenAI, 2024; Guo et al., 2025; Snell et al., 2024; Muennighoff et al., 2025). CoT is an iterative generation process: each intermediate reasoning step is appended to the current context and treated as the input in subsequent reasoning. The context grows until reaching a final answer. While such an iterative model is theoretically powerful – capable, in principle, of tackling many intricate problems given unlimited length (Merrill & Sabharwal, 2023; Feng et al., 2024; Li et al., 2024b) – it suffers from the inherent *write-only* limitation: partial computation remains in the context even when no longer needed for future thought generation. This design becomes particularly problematic for inherently hard reasoning tasks, where no efficient algorithm exists and thus reasoning inevitably spans many steps, forcing the context length to grow indefinitely. This not only demands excessive memory resources that become impractical for computationally hard tasks, but could also degrades the model's ability to effectively retrieve information in the context, even when the maximum length is not exceeded (Liu et al., 2024).

Memory management is a major issue in modern computer systems. Turing machines, for example, can overwrite tape cells and reclaim space for new computations, while high-level programming languages rely on stack frames, function calls, and garbage collection to discard unneeded data. While some previous works have attempted to augment LLMs with external memory (e.g. (Gao et al., 2023; Wang et al., 2024)), they often lack a direct mechanism for reclamation of no longer needed memory as stack deallocation or garbage collection. This paper proposes **PENCIL**, [1] which introduces cleaning mechanisms to CoT for space-efficient and long-chain reasoning.

In a nutshell, PENCIL combines a next-token generator (e.g., a decoder-only transformer) and a *reduction rule*, and applies the reduction rule whenever possible throughout the standard iterative next-token generation process to reduce context length. In this paper, we focus on a simple yet

[1] **P**ENCIL **EN**ables **C**ontext-efficient **I**nference and **L**earning



(a) PENCIL (b) Internalized thoughts of PENCIL

Figure 1: A toy example illustrating how PENCIL would potentially solve an arithmetic problem. Bold text indicates content generated in the current iteration, content highlighted in blue indicates intermediate thoughts to be erased by the reduction rule. See a concrete example of the complete thinking process for solving QBF in Fig. 5, and an illustration for Einstein's puzzle in Fig. 6. All details are included in Appendix G.

universal reduction rule motivated by the function call stack in modern computers.

$$\mathbf{C}\ \texttt{[CALL]}\ \mathbf{T}\ \texttt{[SEP]}\ \mathbf{A}\ \texttt{[RETURN]}\ \Rightarrow\ \mathbf{C}\ \mathbf{A} \tag{1}$$

where [CALL], [SEP], and [RETURN] are special tokens that separate the context (**C**), thoughts (**T**), and answer (**A**) in the sequence. Once a computation completes (marked by [RETURN]), all intermediate reasoning steps (those between [CALL] and [SEP]) will be removed, merging the answer back into the context. Importantly, this process can be applied recursively, allowing for hierarchical reasoning structures similar to nested function calls in programming. PENCIL alternates between standard CoT-style generation and this reduction step, automatically discarding unneeded thoughts based on patterns learned from training. Figure 1 gives a hypothetical example of how PENCIL might be applied to natural language thoughts.

We train and evaluate PENCIL on SAT, QBF, and Einstein's puzzle — tasks that inherently require exponential computation time. PENCIL effectively reduces the maximal CoT length (i.e. the space requirement) from exponential to polynomial. Consequently, under fixed architecture and context window, PENCIL allows solving larger-sized problems whereas CoT fails due to exploding context length. Furthermore, by continually discarding irrelevant tokens, PENCIL can significantly save training computes and converge faster even when memory or expressiveness is not a bottleneck. Notably, on the 5×5 Einstein puzzle – a challenging natural-language logic puzzle that even large models like GPT-4 struggle with – PENCIL achieves a 97% success rate by using a small transformer with 25M-parameter and 2048-token context.

See discussions about related work in Appendix E.

2 PENCIL: Iterative Generation & Reduction

Chain-of-Thought (CoT) (Wei et al., 2022) allows language models to generate intermediate reasoning steps before producing a final answer. Formally, given a finite alphabet Σ, let $\pi : \Sigma^* \to \Sigma$ be a *next-token predictor* with parameters θ, which maps an input sequence $(x_1, x_2, \cdots, x_n) \in \Sigma^n$ to the next token $x_{n+1} \in \Sigma$. Correspondingly, we can define a sequence-to-sequence mapping $f : \Sigma^* \to \Sigma^*$ as

$$f(x_1, \ldots, x_n) \triangleq (x_1, \ldots, x_n, \pi(x_1, \ldots, x_n)) \tag{2}$$

which concatenates the next token to the current context. For brevity, we will write f instead of f when the context is clear. CoT with k steps is denoted as $f^k : \Sigma^* \to \Sigma^*$, where $f^k \triangleq f \circ f^{k-1}$ and $f^1 \triangleq f$. Given any input sequence $x = (x_1, x_2, \ldots, x_n) \in \Sigma^n$, each application of f extends the sequence by one token, such that $f^k(x) \in \Sigma^{n+k}$. Throughout this paper, we use shorthand $x_{:j}$ to denote $(x_1, \dots, x_j)$, and $x_{i:j}$ the subsequence from x_i to x_j.

However, the iterative generation process of CoT is inherently limited by its write-once nature; that is, once written, intermediate computations permanently occupy the context, regardless of their relevance in the subsequent reasoning steps. Consequently, the context length would eventually grow overwhelmingly large for complex reasoning problems. To address this, we introduce **PENCIL**, which is CoT equipped with a reduction rule that enables selective elimination of reasoning traces, allowing the model to scale to larger problems with less memory.

2.1 Reduction Rule and PENCIL

A *reduction rule* (a.k.a. rewriting rule) (Baader & Nipkow, 1998) is a formal mechanism originated from logic for transforming one expression to another via predefined patterns and ultimately reaching a final normal form, i.e. the answer. It serves as a fundamental model of computation in classic functional programming languages such as λ-calculus (O'Donnell, 1985), and proof assistants for automated theorem proving and reasoning (Wos et al., 1992). Mathematically, the reduction rule can be thought of as a unique sequence-to-sequence function $\phi : \Sigma^* \to \Sigma^*$, which in this paper is from a longer sequence $(x_1, \dots, x_a) \in \Sigma^a$ to a shorter one $(x_{i_1}, \dots, x_{i_b}) \in \Sigma^b$ where $b \leq a$.

The Reduction Rule Let $\hat{\Sigma} = \Sigma \cup \{ \texttt{[CALL]}, \texttt{[SEP]}, \texttt{[RETURN]} \}$ be the extended alphabet including three special tokens that indicate certain structures of the reasoning trace. Given the new alphabet, we can instantiate the rule ϕ as (1), where $\mathbf{C} \in (\Sigma \cup \{\texttt{[CALL]}, \texttt{[SEP]}, \texttt{[RETURN]}\})^*$, $\mathbf{T} \in (\Sigma \cup \{\texttt{[SEP]}, \texttt{[RETURN]}\})^*$, $\mathbf{A} \in (\Sigma \cup \texttt{[CALL]})^*$ are subsequences separated by the special tokens. The allowance of difference special tokens in **C**, **T**, **A** ensures that the `[SEP]` token in (1) is the one immediately before `[RETURN]`, and `[CALL]` is immediately before `[SEP]`, and thus the matching is unique.

Intuitively, **C** can be understood as context that can include information that is either directly relevant to solving the current problem or irrelevant but useful for solving future problems; **T** represents the intermediate thoughts for deriving the answer and **A** represents the answer. If the input sequence satisfy the pattern **C** `[CALL]` **T** `[SEP]` **A** `[RETURN]`, the rule will activate. Consequently, the entire intermediate thoughts and the special token triplet will be removed, with the answer being merged back into the context. Otherwise if the pattern is not satisfied, the rule will leave the input sequence unchanged.

PENCIL consists of a learnable next-token predictor f as defined in (2) which is responsible for generating the intermediate reasoning steps (including special tokens `[CALL]`, `[SEP]`, `[RETURN]`) as in the standard CoT, and the reduction rule ϕ as defined in (1) that serves to reduce the context and clean the memory. Formally, we define one step and k-steps of PENCIL as $\mathrm{PENCIL}^1 = \phi \circ f$ and $\mathrm{PENCIL}^k = (\phi \circ f)^k$. Namely, each step of PENCIL first generates the next token as in standard CoT and then applies the reduction rule ϕ, deleting the intermediate computations if the new sequence matches the pattern. PENCIL is defined as a sequence of mappings $\{\mathrm{PENCIL}^1, \mathrm{PENCIL}^2, \mathrm{PENCIL}^3, \ldots\}$, which produces the entire thinking process on input x.

2.2 Alternated Generation and Reduction Process

The iterative generation and reduction process of PENCIL can be formalized by grouping the f functions that are interleaved by ineffective reduction steps (where ϕ does not match the pattern):

$$\mathrm{PENCIL}^k = f^{k_{r+1}} \circ \phi \circ f^{k_r} \circ \phi \circ \cdots \circ \phi \circ f^{k_1} \tag{3}$$

where $k = \sum_{i=1}^{r+1} k_i$, and k_i denotes the number of tokens generated between the $(i-1)$-th and i-th effective reduction. Here r is the total number of effective reductions, assuming the model terminates with a `[EOS]` token indicating stop generation. This process alternates between two phases

$$\textbf{Generation} \quad x^{(i)} \triangleq f^{k_i} \circ \phi \cdots \phi \circ f^{k_1}(x_{:n}) \quad , \textbf{Reduction} \quad x^{(i-0.5)} \triangleq \phi \circ f^{k_{i-1}} \cdots \phi \circ f^{k_1}(x_{:n}) \tag{4}$$

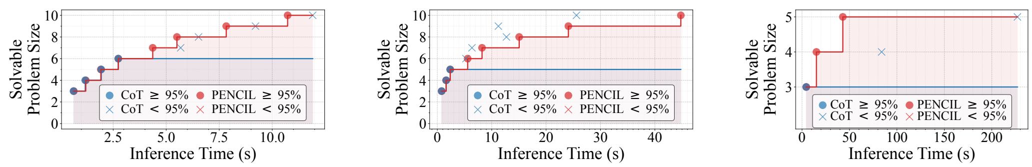
Each element in $\{x^{(1)}, x^{(2)}, \dots, x^{(r+1)}\}$ represents a generated sequence ending with `[RETURN]`, except for $x^{(r+1)}$ which ends with EOS. The set $\{x^{(0.5)}, x^{(1.5)}, \dots, x^{(r+0.5)}\}$ contains the reduced sequences after each effective reduction, with $x^{(0.5)} = x$ being the input prompt. The complete reasoning trace can be expressed as:

$$x \xrightarrow{f^{k_1}} x^{(1)} \xrightarrow{\phi} x^{(1.5)} \quad \cdots \quad x^{(r+0.5)} \xrightarrow{f^{k_{r+1}}} x^{(r+1)} \tag{5}$$

	$n =$	3	4	5	6	7	8	9	10
Baseline	Acc.	66	57	46	51	46	51	49	51
CoT	Acc.	100	100	100	99	84	63	54	50
	TR.	99.6	99.0	98.0	96.2	74.0	69.9	63.8	51.4
PENCIL	Acc.	100	100	100	99	99	100	100	100
	TR.	100	99.0	97.1	95.9	91.8	93.3	92.9	83.0

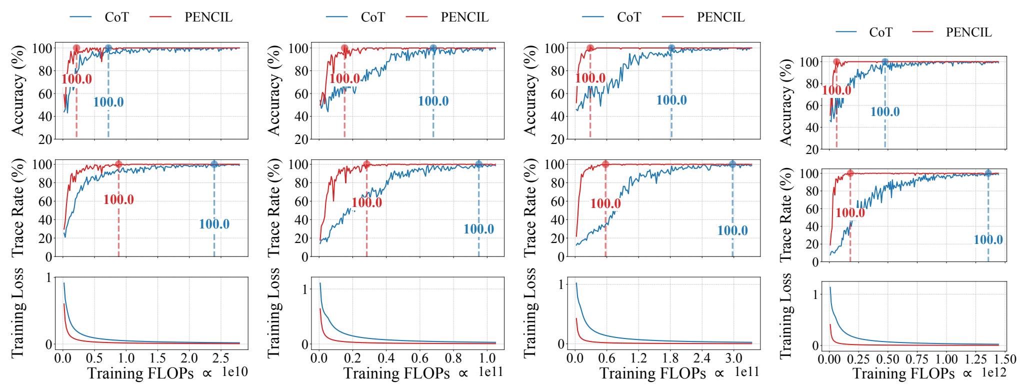
	$n =$	3	4	5	6	7	8	9	10
Baseline	Acc.	90	82	85	68	60	69	71	66
CoT	Acc.	100	100	97	94	74	72	69	73
	TR.	100	100	98.3	93.9	65.1	49.4	40.7	32.8
PENCIL	Acc.	100	100	100	100	100	100	100	100
	TR.	100	100	100	100	100	100	100	100

Table 1: Performance comparison on SAT and QBF. Acc denotes the Accuracy (%) and TR denotes the trace rate (%).



(a) SAT (b) QBF (c) Einstein's puzzle

Figure 2: Test-time scalability. Comparison of maximally solvable problem size (with $\geq$ 95% accuracy) given different inference time budgets.



(a) QBF $n = 3$ (b) QBF $n = 4$ (c) QBF $n = 5$ (d) QBF $n = 6$

Figure 3: Comparison of convergence speed on the QBF problem (with n ranges from 3 to 6). Circles and vertical lines indicate the first time each method reaches optimal performance.

That is, at each iteration i, PENCIL first generates from $x^{(i-0.5)}$, which serves as the prompt for this iteration, to $x^{(i)}$, a prompt-response pair, until producing `[RETURN]`; then PENCIL applies the reduction rule to transform the prompt-response pair $x^{(i)}$ into a new prompt $x^{(i+0.5)}$ for the next iteration $i + 1$.

See Appendix A for a discussion of how PENCIL can significantly reduce the space requirement, save computes during training, and enable solving larger-scale problems.

3 Thinking with PENCIL

In Appendix B we demonstrate how the reduction rule can be applied to several concrete computationally intensive problems (including SAT, QBF and Einstein's puzzle) and how PENCIL could potentially solve them space efficiently.

4 Experiments

We next conduct experiments on the problems mentioned in the last section to evaluate the effectiveness of PENCIL. Implementation and experimental details are deferred to Appendix C.

4.1 Results on SAT and QBF

Performance As shown in Table 1, both CoT and PENCIL significantly outperform the baseline (i.e. without using CoT) and achieve almost perfect performance ($\geq$ 95% accuracy) on small problems ($n \leq 6$ for SAT and 5 for QBF). While CoT's performance degrades sharply when problem size increases - dropping to 50% accuracy on SAT and 61% on QBF when $n = 10$, PENCIL maintains near-perfect accuracy across all problem sizes. Furthermore, PENCIL's consistently high trace rate

Puzzle Size		CoT	PENCIL
5×5	Accuracy (%)	25	97
	Trace Rate (%)	2.97	78.27
4×4	Accuracy (%)	34	100
	Trace Rate (%)	8.33	86.52
3×3	Accuracy (%)	99	99
	Trace Rate (%)	99.37	99.66

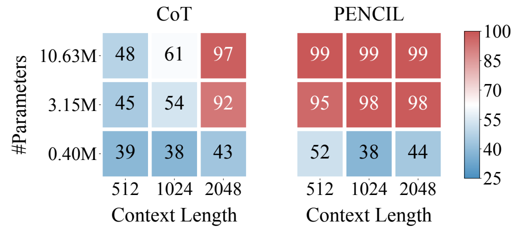


Figure 4: **(left)** Comparison of performance w/o and with the reduction rule on the Einstein's puzzle. **(right)** Effects of model size and context length on accuracy for solving 3×3 Einstein's puzzle.

(above 90% for most problem sizes) indicates that it precisely follows the intended algorithm's reasoning steps.

Test-Time Scalability Figure 2 compares the test-time scalability of CoT and PENCIL given different inference time budget. For both SAT and QBF problems, PENCIL can effectively solve larger problems with increased time budget, handling up to $n = 10$ with inference time around 10s and 40s respectively while CoT struggles to scale up even when given more time. This is because the reduction rule enables PENCIL to keep the reasoning length growing polynomially rather than exponentially with problem size, significantly reducing the requirement of space during generation.

Convergence Figure 3 compares the convergence speed of CoT and PENCIL on the QBF problem given fixed training FLOPs budget calculated based on (6). To isolate the impact of memory constraints, which limit the expressiveness of models, we allow unlimited context window length in this experiment, enabling both methods to potentially achieve perfect performance. Since since for larger problems CoT's space consumption becomes prohibitively large and will cause out-of-memory, we only report results for $n = 3$ to 6. The results show that PENCIL can effectively save computation, and thus can consistently achieve better performance under the same compute budget and converge faster, with the gap becoming more significant as problem size increases.

4.2 Results on Einstein's Puzzle

Besides of the original challenging 5×5 Einstein's puzzle, we also consider two simplified variants: 3×3, 4×4. For each size of the puzzle, we generate $10,000$ training instances by randomly assigning attributes to houses and deriving valid constraints that ensure a unique solution. The accuracy is evaluated based on whether the model can successfully answer the question "who owns the Fish" on 100 unseen validation samples.

Main Results Figure 4 (left) reports the performance with and without using the reduction rule to solve different sizes of Einstein's puzzles. Remarkably, PENCIL solves the original 5×5 puzzle at 97% accuracy using only 25.19M parameters (significantly smaller than GPT-2) and 2048 context length (the same as GPT-2), with average inference time per sample 42.98s. In comparison, CoT fails catastrophically on puzzles beyond 3×3, with accuracy dropping to 25% (i.e. close to random guessing) on 5×5 puzzles, despite using the same architecture and training.

Effects of Model Size As shown in Figure 4 (right), PENCIL achieves consistently high accuracy with sufficient model capacity (with $\geq$ 3.15M parameters, i.e. a 4-layer transformer) even with limited context length, while CoT requires both larger models and longer context to achieve comparable performance. However, when the model size is too small, both methods fail to solve the puzzle effectively, suggesting a minimum model capacity threshold.

5 Space-Efficient and Universal Simulation Power of PENCIL

Theoretically, we show that PENCIL could simulate Turing machine running in T steps and S space by generating $O(T)$ tokens with maximal sequence length $O(S)$, with a mildly powerful next-token generator, like linear-time generator, or even *decoder-only transformer*, as suggested by recent works (Pérez et al., 2021; Merrill & Sabharwal, 2023). This is impossible for the standard CoT. See detailed discussions in Appendix D.

6 CONCLUSION

This paper identifies a fundamental limitation of CoT where intermediate computations accumulate indefinitely in the context, and introduce PENCIL to address this. PENCIL adopts a simple reduction rule to "clean up" unneeded reasoning steps as soon as they are finalized. This mechanism effectively transforms long traces into compact representations, enabling efficient training and allowing the model to handle substantially larger problems under the same memory constraints. Extensive experiments are done to demonstrate the effectiveness of PENCIL to handle inherently challenging tasks with less computes and smaller memory.

REFERENCES

Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1998.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. In *The Eleventh International Conference on Learning Representations*, 2022.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36, 2024.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36, 2024.

Dan Friedman, Alexander Wettig, and Danqi Chen. Learning transformer programs. *Advances in Neural Information Processing Systems*, 36, 2024.

Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. Lazyllm: Dynamic token pruning for efficient long context llm inference. *arXiv preprint arXiv:2407.14057*, 2024.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Erik Garrison. Memory makes computation universal, remember? *arXiv preprint arXiv:2412.17794*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*, 2023.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 784–794, 2022.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024a.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*, 2024b.

David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Tom McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability. *Advances in Neural Information Processing Systems*, 36, 2024.

Chenxiao Liu, Shuai Lu, Weizhu Chen, Daxin Jiang, Alexey Svyatkovskiy, Shengyu Fu, Neel Sundaresan, and Nan Duan. Code execution with pre-trained language models. *arXiv preprint arXiv:2305.05383*, 2023.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

William Merrill and Ashish Sabharwal. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

William Merrill, Ashish Sabharwal, and Noah A Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856, 2022.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

Franz Nowak, Anej Svete, Alexandra Butoi, and Ryan Cotterell. On the representational capacity of neural language models with chain-of-thought reasoning. *arXiv preprint arXiv:2406.14197*, 2024.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Michael J O'Donnell. *Equational logic as a programming language*. Springer, 1985.

OpenAI. Learning to reason with llms, September 2024. URL `https://openai.com/index/learning-to-reason-with-llms/`.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021.

Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational intelligence*, 9(3):268–299, 1993.

Bilgehan Sel, Ahmad Al-Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. *arXiv preprint arXiv:2308.10379*, 2023.

Bart Selman, David G Mitchell, and Hector J Levesque. Generating hard satisfiability problems. *Artificial intelligence*, 81(1-2):17–29, 1996.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Mirac Suzgun and Adam Tauman Kalai. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pp. 11080–11090. PMLR, 2021.

Larry Wos, Ross Overbeek, Ewing Lusk, and Jim Boyle. *Automated reasoning introduction and applications*. McGraw-Hill, Inc., 1992.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Dylan Zhang, Curt Tigges, Zory Zhang, Stella Biderman, Maxim Raginsky, and Talia Ringer. Transformer-based models are not yet perfect at learning to emulate structural recursion. *arXiv preprint arXiv:2401.12947*, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.

CONTENTS

A COMPUTATIONAL BENEFITS OF PENCIL

Space Comparison To facilitate direct comparison, we define *scaffolded CoT* as the trace that would be produced by PENCIL but without actually removing the thoughts. (We call it scaffolded because it contains the special tokens that indicate the hierarchical reasoning structure.) Formally, for any input x, scaffolded CoT is defined as $(x \ , \ x^{(1)} \backslash x^{(0.5)} \ , \ \ldots \ , \ x^{(r+1)} \backslash x^{(r+0.5)})$, where $x^{(i)} \backslash x^{(i-0.5)}$ represents the tokens generated at iteration i. The maximal sequence length in PENCIL is $\max\{|x^{(1)}|, |x^{(2)}|, \ldots, |x^{(r+1)}|\}$, whereas the scaffolded CoT has a length of $n + k$ (or $n + k - 3r$ if excluding special tokens). As we will demonstrate in Sec. B, this difference becomes particularly significant for complex reasoning tasks where $n + k$ can grow exponentially while $\max\{|x^{(1)}|, |x^{(2)}|, \ldots, |x^{(r+1)}|\}$ is kept polynomial.

Computes Comparison Moreover, though the total number of predicted tokens remains unchanged with or without reduction, PENCIL achieves significant compute savings by maintaining a substantially shorter context for each prediction. Specifically, when using a standard causal-masking transformer with KV cache for storing key and value matrices, the training FLOPs of PENCIL required for a problem instance $x_{:n}$ is proportional to:

$$\begin{aligned} &\sum\nolimits_{i=1}^{r+1} \left(|x^{(i-0.5)}| + |x^{(i)}| + 1\right) \cdot \left(|x^{(i)}| - |x^{(i-0.5)}|\right) \ + \\ &\sum\nolimits_{i=1}^{r} \left(|x^{(i)} \cap x^{(i+0.5)}| + |x^{(i+0.5)}| + 1\right) \cdot |x^{(i+0.5)} \backslash x^{(i)}| \end{aligned} \tag{6}$$

where $x^{(i)} \cap x^{(i+0.5)}$ represents the shared context **C** before the `[CALL]` token, and $x^{(i+0.5)} \backslash x^{(i)}$ denotes the answer **A** between `[SEP]` and `[RETURN]` tokens. The first term accounts for model generation steps, while the second term captures the computation cost of reduction steps where KV cache must be recomputed for **A** after merging it back into the context. We will empirically quantify (6) in Sec. 4.

B THINKING WITH PENCIL

B.1 SAT AND QBF

SAT is a canonical NP-complete problem. We consider the 3-SAT variant, where each instance is a Boolean formula in conjunctive normal form with clauses of length three, e.g. $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$. The ratio between number of clauses and variables is set as 4.3, larger than the threshold 4.267 where instances are empirically hardest to solve and satisfiability probability transitions sharply from 1 to 0 (Selman et al., 1996). **QBF** is a PSPACE-complete problem that generalizes SAT by adding universal ($\forall$) and existential ($\exists$) quantifiers. Each instance is a quantified Boolean formula in Prenex normal form, e.g., $\exists x_1 \forall x_2 \exists x_3 : (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$. We set the probability of a variable being existentially quantified as 0.5.

We consider using the DPLL algorithm to solve the SAT problem, and solving the QBF problem by recursively handling quantifiers and trying variable values. The PENCIL reasoning traces are generated as we run the algorithm. Both algorithms recursively explore variable assignments by splitting on an unassigned variable x_i and trying branches $x_i = \texttt{True}$ and $x_i = \texttt{False}$. The reduction rule wraps each branch with `[CALL]`, `[SEP]` and `[RETURN]`, which creates a hierarchical binary tree structure. See Fig. 5.

Without the reduction rule, the context must retain the complete recursive trace — all partial assignments and intermediate formulas — leading to worst-case exponential space complexity $O(2^n)$. For PENCIL, once a branch returns, its intermediate reasoning steps are discarded, therefore search paths will be discarded, preserving only the final answer. This reduces the maximal length to $O(n)$, bounded by the search tree depth. As shown in Fig. 7, at $n = 10$, the maximal sequence length drops from $13,804$ to $2,507$ for SAT and from $151,661$ to 649 for QBF.

B.2 TAIL RECURSION AND EINSTEIN'S PUZZLE

Einstein's Puzzle We further consider Einstein's puzzle (Prosser, 1993), a classic constraint satisfaction problem where the model must learn to reason in natural language. Each problem

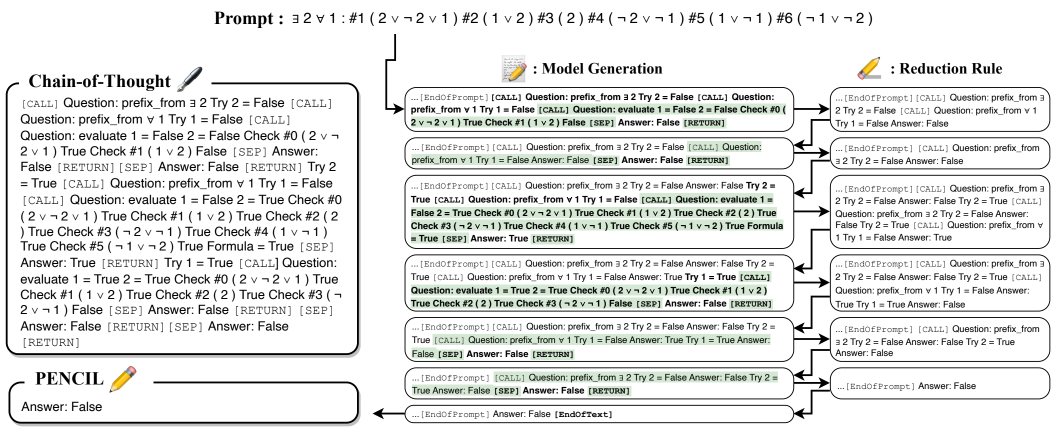


Figure 5: The complete thinking process of PENCIL on a small-sized QBF instance. The "..." at the beginning of a thought hides the prompt. Bold text represents newly generated thoughts, while blue highlights indicate thoughts to be removed.

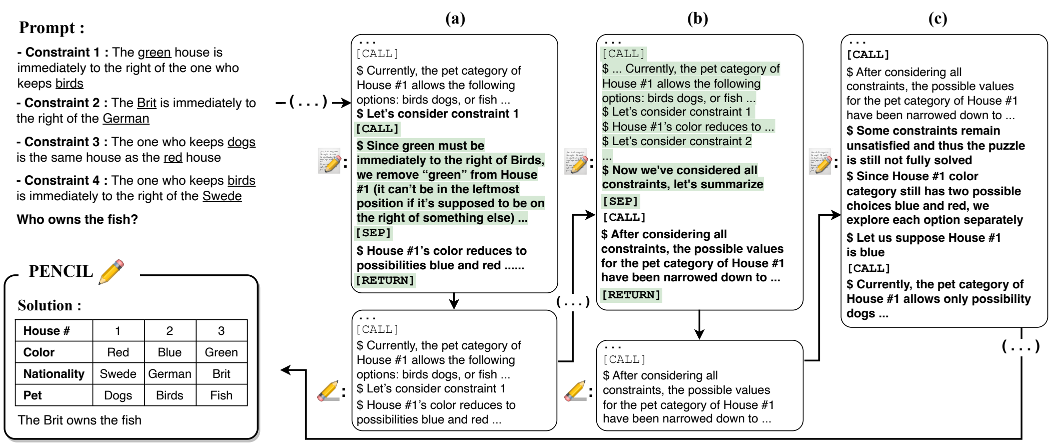


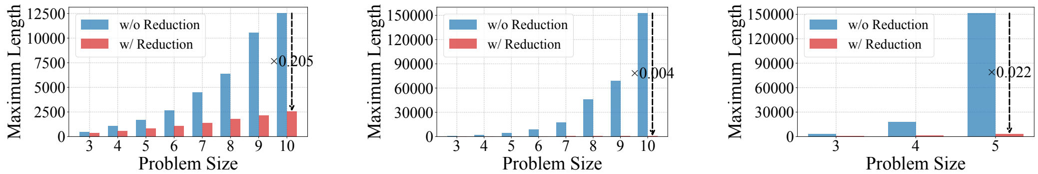
Figure 6: A simplified illustration of the algorithm for generating the thinking process for Einstein's puzzle (3×3). The puzzle requires determining attributes of each house (Color: Blue/Green/Red, Nationality: Brit/German/Swede, Pet: Birds/Dogs/Fish) given a set of constraints, with each house having unique attributes. The "..." in the arrow denotes omitted thoughts for conciseness; the "..." in the box denotes omitted thought. See the complete example in Appendix G.

instance consists of a list of houses with different attributes (e.g., color, nationality, pet), and given a set of constraints or clues as the prompt (e.g. the green house is immediately to the right of the one who keeps birds), the goal is to determine the attributes of each house through logical deduction. The original puzzle has size 5×5 (5 houses and 5 attribute categories, totaling 25 variables), which presents a significant challenge for language models to solve – even GPT-4 fails to solve it with few-shot CoT (Dziri et al., 2024).

Special Use Case: Tail Recursion A notable special case of the reduction rule is when the answer itself leads to another question: when $\mathbf{A} = \texttt{[CALL]}\ \mathbf{T'}$, (1) becomes

$$\begin{aligned} &\mathbf{C}\ \texttt{[CALL]}\ \mathbf{T}\ \texttt{[SEP]}\ \texttt{[CALL]}\ \mathbf{T'}\ \texttt{[RETURN]} \\ \Rightarrow\ &\mathbf{C}\ \texttt{[CALL]}\ \mathbf{T'}. \end{aligned} \tag{7}$$

We refer to this special use case as *tail recursion* since it mimics the tail recursion in functional programming where a function's returned value is another function call. A practical application of this rule is to simplify an originally complex question by iteratively reducing it, through some intermediate reasoning steps, to a more tractable form. In Sec. D we will use this to prove PENCIL's space efficiency.



(a) SAT (b) QBF (c) Einstein's puzzle

Figure 7: Maximal sequence length with and without the reduction rule.

See Fig. 6 for an illustration of how reduction rules can be applied to solve the Einstein puzzle, which consists of the following steps in one round of iteration: **(a)** Propagating constraints to eliminate impossible attributes combinations; **(b)** Use the tail recursion rule to merge results from constraints propagation and update the house states; **(c)** Iteratively explore different solution branches and discard intermediate reasoning steps from each branch, only preserving the final answer. As shown in Fig. 7, for 5×5 puzzle, the maximal sequence reduces dramatically from $151,192$ to $3,335$ (without tail recursion this number is $7,705$).

C EXPERIMENTAL DETAILS

Training The training of PENCIL is nearly identical to that of CoT with a key difference being how the data is processed. Specifically, the training pipeline of PENCIL consists of the following steps:

For data preparation, we implement the algorithms for solving the problems mentioned in Sec. B, generates the corresponding scaffolded CoT with special tokens `[CALL]`, `[SEP]`, `[RETURN]` as we run the algorithm, and then transform the long scaffolded CoT sequence into a set of smaller sequences $\{x^{(1)}, x^{(2)}, \ldots, x^{(r+1)}\}$ that ends with either `[RETURN]` or EOS.

During training, the loss function is crucial for the success of training PENCIL. In particular, we need not compute loss on every single token in each shorter sequence $x^{(i)}$, but only those that are generated starting from last iteration's reduction step (i.e. $x^{(i)} \backslash x^{(i-0.5)}$). We maintain an index for each $x^{(i)}$ for storing the information of the index where the model generation starts. We can either feed all shorter sequences into one batch (which is our default choice in experiments), which makes it possible to reuse the KV cache of other sequences to reduce training computes, or randomly sample from these sequences from all problem instance, which would lead to similar performance.

Implementation Unless otherwise stated, for model architecture, we choose a 6-layer transformer with 10.63M parameters for SAT and QBF problems, and an 8-layer transformer with 25.19M parameters for the more complex Einstein's puzzle. All experiments use a context window of 2048 tokens and rotary positional encoding (Su et al., 2024); we truncate the sequence to the maximal context window to fit into the model for all methods if it exceeds the model's capacity. We use the same batch size and learning rate for all methods across experiments.

Experimental Setting We adopt the online learning setting where models train until convergence with unconstrained data access, mirroring the common scenarios in language model training where data can be effectively infinite (Hoffmann et al., 2022). To ensure fair comparison, we include special tokens in the CoT, which might benefit its training by introducing additional structural information.

Evaluation Protocol We evaluate on a held-out validation set of 100 problem instances using two metrics: accuracy (percentage of correct predictions) and trace rate (percentage of reasoning steps matching the ground truth). For all problems, the labels for different classes are balanced.

D SPACE-EFFICIENT AND UNIVERSAL SIMULATION POWER OF PENCIL

In this section, we will theoretically analyze the expressive power of PENCIL by showing it can simulate the Turing machine space-efficiently with the maximal context length proportional to the space complexity. We first define the iterative next-token generator as a general purpose computation model, which subsumes Turing machine as an example.

Definition 1 (Iterative Next-token Generator)**.** *An* iterative next-token generator *is a tuple* $\mathcal{M} = (\Sigma, \pi, \Sigma_{accept}, \Sigma_{reject})$*, where* Σ *is a finite alphabet and* $\pi : \Sigma^* \to \Sigma$ *is a function that generates the* *next token based on the current context, and $\Sigma_{accept}, \Sigma_{reject} \subseteq \Sigma$ are disjoint sets of accepting and rejecting tokens respectively.*

For any input $x \in \Sigma^*$, the iterative next-token generator $\mathcal{M}$ recursively generates a sequence of tokens $\pi^k(x)$ and append each of them to the current sequence before the next generation until it outputs a token in Σ_{accept} or Σ_{reject}. We define the output of iterative next-token generator $\mathcal{M}$ on input x as 1 if it outputs a token in Σ_{accept} and 0 otherwise.

***Time complexity** $T(\mathcal{M}, x)$ is defined as the number of steps the iterative next-token generator $\mathcal{M}$ takes to halt on input x. We define $T(\mathcal{M}, x) = \infty$ if it does not halt.*

The gap between required space and time for a computation occurs when one can summarize the past computational trace into a shorter *state*. The summarization is required to be complete, that is, it only deletes useless information from computational trace and preserves all the tokens generated in the future.

Definition 2 (State of Iterative Next-token Generator)**.** *We say a function $s : \Sigma^* \to \Sigma^*$ is a* state function *of an iterative next-token generator $\mathcal{M} = (\Sigma, \pi, \Sigma_{accept}, \Sigma_{reject})$ if 1) $\pi \circ s = \pi$; 2) for all $x, x', y \in \Sigma^*$, $s(x) = s(x') \Longrightarrow s((x, y)) = s((x', y))$; 3) $s^2 = s$.*

Note the above definition of state function easily implies that the future trace of the iterative next-token generator $\mathcal{M}$, i.e. $\pi^k(x)$ for $k = 1, 2, \ldots$, can be uniquely determined by the state function s of $\mathcal{M}$. (See equation 1 below)

Lemma 1. *Let s be a state function of an iterative next-token generator $\mathcal{M} = (\Sigma, \pi, \Sigma_{accept}, \Sigma_{reject})$. It holds that $s \circ f_\pi^k \circ s = s \circ f_\pi^k$ and that $\pi^{k+1} = \pi^{k+1} \circ s$ for any $k \geq 0$.*

Given $\mathcal{M}$ and state function s, we define the **space complexity** of $(\mathcal{M}, s)$ on input x, $S(\mathcal{M}, s, x)$ as the maximal length of the states $(s \circ f_\pi)^k(x)$ for all steps k. Note by equation 2 and equation 1, $S(\mathcal{M}, s, x)$ is also equal to the maximal length $s(f_\pi^k(x))$ for all steps k. In other words, s defines a equivalent class over all possible computational traces of $\mathcal{M}$, and the mapping $x \mapsto s(x)$ erases unnecessary information in the trace of $\mathcal{M}$ to save space.

In Appendix F.1, we present the details of Turing machine as an iterative next-token generator and its state function, where each next token is one step simulation of Turing machine and the size of state is exactly the tape length.

SCROLL Rule Now we introduce a simplified version of PENCIL, SCROLL, which is already powerful enough to simulate general target iterative next-token generators with optimal time and space efficiency

$$\text{SCROLL}: \quad \mathbf{T}\ \texttt{[SEP]}\ \mathbf{T'}\ \texttt{[RETURN]} \ \Rightarrow\ \mathbf{T'} \tag{8}$$

More concretely, SCROLL uses one less special token than PENCIL, `[CALL]`, and can be simulated by PENCIL using tail recursion equation 7. (To simulate SCROLL with PENCIL, generator with PENCIL ensures its input starts with `[CALL]`, and outputs the same token as that with SCROLL. The only exception is, when later outputs `[SEP]`, the generator with PENCIL must output `[SEP][CALL]` consecutively.)

Next we will see how SCROLL can be used to simulate a target iterative next-token generator with optimal time and space efficiency. For now we assume our base next-token generator f is powerful enough so we just focus on designing the possible computational trace of SCROLL. We start with a simple solution with optimal space but suboptimal time efficiency to illustrate how SCROLL saves space.

A Space-Efficient but Time-Inefficient Solution Using the same notation in equation 4, we describe the trace of SCROLL as follows: $\forall i \in \mathbb{N}$,

$$x^{(i)} = \left(f_\pi \circ s \circ f_\pi^i(x), \texttt{[SEP]}, s \circ f_\pi^{i+1}(x), \texttt{[RETURN]}\right)$$

In other words, each time SCROLL reduction applies, it finishes one step of simulation of the target iterative next-token generator. The space complexity of SCROLL is twice the maximal length of the states $s(f_\pi^i(x))$ for all steps $0 \leq i \leq T(\mathcal{M}, x)$, that is, $S(\mathcal{M}, s, x)$, which is optimal up to a multiplicative constant. However, the time complexity of SCROLL is suboptimal, $T(\mathcal{M}, x) \cdot S(\mathcal{M}, s, x)$.

The issue above is that we apply reduction too often such that most of the steps of our base generator is spent on applying the reduction rule, instead of generating new tokens. To address this issue, we introduce a trick called *efficient garbage collection*, which only applies the reduction rule when the length of the state is smaller than half of the sequence length before the reduction.

The Efficient Solution in Both Space and Time. We define t_i recursively with $t_0 = 0$. For $i \in \mathbb{N}^+$, we define t^i as the smallest integer larger than t_{i-1} such that length of the state $s \circ f_\pi^{t_i}(x)$ is *no more than half of the sequence length* of $f_\pi^{t_i - t_{i-1}} \circ s \circ f_\pi^{t_{i-1}}(x)$. (note they are equal at t_{i-1}) $\forall i \in \mathbb{N}$, we define the trace of SCROLL as follows:

$$x^{(i)} = \left(f_\pi^{t_i - t_{i-1}} \circ s \circ f_\pi^{t_{i-1}}(x), \texttt{[SEP]}, s \circ f_\pi^{t_i}(x), \texttt{[RETURN]}\right)$$

Theorem 2. *SCROLL can simulate the target iterative next-token generator $\mathcal{M}$ by using $O(T(\mathcal{M}, x))$ total tokens and a maximal sequence length of $O(S(\mathcal{M}, s, x))$, provided the next token generator of SCROLL powerful enough to simulate the next-token generator in $\mathcal{M}$ and the state function s. So does PENCIL.*

Proof of equation 2 are deferred to Appendix F.2. We want to comment that for simulating the Turing machine, using the construction in equation F.1, a linear-time next token generator (time spent before generating the next token is proportional to its input length) suffices. Recent literature (Pérez et al., 2021; Merrill & Sabharwal, 2023) suggest that decoder-only transformers with average hard attention, which are also linear time, are capable of simulating next-token generator. By a similar argument, one could in principle show that they can also simulate the state function in the above design. With this, we conclude that **PENCIL can perform universal computation by simulating any Turing machine using same amount of time and space, with a mildly powerful next-token generator —** linear time generator for sure suffices, and recent study suggests even transformer would work.

E Related Work

Structured Reasoning A key distinction of scaffolded reasoning approaches stems from how space is managed during generation. At one extreme, Chain-of-Thought (Wei et al., 2022; Nye et al., 2021; Kojima et al., 2022) demonstrates that explicit intermediate steps can dramatically improve performance on complex problems, but at the expense of unbounded context growth. This limitation has motivated approaches leveraging reasoning structures such as trees and graphs (Yao et al., 2024; Long, 2023; Besta et al., 2024; Sel et al., 2023; Chen et al., 2022), adopting task decomposition strategies (Zhou et al., 2022; Drozdov et al., 2022; Khot et al., 2022) or some other prompting frameworks (Zelikman et al., 2022; Madaan et al., 2024; Suzgun & Kalai, 2024). While these methods enable more complex reasoning patterns, they require carefully crafted prompts and multiple rounds of interactions, whereas our approach achieves structured reasoning through end-to-end training.

Test-Time Scaling Extensive work has focused on addressing the computational bottlenecks of transformer architectures, particularly during long-context inference. One line of research explores architectural innovations through sparse and local attention patterns (Beltagy et al., 2020; Kitaev et al., 2020; Zaheer et al., 2020; Choromanski et al., 2020), while another focuses on memory optimization via KV-cache reduction (Zhang et al., 2023; Fu et al., 2024; Li et al., 2024a; Nawrot et al., 2024) and strategic context pruning (Kim et al., 2022; Jiang et al., 2023). However, these approaches still rely on next-token prediction that fundamentally treats the context window as append-only storage, leading to inherently inefficient space utilization.

LLMs as Programming Language Recent work has also explored intersections between programming languages and LLMs. For example, Weiss et al. (2021) proposes a language called RASP, programs in which can be encoded into and learned by transformers (Lindner et al., 2024; Friedman et al., 2024; Zhou et al., 2023). Liu et al. (2023) empirically shows that language models can be pre-trained to predict the execution traces of Python code. The reduction rule introduced in this work draws inspiration from term rewriting systems (Baader & Nipkow, 1998), a foundational means of computation in functional programming. This enables language models to explicitly emulate recursion that is otherwise hard to learn (Zhang et al., 2024), and manage space efficiently by erasing irrelevant contents in memory and focusing attention on those that are useful.

Computational Power / Limitation of CoT While transformers can theoretically simulate Turing machines (Pérez et al., 2021; Merrill & Sabharwal, 2023; Strobl et al., 2024; Nowak et al., 2024) with CoT, their practical computational power is fundamentally constrained by context window limitations. Particularly, we show that even with CoT, transformers with inherent space constraints would fail to handle problems requiring extensive intermediate computation. This parallels classical space-bounded computation theory, where memory management is crucial for algorithmic capabilities (Arora & Barak, 2009; Garrison, 2024). Our approach addresses this limitation by enabling more efficient use of the context.

F Additional Theoretical Results

F.1 Details on Turing machine as a iterative next-token generator and its state function

Definition 3 (Turing Machine)**.** *A Turing machine is a 7-tuple* $\mathsf{TM} = (\Gamma, b, Q, q_0, \delta, Q_{accept}, Q_{reject})$ *where:*

- Γ *is a finite set of the tape alphabet symbols*
- $b \in \Gamma$ *is the blank symbol*
- Q *is a finite set of states*
- $q_0 \in Q$ *is the initial state*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times L, R$ *is the transition function*
- $Q_{accept} \subseteq Q$ *is the set of accepting states*
- $Q_{reject} \subseteq Q$ *is the set of rejecting states*

where $Q_{accept} \cap Q_{reject} = \emptyset$ *(accepting and rejecting states are disjoint).*

Example 1 (Turing Machine as Iterative Next-token Generator)**.** *Given a Turing machine* $\mathsf{TM} = (\Gamma, b, Q, q_0, \delta, Q_{accept}, Q_{reject})$ *(see definition in 3), we construct an iterative next-token generator* $\mathcal{M} = (\Sigma, \pi, \Sigma_{\mathit{accept}}, \Sigma_{\mathit{reject}})$ *as follows:*

- *Let* $\Sigma = Q \times \Gamma \times \{L, R, N\}$ *be the set of tokens, where each token represents a tuple* (q, a, d) *of the symbol written* (a)*, the resulting state* (q)*, and the direction of head movement* (d)*.*
- *Define* $\Sigma_{\mathit{accept}} = Q_{accept} \times \Gamma \times \{L, R, N\}$ *and* $\Sigma_{\mathit{reject}} = Q_{reject} \times \Gamma \times \{L, R, N\}$ *to be the accepting and rejecting token sets respectively.*
- *The function* $\pi : \Sigma^* \to \Sigma$ *operates as follows on input context* $x = (q_1, a_1, d_1), \ldots, (q_k, a_k, d_k)$*:*
  1. *recover Turing machine state*
     - *The current state is* q_k *from the last token*
     - *The head position* p *can be computed by starting at 0 and following the directions* $d_1, \ldots, d_k$
     - *The tape contents can be reconstructed by applying the written symbols* $a_1, \ldots, a_k$ *at their respective positions*
  2. *Read the symbol* a *at the current head position* p *on the reconstructed tape*
  3. *If* $q_k \in Q_{accept} \cup Q_{reject}$*, halt*
  4. *Otherwise, compute* $\delta(q_k, a) = (q', a', d')$ *and output the token* (q', a', d')

Note that this construction preserves both the time and space complexity of the original Turing machine. Each step of the Turing machine corresponds to exactly one token generation, and the tape contents can be efficiently reconstructed from the computation history. This is the standard construction used to show transformer can simulate Turing machine (Pérez et al., 2021; Merrill et al., 2022).

Example 2 (State function for Iterative Next-Token Generator Simulating Turing Machine)**.** *Given a Turing machine* $\mathsf{TM} = (\Gamma, b, Q, q_0, \delta, Q_{accept}, Q_{reject})$ *and its corresponding iterative next-token generator* $\mathcal{M} = (\Sigma, \pi, \Sigma_{\mathit{accept}}, \Sigma_{\mathit{reject}})$*, where:*

- $\Sigma = Q \times \Gamma \times L, R, N$ *represents tokens of form* (q, a, d) *where* q *is the resulting state,* a *is the symbol written, and* d *is the direction of head movement*
- $\Sigma_{\mathit{accept}} = Q_{accept} \times \Gamma \times L, R, N$ *and* $\Sigma_{\mathit{reject}} = Q_{reject} \times \Gamma \times L, R, N$

We define a state function $s : \Sigma^* \to \Sigma^*$ *as follows: For input context* $x \in \Sigma^*$ *consisting of tokens* $(q_1, a_1, d_1), \ldots, (q_k, a_k, d_k)$*:*

1. *First, reconstruct the tape contents and head position from x:*
   - *Initialize an empty tape T with all cells containing blank symbol b*
   - *Start with head position $p_0 = 0$*
   - *For each token (q_i, a_i, d_i) in order:*
     - *Write symbol a_i at position p_{i-1} in T*
     - *Update position: $p_i = p_{i-1} + 1$ if $d_i = R$, $p_i = p_{i-1} - 1$ if $d_i = L$, $p_i = p_{i-1}$ if $d_i = N$*
2. *Let $[l, r]$ be the minimal interval containing:*
   - *The current head position p_k*
   - *All non-blank symbols in tape T*
3. *The state function $s(x)$ outputs a sequence of tokens that reconstructs the minimal state using a two-pass approach. Let $T[l : r]$ denote the contents of tape T in interval $[l, r]$. The output sequence is constructed as follows:*

Algorithm 1: Construction of Minimal Token Sequence (Two-Pass)

Input : *Tape contents $T[l : r]$, final state q_k, target head position p_k*
Output : *Sequence of tokens S*

```
S ← empty sequence; /* First pass:  Move right, writing all symbols
    */
for i ← l to r do
 | Append (q_k, T[i], R) to S;
end
/* Second pass:  Move left to target position                  */
i ← 0; steps_left ← r − p_k ;                        // Distance to target
for i ← r to p_k do
 | Append (q_k, T[i], R) to S;
end
return S;
```

This improved construction guarantees that:

- *First pass writes all required symbols by moving right*
- *Second pass moves to the target position p_k by moving left*
- *The final state q_k is preserved in the last token*
- *Only the minimal necessary information is retained*

The sequence satisfies the required properties as before:

***Next-token preservation**: $\pi \circ s = \pi$ since:*

- *The first pass writes all symbols in their correct positions*
- *The second pass moves to the correct head position*
- *The final state q_k is preserved in the last token*

***Future trace preservation**: If $s(x) = s(x')$ then $s((x, y)) = s((x', y))$ since:*

- *Two traces with same s output have identical configurations*
- *Future computation depends only on the current configuration*

***Idempotence**: $s^2 = s$ since:*

- *$s(x)$ already produces minimal representation*
- *Applying s again yields same minimal representation*

The space complexity $S(\mathcal{M}, s, x)$ matches the space used by the original Turing machine, as $s(x)$ retains only the minimal interval containing non-blank symbols and the head position.

F.2 Missing proofs

Proof of 1. For any $z \in \Sigma^*$, we have that $s^2(z) = s(z)$. Now let $x = s(z), x' = z$ and $y = \pi(z) = \pi(s(z))$, since $s(x) = s(x')$, we have $s((x, y)) = s((x', y))$, which further implies that $s(f_\pi(s(z))) = s((x, y)) = s((x', y)) = s(f_\pi(z))$. Therefore, $s \circ f_\pi \circ s = s \circ f_\pi$. Now we use induction to prove that $s \circ f_\pi^k \circ s = s \circ f_\pi^k$ for all $k \in \mathbb{N}^+$. The base case $k = 1$ is already proved. Now suppose $s \circ f_\pi^k \circ s = s \circ f_\pi^k$, we have $s \circ f_\pi^{k+1} \circ s = s \circ f_\pi \circ f_\pi^k \circ s = s \circ f_\pi \circ s \circ f_\pi^k \circ s = s \circ f_\pi \circ s \circ f_\pi^k = s \circ f_\pi \circ f_\pi^k$, which completes the induction. The second part of the lemma is trivial by the definition of state function.

Now we turn to the second part, which is a simple consequence of the first part. Note that for $k \geq 1$, $\pi^k = \pi \circ f_\pi^{k-1} = \pi \circ s \circ f_\pi^{k-1}$. By first part, $s \circ f_\pi^{k-1} = s \circ f_\pi^{k-1} \circ s$. This completes the proof of the second part. $\square$

Proof of 2. The maximal sequence length must be attained when SCROLL outputs `[RETURN]` and before applies the reduction rule. By definition of t_i, it holds that the length of the state $s \circ f_\pi^{t_i - 1}(x)$ is more than half of the sequence length of $f_\pi^{t_i - 1 - t_{i-1}} \circ s \circ f_\pi^{t_{i-1}}(x)$. Therefore, the sequence length at t_i is no more than $2 \cdot |s \circ f_\pi^{t_i - 1}(x)| + 2 + |s \circ f_\pi^{t_i}(x)| + 2 = O(S(\mathcal{M}, s, x))$.

The total tokens generated by SCROLL is upper bounded by the sum of length of $x^{(i)}$, which is $\sum_{i \geq 1}^{I} (t_i - t_{i-1} + |s \circ f_\pi^{t_i}(x)| + |s \circ f_\pi^{t_{i+1}}(x)| + 2)$. By definition of t_i, it holds that

$$t_i - t_{i-1} + |s \circ f_\pi^{t_i}(x)| \geq 2 \cdot |s \circ f_\pi^{t_{i-1}}(x)|, \tag{9}$$

Telescoping 9 from $i = 1$ to $I - 1$, we have that

$$t_I - t_0 + |s \circ f_\pi^{t_I}(x)| \geq \sum_{i=1}^{I-1} |s \circ f_\pi^{t_{i-1}}(x)|. \tag{10}$$

Thus we conclude the total length is at most $3t_I + 2I + 2|s \circ f_\pi^{t_I}(x)|$, which is at most $5T(\mathcal{M}, x) + 2M(\mathcal{M}, s, x) = O(T(\mathcal{M}, x))$, which completes the proof. $\square$

G Detailed Examples

G.1 Boolean Satisfiability (SAT)

Chain-of-Thought (w/o Reduction Rule)

- **Prompt:**

```
<|startoftext|> ( 4 ∨ ¬ 3 ∨ ¬ 2 ) ∧ ( ¬ 4 ∨ ¬ 2 ∨ 1 ) ∧ ( ¬ 2 ∨ ¬ 1 ∨ 3 ) ∧ ( 3 ∨
    ↪ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 2 ∨ 4 ) ∧ ( ¬ 1 ∨ 4 ∨ 2 ) ∧ ( 1 ∨ ¬ 4 ∨ 2 ) ∧ ( 4 ∨ ¬ 2 ∨
    ↪  3 ) ∧ ( ¬ 4 ∨ 1 ∨ ¬ 3 ) ∧ ( 3 ∨ 2 ∨ 4 ) ∧ ( ¬ 3 ∨ 1 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ 2 ∨
    ↪ ¬ 3 ) ∧ ( 2 ∨ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 1 ∨ ¬ 4 ) ∧ ( ¬ 1 ∨ ¬ 4 ∨
    ↪  ¬ 2 ) ∧ ( 2 ∨ 3 ∨ 4 ) <|endofprompt|>
```

- **Response:**

```
[CALL] Question: ( 4 ∨ ¬ 3 ∨ ¬ 2 ) ∧ ( ¬ 4 ∨ ¬ 2 ∨ 1 ) ∧ ( ¬ 2 ∨ ¬ 1 ∨ 3 ) ∧ ( 3 ∨
    ↪  4 ∨ 2 ) ∧ ( 3 ∨ ¬ 2 ∨ 4 ) ∧ ( ¬ 1 ∨ 4 ∨ 2 ) ∧ ( 1 ∨ ¬ 4 ∨ 2 ) ∧ ( 4 ∨ ¬ 2
    ↪ ∨ 3 ) ∧ ( ¬ 4 ∨ 1 ∨ ¬ 3 ) ∧ ( 3 ∨ 2 ∨ 4 ) ∧ ( ¬ 3 ∨ 1 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ 2 ∨
    ↪  ¬ 3 ) ∧ ( 2 ∨ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 1 ∨ ¬ 4 ) ∧ ( ¬ 1 ∨ ¬ 4
    ↪ ∨ ¬ 2 ) ∧ ( 2 ∨ 3 ∨ 4 ) Try 1 = True [CALL] Question: ( 4 ∨ ¬ 3 ∨ ¬ 2 ) ∧ (
    ↪  ¬ 2 ∨ 3 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 2 ∨ 4 ) ∧ ( 4 ∨ 2 ) ∧ ( 4 ∨ ¬ 2 ∨ 3 )
    ↪ ∧ ( 3 ∨ 2 ∨ 4 ) ∧ ( ¬ 4 ∨ 2 ∨ ¬ 3 ) ∧ ( 2 ∨ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 3
    ↪  ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 2 ) ∧ ( 2 ∨ 3 ∨ 4 ) Try 2 = True [CALL] Question: ( 4
    ↪ ∨ ¬ 3 ) ∧ ( 3 ) ∧ ( 3 ∨ 4 ) ∧ ( 4 ∨ 3 ) ∧ ( 3 ∨ ¬ 4 ) ∧ ( ¬ 4 ) Found 3 Let
    ↪  3 = True [CALL] Question: ( 4 ) ∧ ( ¬ 4 ) Found 4 Let 4 = True [SEP]
    ↪ Answer: False [RETURN] [SEP] Answer: False [RETURN] Try 2 = False [CALL]
    ↪ Question: ( 3 ∨ 4 ) ∧ ( 4 ) ∧ ( 3 ∨ 4 ) ∧ ( ¬ 4 ∨ ¬ 3 ) ∧ ( 4 ∨ ¬ 3 ) ∧ ( 3
    ↪  ∨ 4 ) ∧ ( 3 ∨ ¬ 4 ) ∧ ( 3 ∨ 4 ) Found 4 Let 4 = True [CALL] Question: ( ¬
    ↪ 3 ) ∧ ( 3 ) Found ¬ 3 Let 3 = False [SEP] Answer: False [RETURN] [SEP]
    ↪ Answer: False [RETURN] [SEP] Answer: False [RETURN] Try 1 = False [CALL]
    ↪ Question: ( 4 ∨ ¬ 3 ∨ ¬ 2 ) ∧ ( ¬ 4 ∨ ¬ 2 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 2 ∨ 4
    ↪  ) ∧ ( ¬ 4 ∨ 2 ) ∧ ( 4 ∨ ¬ 2 ∨ 3 ) ∧ ( ¬ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 2 ∨ 4 ) ∧ ( ¬ 3
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ 2 ∨ ¬ 3 ) ∧ ( 2 ∨ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 2 ∨ 3 ∨ 4
    ↪  ) Try 2 = True [CALL] Question: ( 4 ∨ ¬ 3 ) ∧ ( ¬ 4 ) ∧ ( 3 ∨ 4 ) ∧ ( 4 ∨
```

```
↪ 3 ) ∧ ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 4 ) Found ¬ 4 Let 4 = False [CALL] Question
↪ : ( ¬ 3 ) ∧ ( 3 ) ∧ ( 3 ) Found ¬ 3 Let 3 = False [SEP] Answer: False [
↪ RETURN] [SEP] Answer: False [RETURN] Try 2 = False [CALL] Question: ( 3 ∨ 4
↪ ) ∧ ( ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ) ∧ ( ¬ 3 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 3 ) ∧
↪ ( 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ) ∧ ( 3 ∨ 4 ) Found ¬ 4 Let 4 = False [CALL] Question
↪ : ( 3 ) ∧ ( 3 ) ∧ ( ¬ 3 ) ∧ ( 3 ) ∧ ( 3 ) Found 3 Let 3 = True [SEP] Answer
↪ : False [RETURN] [SEP] Answer: False [RETURN] [SEP] Answer: False [RETURN]
↪ [SEP] Answer: False [RETURN] <|endoftext|>
```

PENCIL (WITH REDUCTION RULE)

- **Prompt:**

```
<|startoftext|> ( 4 ∨ ¬ 3 ∨ ¬ 2 ) ∧ ( ¬ 4 ∨ ¬ 2 ∨ 1 ) ∧ ( ¬ 2 ∨ ¬ 1 ∨ 3 ) ∧ ( 3 ∨
↪ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 2 ∨ 4 ) ∧ ( ¬ 1 ∨ 4 ∨ 2 ) ∧ ( 1 ∨ ¬ 4 ∨ 2 ) ∧ ( 4 ∨ ¬ 2 ∨
↪ 3 ) ∧ ( ¬ 4 ∨ 1 ∨ ¬ 3 ) ∧ ( 3 ∨ 2 ∨ 4 ) ∧ ( ¬ 3 ∨ 1 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ 2 ∨
↪ ¬ 3 ) ∧ ( 2 ∨ 4 ∨ ¬ 3 ) ∧ ( 3 ∨ 4 ∨ 2 ) ∧ ( 3 ∨ ¬ 1 ∨ ¬ 4 ) ∧ ( ¬ 1 ∨ ¬ 4 ∨
↪ ¬ 2 ) ∧ ( 2 ∨ 3 ∨ 4 ) <|endofprompt|>
```

- **Response:**

```
Answer: False <|endoftext|>
```

INTERNAL REASONING PROCESS OF PENCIL

Model Generation (1)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
↪ 1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
↪ 1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
↪ 2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True [CALL] Question: ( ¬
↪ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 4 ∨ 3 ) Try 3 = True [CALL]
↪ Question: ( ¬ 4 ) ∧ ( 4 ) ∧ ( 4 ) Found ¬ 4 Let 4 = False [SEP] Answer:
↪ False [RETURN]
```

Reduction Rule (1)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
↪ 1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
↪ 1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
↪ 2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True [CALL] Question: ( ¬
↪ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 4 ∨ 3 ) Try 3 = True Answer:
↪ False
```

Model Generation (2)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
↪ 1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
↪ 1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
↪ 2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True [CALL] Question: ( ¬
↪ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 4 ∨ 3 ) Try 3 = True Answer:
↪ False Try 3 = False [CALL] Question: ( ¬ 4 ) Found ¬ 4 Let 4 = False [SEP]
↪ Answer: True [RETURN]
```

Reduction Rule (2)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
    ↪  1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
    ↪  1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
    ↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
    ↪  2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
    ↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True [CALL] Question: ( ¬
    ↪  4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 4 ∨ 3 ) Try 3 = True Answer:
    ↪ False Try 3 = False Answer: True
```

Model Generation (3)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
    ↪  1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
    ↪  1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
    ↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
    ↪  2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
    ↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True [CALL] Question: ( ¬
    ↪  4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 4 ∨ 3 ) Try 3 = True Answer:
    ↪ False Try 3 = False Answer: True [SEP] Answer: True [RETURN]
```

Reduction Rule (3)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
    ↪  1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
    ↪  1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
    ↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
    ↪  2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
    ↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True Answer: True
```

Model Generation (4)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
    ↪  1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
    ↪  1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
    ↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True [CALL] Question: ( ¬ 4 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨
    ↪  2 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4
    ↪ ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 2 = True Answer: True [SEP]
    ↪ Answer: True [RETURN]
```

Reduction Rule (4)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
    ↪  1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
    ↪  1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
    ↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True Answer: True
```

Model Generation (5)

```
[CALL] Question: ( ¬ 3 ∨ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨ 2 ) ∧ ( ¬ 4 ∨ ¬ 3 ∨ ¬ 1 ) ∧ ( ¬ 3 ∨ ¬
    ↪  1 ∨ 2 ) ∧ ( 4 ∨ 1 ∨ 3 ) ∧ ( 4 ∨ 1 ∨ ¬ 3 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 1 ∨ 2
    ↪ ∨ ¬ 4 ) ∧ ( ¬ 3 ∨ ¬ 1 ∨ 4 ) ∧ ( ¬ 3 ∨ 2 ∨ ¬ 4 ) ∧ ( ¬ 4 ∨ ¬ 1 ∨ 3 ) ∧ ( 2 ∨
    ↪  1 ∨ ¬ 3 ) ∧ ( 1 ∨ 4 ∨ 3 ) ∧ ( 2 ∨ ¬ 3 ∨ 4 ) ∧ ( 2 ∨ ¬ 4 ∨ 1 ) ∧ ( 1 ∨ 3 ∨
    ↪ 2 ) ∧ ( 4 ∨ 2 ∨ ¬ 3 ) Try 1 = True Answer: True [SEP] Answer: True [RETURN]
```

Reduction Rule (5)

```
Answer: True
```

Model Generation (Final Response)

```
Answer: True <|endoftext|>
```

G.2 Quantified Boolean Formula (QBF)

Chain-of-Thought (w/o Reduction Rule)

• Prompt:

```
<|startoftext|> ∀ 3 ∀ 2 ∃ 4 ∃ 1 : #1 ( ¬ 3 ∨ 2 ∨ 4 ) #2 ( ¬ 4 ∨ 3 ) #3 ( ¬ 4 ∨ 4 ∨
    ↪  3 ) #4 ( ¬ 3 ∨ 1 ) #5 ( ¬ 3 ∨ 3 ) #6 ( ¬ 4 ∨ ¬ 2 ) #7 ( ¬ 2 ∨ 1 ) #8 ( ¬ 1
    ↪  ∨ 3 ∨ 2 ) <|endofprompt|>
```

• Response:

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False [CALL] Question: prefix_from ∃ 4 Try 4 = False [CALL]
    ↪ Question: prefix_from ∃ 1 Try 1 = False [CALL] Question: evaluate #1 =
    ↪ False #2 = False #3 = False #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check
    ↪ #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True
    ↪  Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 )
    ↪  True Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula = True [SEP] Answer: True [
    ↪ RETURN] [SEP] Answer: True [RETURN] [SEP] Answer: True [RETURN] Try 2 =
    ↪ True [CALL] Question: prefix_from ∃ 4 Try 4 = False [CALL] Question:
    ↪ prefix_from ∃ 1 Try 1 = False [CALL] Question: evaluate #1 = False #2 =
    ↪ True #3 = False #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3
    ↪  ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True Check #4 (
    ↪ ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 ) False [SEP]
    ↪  Answer: False [RETURN] Try 1 = True [CALL] Question: evaluate #1 = True #2
    ↪  = True #3 = False #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4
    ↪ ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True Check #4
    ↪  ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 ) True
    ↪ Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula = True [SEP] Answer: True [RETURN] [
    ↪ SEP] Answer: True [RETURN] [SEP] Answer: True [RETURN] [SEP] Answer: True [
    ↪ RETURN] Try 3 = True [CALL] Question: prefix_from ∀ 2 Try 2 = False [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False [CALL] Question: evaluate #1 = False #2 = False #3 = True #4
    ↪ = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) False [SEP] Answer: False [RETURN] Try 1 =
    ↪  True [CALL] Question: evaluate #1 = True #2 = False #3 = True #4 = False
    ↪ Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) False [SEP] Answer: False [RETURN] [SEP] Answer:
    ↪ False [RETURN] Try 4 = True [CALL] Question: prefix_from ∃ 1 Try 1 = False
    ↪ [CALL] Question: evaluate #1 = False #2 = False #3 = True #4 = True Check
    ↪ #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 )
    ↪ True Check #3 ( ¬ 3 ∨ 1 ) False [SEP] Answer: False [RETURN] Try 1 = True [
    ↪ CALL] Question: evaluate #1 = True #2 = False #3 = True #4 = True Check #0
    ↪ ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 )
    ↪ True Check #3 ( ¬ 3 ∨ 1 ) True Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬
    ↪  2 ) True Check #6 ( ¬ 2 ∨ 1 ) True Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula =
    ↪  True [SEP] Answer: True [RETURN] [SEP] Answer: True [RETURN] [SEP] Answer:
    ↪  True [RETURN] Try 2 = True [CALL] Question: prefix_from ∃ 4 Try 4 = False
    ↪ [CALL] Question: prefix_from ∃ 1 Try 1 = False [CALL] Question: evaluate #1
    ↪  = False #2 = True #3 = True #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check
    ↪  #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 )
    ↪ False [SEP] Answer: False [RETURN] Try 1 = True [CALL] Question: evaluate
    ↪ #1 = True #2 = True #3 = True #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True
    ↪ Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1
    ↪ ) True Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2
    ↪  ∨ 1 ) True Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula = True [SEP] Answer: True
    ↪  [RETURN] [SEP] Answer: True [RETURN] [SEP] Answer: True [RETURN] [SEP]
    ↪ Answer: True [RETURN] [SEP] Answer: True [RETURN] <|endoftext|>
```

PENCIL (with Reduction Rule)

• Prompt:

```
<|startoftext|> ∀ 3 ∀ 2 ∃ 4 ∃ 1 : #1 ( ¬ 3 ∨ 2 ∨ 4 ) #2 ( ¬ 4 ∨ 3 ) #3 ( ¬ 4 ∨ 4 ∨
    ↪  3 ) #4 ( ¬ 3 ∨ 1 ) #5 ( ¬ 3 ∨ 3 ) #6 ( ¬ 4 ∨ ¬ 2 ) #7 ( ¬ 2 ∨ 1 ) #8 ( ¬ 1
    ↪  ∨ 3 ∨ 2 ) <|endofprompt|>
```

• Response:

```
Answer: True <|endoftext|>
```

INTERNAL REASONING PROCESS OF PENCIL

Model Generation (1)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False [CALL] Question: prefix_from ∃ 4 Try 4 = False [CALL]
    ↪ Question: prefix_from ∃ 1 Try 1 = False [CALL] Question: evaluate #1 =
    ↪ False #2 = False #3 = False #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check
    ↪ #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True
    ↪  Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 )
    ↪  True Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula = True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (1)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False [CALL] Question: prefix_from ∃ 4 Try 4 = False [CALL]
    ↪ Question: prefix_from ∃ 1 Try 1 = False Answer: True
```

Model Generation (2)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False [CALL] Question: prefix_from ∃ 4 Try 4 = False [CALL]
    ↪ Question: prefix_from ∃ 1 Try 1 = False Answer: True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (2)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False [CALL] Question: prefix_from ∃ 4 Try 4 = False Answer: True
```

Model Generation (3)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False [CALL] Question: prefix_from ∃ 4 Try 4 = False Answer: True [
    ↪ SEP] Answer: True [RETURN]
```

Reduction Rule (3)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True
```

Model Generation (4)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False [CALL]
    ↪ Question: evaluate #1 = False #2 = True #3 = False #4 = False Check #0 ( ¬
    ↪ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True
    ↪ Check #3 ( ¬ 3 ∨ 1 ) True Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 )
    ↪ True Check #6 ( ¬ 2 ∨ 1 ) False [SEP] Answer: False [RETURN]
```

Reduction Rule (4)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
```

Model Generation (5)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
    ↪ Try 1 = True [CALL] Question: evaluate #1 = True #2 = True #3 = False #4 =
    ↪ False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬
    ↪ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True Check #4 ( ¬ 3 ∨ 3 ) True Check
    ↪ #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 ) True Check #7 ( ¬ 1 ∨ 3 ∨ 2 )
    ↪ True Formula = True [SEP] Answer: True [RETURN]
```

Reduction Rule (5)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
    ↪ Try 1 = True Answer: True
```

Model Generation (6)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
    ↪ Try 1 = True Answer: True [SEP] Answer: True [RETURN]
```

Reduction Rule (6)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: True
```

Model Generation (7)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: True [SEP] Answer: True [RETURN]
```

Reduction Rule (7)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True Answer: True
```

Model Generation (8)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False [CALL] Question: prefix_from ∀ 2
    ↪ Try 2 = False Answer: True Try 2 = True Answer: True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (8)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True
```

Model Generation (9)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False [CALL]
    ↪ Question: evaluate #1 = False #2 = False #3 = True #4 = False Check #0 ( ¬
    ↪ 3 ∨ 2 ∨ 4 ) False [SEP] Answer: False [RETURN]
```

Reduction Rule (9)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
```

Model Generation (10)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
    ↪ Try 1 = True [CALL] Question: evaluate #1 = True #2 = False #3 = True #4 =
    ↪ False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) False [SEP] Answer: False [RETURN]
```

Reduction Rule (10)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
    ↪ Try 1 = True Answer: False
```

Model Generation (11)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False [CALL] Question: prefix_from ∃ 1 Try 1 = False Answer: False
    ↪ Try 1 = True Answer: False [SEP] Answer: False [RETURN]
```

Reduction Rule (11)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False
```

Model Generation (12)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False [CALL] Question: evaluate #1 = False #2 = False #3 = True #4
    ↪ = True Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬
    ↪  4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) False [SEP] Answer: False [RETURN]
```

Reduction Rule (12)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False
```

Model Generation (13)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False Try 1 = True [CALL] Question: evaluate #1 =
    ↪ True #2 = False #3 = True #4 = True Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1
    ↪ ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True
    ↪ Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 )
    ↪ True Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula = True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (13)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False Try 1 = True Answer: True
```

Model Generation (14)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False Try 1 = True Answer: True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (14)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True Answer: True
```

Model Generation (15)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False [CALL] Question: prefix_from ∃ 4
    ↪ Try 4 = False Answer: False Try 4 = True Answer: True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (15)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True
```

Model Generation (16)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False [CALL] Question: evaluate #1 = False #2 = True #3 = True #4 =
    ↪  False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1 ( ¬ 4 ∨ 3 ) True Check #2 ( ¬
    ↪  4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) False [SEP] Answer: False [RETURN]
```

Reduction Rule (16)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False
```

Model Generation (17)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False Try 1 = True [CALL] Question: evaluate #1 =
    ↪ True #2 = True #3 = True #4 = False Check #0 ( ¬ 3 ∨ 2 ∨ 4 ) True Check #1
    ↪ ( ¬ 4 ∨ 3 ) True Check #2 ( ¬ 4 ∨ 4 ∨ 3 ) True Check #3 ( ¬ 3 ∨ 1 ) True
    ↪ Check #4 ( ¬ 3 ∨ 3 ) True Check #5 ( ¬ 4 ∨ ¬ 2 ) True Check #6 ( ¬ 2 ∨ 1 )
    ↪ True Check #7 ( ¬ 1 ∨ 3 ∨ 2 ) True Formula = True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (17)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False Try 1 = True Answer: True
```

Model Generation (18)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False [CALL] Question: prefix_from ∃ 1
    ↪ Try 1 = False Answer: False Try 1 = True Answer: True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (18)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False Answer: True
```

Model Generation (19)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True [CALL]
    ↪ Question: prefix_from ∃ 4 Try 4 = False Answer: True [SEP] Answer: True [
    ↪ RETURN]
```

Reduction Rule (19)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True Answer:
    ↪ True
```

Model Generation (20)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True [CALL]
    ↪ Question: prefix_from ∀ 2 Try 2 = False Answer: True Try 2 = True Answer:
    ↪ True [SEP] Answer: True [RETURN]
```

Reduction Rule (20)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True Answer:
    ↪ True
```

Model Generation (21)

```
[CALL] Question: prefix_from ∀ 3 Try 3 = False Answer: True Try 3 = True Answer:
    ↪ True [SEP] Answer: True [RETURN]
```

Reduction Rule (21)

```
Answer: True
```

Model Generation (Final Response)

```
Answer: True <|endoftext|>
```

G.3 EINSTEIN'S PUZZLE

CHAIN-OF-THOUGHT (W/O REDUCTION RULE)

• Prompt:

```
<|startoftext|> Constraint#1 : the Green house is immediately to the right of the
    ↪ one who keeps Birds
 Constraint#2 : the Brit is immediately to the right of the German
 Constraint#3 : the one who keeps Dogs is the same house as the Red house
 Constraint#4 : the one who keeps Birds is immediately to the right of the Swede
     ↪ <|endofprompt|>
```

• Response:

```
[CALL] ====== Possible Assignments ======
 House#1
 Color category have 3 possibilities Blue Green Red
 Nationality category have 3 possibilities Brit German Swede
 Pet category have 3 possibilities Birds Dogs Fish
 House#2
 Color category have 3 possibilities Blue Green Red
 Nationality category have 3 possibilities Brit German Swede
 Pet category have 3 possibilities Birds Dogs Fish
 House#3
 Color category have 3 possibilities Blue Green Red
 Nationality category have 3 possibilities Brit German Swede
 Pet category have 3 possibilities Birds Dogs Fish
 Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
 => Puzzle not solved yet
 ====== Propagation ======
 Applying Constraint#1 [CALL]
 PHASE 1: Single-value logic for Green and Birds under RIGHT constraint
 PHASE 2: Handling relation Green RIGHT Birds
 Green is immediately RIGHT of Birds
 Removing Green from House#1 because Green can't be in the leftmost house if it's
     ↪ to the RIGHT of Birds
 Removing Birds from House#3 can't be in the rightmost house if it's to the LEFT
     ↪ of Green
 [SEP] House#1 Color category changed from 3 possibilities Blue Green Red to 2
     ↪ possibilities Blue Red
 House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 2
     ↪ possibilities Dogs Fish [RETURN]
 Applying Constraint#2 [CALL]
 PHASE 1: Single-value logic for Brit and German under RIGHT constraint
```

```
PHASE 2: Handling relation Brit RIGHT German
Brit is immediately RIGHT of German
Removing Brit from House#1 because Brit can't be in the leftmost house if it's to
    ↪  the RIGHT of German
Removing German from House#3 can't be in the rightmost house if it's to the LEFT
    ↪ of Brit
[SEP] House#1 Nationality category changed from 3 possibilities Brit German Swede
    ↪  to 2 possibilities German Swede
House#3 Nationality category changed from 3 possibilities Brit German Swede to 2
    ↪ possibilities Brit Swede [RETURN]
Applying Constraint#3 [CALL]
PHASE 1: Single-value logic for Dogs and Red under SAME constraint
PHASE 2: Handling relation Dogs SAME Red
Dogs must be in the SAME house as Red
[SEP] No changes from this constraint [RETURN]
Applying Constraint#4 [CALL]
PHASE 1: Single-value logic for Birds and Swede under RIGHT constraint
PHASE 2: Handling relation Birds RIGHT Swede
Birds is immediately RIGHT of Swede
Removing Birds from House#1 because Birds can't be in the leftmost house if it's
    ↪ to the RIGHT of Swede
Removing Swede from House#3 can't be in the rightmost house if it's to the LEFT
    ↪ of Birds
[SEP] House#3 Nationality category changed from 2 possibilities Brit Swede to 1
    ↪ possibilities Brit
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Dogs Fish [RETURN]
[SEP] [CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Red
Nationality category have 2 possibilities German Swede
Pet category have 2 possibilities Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category is Brit
Pet category have 2 possibilities Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4 [
    ↪ RETURN]
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Red
Trying possibility Blue in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Blue
Nationality category have 2 possibilities German Swede
Pet category have 2 possibilities Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category is Brit
Pet category have 2 possibilities Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 [CALL]
PHASE 1: Single-value logic for Green and Birds under RIGHT constraint
Removing Blue from House#2 Color category because Blue is pinned in another house
Removing Blue from House#3 Color category because Blue is pinned in another house
Forcing Birds in House#2 Pet category because it can only appear here
```

```
PHASE 2: Handling relation Green RIGHT Birds
Green is immediately RIGHT of Birds
Since Birds is pinned to House#2 , removing Green from House#2 because Green must
    ↪  be right of House#2
Placing Green in House#3 because Birds is pinned to House#2
[SEP] House#2 Color category changed from 3 possibilities Blue Green Red to 1
    ↪ possibilities Red
House#3 Color category changed from 3 possibilities Blue Green Red to 1
    ↪ possibilities Green
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds [RETURN]
Remove Constraint#1 because it is satisfied
Applying Constraint#2 [CALL]
PHASE 1: Single-value logic for Brit and German under RIGHT constraint
Removing Brit from House#2 Nationality category because Brit is pinned in another
    ↪  house
PHASE 2: Handling relation Brit RIGHT German
Brit is immediately RIGHT of German
German must be exactly one house to the LEFT , removing from House#1
Placing German in House#2 because Brit is pinned to House#3
[SEP] House#1 Nationality category changed from 2 possibilities German Swede to 1
    ↪  possibilities Swede
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities German [RETURN]
Remove Constraint#2 because it is satisfied
Applying Constraint#3 [CALL]
PHASE 1: Single-value logic for Dogs and Red under SAME constraint
PHASE 2: Handling relation Dogs SAME Red
Dogs must be in the SAME house as Red
Since Red is pinned to House#2 , removing Dogs from House#1
Since Red is pinned to House#2 , removing Dogs from House#3
House#2 can't hold Dogs since it can't hold Red
[SEP] House#2 Color category changed from 1 possibilities Red to 0 possibilities
    ↪ empty
House#1 Pet category changed from 2 possibilities Dogs Fish to 1 possibilities
    ↪ Fish
House#3 Pet category changed from 2 possibilities Dogs Fish to 1 possibilities
    ↪ Fish [RETURN]
[SEP] No Solution [RETURN]
Trying possibility Red in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Red
Nationality category have 2 possibilities German Swede
Pet category have 2 possibilities Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category is Brit
Pet category have 2 possibilities Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 [CALL]
PHASE 1: Single-value logic for Green and Birds under RIGHT constraint
Removing Red from House#2 Color category because Red is pinned in another house
Removing Red from House#3 Color category because Red is pinned in another house
Forcing Birds in House#2 Pet category because it can only appear here
PHASE 2: Handling relation Green RIGHT Birds
Green is immediately RIGHT of Birds
Since Birds is pinned to House#2 , removing Green from House#2 because Green must
    ↪  be right of House#2
Placing Green in House#3 because Birds is pinned to House#2
```

```
[SEP] House#2 Color category changed from 3 possibilities Blue Green Red to 1
    ↪ possibilities Blue
House#3 Color category changed from 3 possibilities Blue Green Red to 1
    ↪ possibilities Green
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds [RETURN]
Remove Constraint#1 because it is satisfied
Applying Constraint#2 [CALL]
PHASE 1: Single-value logic for Brit and German under RIGHT constraint
Removing Brit from House#2 Nationality category because Brit is pinned in another
    ↪  house
PHASE 2: Handling relation Brit RIGHT German
Brit is immediately RIGHT of German
German must be exactly one house to the LEFT , removing from House#1
Placing German in House#2 because Brit is pinned to House#3
[SEP] House#1 Nationality category changed from 2 possibilities German Swede to 1
    ↪  possibilities Swede
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities German [RETURN]
Remove Constraint#2 because it is satisfied
Applying Constraint#3 [CALL]
PHASE 1: Single-value logic for Dogs and Red under SAME constraint
PHASE 2: Handling relation Dogs SAME Red
Dogs must be in the SAME house as Red
Placing Dogs in House#1 since Red is in this house
Since Red is pinned to House#1 , removing Dogs from House#3
[SEP] House#1 Pet category changed from 2 possibilities Dogs Fish to 1
    ↪ possibilities Dogs
House#3 Pet category changed from 2 possibilities Dogs Fish to 1 possibilities
    ↪ Fish [RETURN]
Remove Constraint#3 because it is satisfied
Applying Constraint#4 [CALL]
PHASE 1: Single-value logic for Birds and Swede under RIGHT constraint
PHASE 2: Handling relation Birds RIGHT Swede
Birds is immediately RIGHT of Swede
[SEP] No changes from this constraint [RETURN]
Remove Constraint#4 because it is satisfied
[SEP] [CALL] ====== Possible Assignments After Propagation ======
House#1
Color category is Red
Nationality category is Swede
Pet category is Dogs
House#2
Color category is Blue
Nationality category is German
Pet category is Birds
House#3
Color category is Green
Nationality category is Brit
Pet category is Fish
Unsatisfied constraints are [RETURN]
=> Puzzle is solved
[SEP] Solution House#1
Color category is Red
Nationality category is Swede
Pet category is Dogs
House#2
Color category is Blue
Nationality category is German
Pet category is Birds
House#3
Color category is Green
Nationality category is Brit
Pet category is Fish [RETURN]
[SEP] Solution House#1
Color category is Red
Nationality category is Swede
```

```
Pet category is Dogs
House#2
Color category is Blue
Nationality category is German
Pet category is Birds
House#3
Color category is Green
Nationality category is Brit
Pet category is Fish [RETURN]
=> House#3 owns the Fish
=> the Brit owns the Fish <|endoftext|>
```

PENCIL (WITH REDUCTION RULE)

• Prompt:

```
<|startoftext|> Constraint#1 : the Green house is immediately to the right of the
    ↪ one who keeps Birds
Constraint#2 : the Brit is immediately to the right of the German
Constraint#3 : the one who keeps Dogs is the same house as the Red house
Constraint#4 : the one who keeps Birds is immediately to the right of the Swede
    ↪ <|endofprompt|>
```

• Response:

```
Solution House#1
Color category is Red
Nationality category is Swede
Pet category is Dogs
House#2
Color category is Blue
Nationality category is German
Pet category is Birds
House#3
Color category is Green
Nationality category is Brit
Pet category is Fish
=> House#3 owns the Fish
=> the Brit owns the Fish <|endoftext|>
```

INTERNAL REASONING PROCESS OF PENCIL

Model Generation (1)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 [CALL]
PHASE 1: Single-value logic for Fish and Red under RIGHT constraint
PHASE 2: Handling relation Fish RIGHT Red
Fish is immediately RIGHT of Red
Removing Fish from House#1 because Fish can't be in the leftmost house if it's to
    ↪ the RIGHT of Red
```

```
Removing Red from House#3 can't be in the rightmost house if it's to the LEFT of
    ↪ Fish
[SEP] House#3 Color category changed from 3 possibilities Blue Green Red to 2
    ↪ possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs [RETURN]
```

Reduction Rule (1)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
```

Model Generation (2)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 [CALL]
PHASE 1: Single-value logic for Green and Red under LEFT constraint
PHASE 2: Handling relation Green LEFT Red
Green is immediately LEFT of Red
Removing Green from House#3 because Green can't be in the rightmost house if it's
    ↪  to the LEFT of Red
Removing Red from House#1 because Red can't be in the leftmost house if it's to
    ↪ the RIGHT of Green
[SEP] House#1 Color category changed from 3 possibilities Blue Green Red to 2
    ↪ possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue [RETURN]
```

Reduction Rule (2)

```
[CALL] ====== Possible Assignments ======
```

```
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 House#1 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue
```

Model Generation (3)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 House#1 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue
Applying Constraint#3 [CALL]
PHASE 1: Single-value logic for Fish and Swede under RIGHT constraint
PHASE 2: Handling relation Fish RIGHT Swede
Fish is immediately RIGHT of Swede
Removing Swede from House#3 can't be in the rightmost house if it's to the LEFT
    ↪ of Fish
[SEP] House#3 Nationality category changed from 3 possibilities Brit German Swede
    ↪  to 2 possibilities Brit German [RETURN]
```

Reduction Rule (3)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
```

```
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 House#1 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue
Applying Constraint#3 House#3 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 2 possibilities Brit German
```

Model Generation (4)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 House#1 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue
Applying Constraint#3 House#3 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 2 possibilities Brit German
Applying Constraint#4 [CALL]
PHASE 1: Single-value logic for Brit and Birds under LEFT constraint
PHASE 2: Handling relation Brit LEFT Birds
Brit is immediately LEFT of Birds
Removing Brit from House#3 because Brit can't be in the rightmost house if it's
    ↪ to the LEFT of Birds
Removing Birds from House#1 because Birds can't be in the leftmost house if it's
    ↪ to the RIGHT of Brit
[SEP] House#3 Nationality category changed from 2 possibilities Brit German to 1
    ↪ possibilities German
House#1 Pet category changed from 2 possibilities Birds Dogs to 1 possibilities
    ↪ Dogs [RETURN]
```

Reduction Rule (4)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
```

```
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 House#1 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue
Applying Constraint#3 House#3 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 2 possibilities Brit German
Applying Constraint#4 House#3 Nationality category changed from 2 possibilities
    ↪ Brit German to 1 possibilities German
House#1 Pet category changed from 2 possibilities Birds Dogs to 1 possibilities
    ↪ Dogs
```

Model Generation (5)

```
[CALL] ====== Possible Assignments ======
House#1
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#3 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#1 Pet category changed from 3 possibilities Birds Dogs Fish to 2
    ↪ possibilities Birds Dogs
Applying Constraint#2 House#1 Color category changed from 3 possibilities Blue
    ↪ Green Red to 2 possibilities Blue Green
House#3 Color category changed from 2 possibilities Blue Green to 1 possibilities
    ↪  Blue
Applying Constraint#3 House#3 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 2 possibilities Brit German
Applying Constraint#4 House#3 Nationality category changed from 2 possibilities
    ↪ Brit German to 1 possibilities German
House#1 Pet category changed from 2 possibilities Birds Dogs to 1 possibilities
    ↪ Dogs
[SEP] [CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
```

```
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4 [
    ↪ RETURN]
```

Reduction Rule (5)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
```

Model Generation (6)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 [CALL]
PHASE 1: Single-value logic for Fish and Red under RIGHT constraint
Removing Dogs from House#2 Pet category because Dogs is pinned in another house
Removing Dogs from House#3 Pet category because Dogs is pinned in another house
Removing Green from House#2 Color category because Green is pinned in another
    ↪ house
Removing Blue from House#2 Color category because Blue is pinned in another house
PHASE 2: Handling relation Fish RIGHT Red
Fish is immediately RIGHT of Red
```

```
Since Red is pinned to House#2 , removing Fish from House#2 because Fish must be
    ↪ right of House#2
Placing Fish in House#3 because Red is pinned to House#2
[SEP] House#2 Color category changed from 3 possibilities Blue Green Red to 1
    ↪ possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish [RETURN]
```

Reduction Rule (6)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
```

Model Generation (7)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
```

```
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
Applying Constraint#2 [CALL]
PHASE 1: Single-value logic for Green and Red under LEFT constraint
PHASE 2: Handling relation Green LEFT Red
Green is immediately LEFT of Red
[SEP] No changes from this constraint [RETURN]
```

Reduction Rule (7)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
```

```
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
Applying Constraint#2 No changes from this constraint
```

Model Generation (8)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
Applying Constraint#2 No changes from this constraint
Remove Constraint#2 because it is satisfied
Applying Constraint#3 [CALL]
PHASE 1: Single-value logic for Fish and Swede under RIGHT constraint
Removing German from House#1 Nationality category because German is pinned in
    ↪ another house
Removing German from House#2 Nationality category because German is pinned in
    ↪ another house
PHASE 2: Handling relation Fish RIGHT Swede
Fish is immediately RIGHT of Swede
Swede must be exactly one house to the LEFT , removing from House#1
Placing Swede in House#2 because Fish is pinned to House#3
[SEP] House#1 Nationality category changed from 3 possibilities Brit German Swede
    ↪  to 1 possibilities Brit
```

```
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities Swede [RETURN]
```

Reduction Rule (8)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
Applying Constraint#2 No changes from this constraint
Remove Constraint#2 because it is satisfied
Applying Constraint#3 House#1 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 1 possibilities Brit
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities Swede
```

Model Generation (9)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
```

```
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
Applying Constraint#2 No changes from this constraint
Remove Constraint#2 because it is satisfied
Applying Constraint#3 House#1 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 1 possibilities Brit
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities Swede
Remove Constraint#3 because it is satisfied
Applying Constraint#4 [CALL]
PHASE 1: Single-value logic for Brit and Birds under LEFT constraint
PHASE 2: Handling relation Brit LEFT Birds
Brit is immediately LEFT of Birds
[SEP] No changes from this constraint [RETURN]
```

Reduction Rule (9)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
```

```
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
Applying Constraint#2 No changes from this constraint
Remove Constraint#2 because it is satisfied
Applying Constraint#3 House#1 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 1 possibilities Brit
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities Swede
Remove Constraint#3 because it is satisfied
Applying Constraint#4 No changes from this constraint
```

Model Generation (10)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments ======
House#1
Color category is Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Propagation ======
Applying Constraint#1 House#2 Color category changed from 3 possibilities Blue
    ↪ Green Red to 1 possibilities Red
House#2 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Birds
House#3 Pet category changed from 3 possibilities Birds Dogs Fish to 1
    ↪ possibilities Fish
Remove Constraint#1 because it is satisfied
```

```
Applying Constraint#2 No changes from this constraint
Remove Constraint#2 because it is satisfied
Applying Constraint#3 House#1 Nationality category changed from 3 possibilities
    ↪ Brit German Swede to 1 possibilities Brit
House#2 Nationality category changed from 3 possibilities Brit German Swede to 1
    ↪ possibilities Swede
Remove Constraint#3 because it is satisfied
Applying Constraint#4 No changes from this constraint
Remove Constraint#4 because it is satisfied
[SEP] [CALL] ====== Possible Assignments After Propagation ======
House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish
Unsatisfied constraints are [RETURN]
```

Reduction Rule (10)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish
Unsatisfied constraints are
```

Model Generation (11)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
```

```
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish
Unsatisfied constraints are
=> Puzzle is solved
[SEP] Solution House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish [RETURN]
```

Reduction Rule (11)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
Solution House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
```

```
Color category is Blue
Nationality category is German
Pet category is Fish
```

Model Generation (12)

```
[CALL] ====== Possible Assignments After Propagation ======
House#1
Color category have 2 possibilities Blue Green
Nationality category have 3 possibilities Brit German Swede
Pet category is Dogs
House#2
Color category have 3 possibilities Blue Green Red
Nationality category have 3 possibilities Brit German Swede
Pet category have 3 possibilities Birds Dogs Fish
House#3
Color category is Blue
Nationality category is German
Pet category have 3 possibilities Birds Dogs Fish
Unsatisfied constraints are Constraint#1 Constraint#2 Constraint#3 Constraint#4
=> Puzzle not solved yet
====== Branch ======
Branching on House#1 Color category with 2 possibilities Blue Green
Trying possibility Green in House#1 Color category
Solution House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish
[SEP] Solution House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish [RETURN]
```

Reduction Rule (12)

```
Solution House#1
Color category is Green
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish
```

Model Generation (Final Response)

```
Solution House#1
Color category is Green
```

```
Nationality category is Brit
Pet category is Dogs
House#2
Color category is Red
Nationality category is Swede
Pet category is Birds
House#3
Color category is Blue
Nationality category is German
Pet category is Fish
=> House#3 owns the Fish
=> the German owns the Fish <|endoftext|>
```