Synthetically generated proofs in Propositional Logic and Language Model reasoning performance

Anthony Marchiafava[1,*], Atriya Sen[2] and Justin Moua[3]

Abstract

Large language models (LLMs) are a powerful tool in the field of machine learning specifically and artificial intelligence broadly. These LLMs are prone to making a variety of hallucinations and mistakes in reasoning in specific. This paper shows that improving the ability to generate propositional logic proofs in specific does not lead to greater performance on reasoning tasks in general. This is done by evaluating models fine-tuned on a synthetically generated proof by contradiction dataset on a variety of reasoning based benchmarks.

Keywords

Large Language Models, Synthetic Training Data, Reasoning, Machine Learning

1. Introduction

The capacity to use and apply logic has been an important task in artificial intelligence broadly. Recent work in machine learning broadly and using large language models in specific is concerned with improving reasoning performance. Large Language Models (LLMs) are a powerful tool in artificial intelligence but these generate text that is prone to a variety of mistakes and hallucinations in general but exhibit mistakes when it comes to reasoning in specific [1] [2]. Increasing the trustworthiness of these models is a meaningful goal. To evaluate this, a number of problems can be imagined as reasoning problems in a variety of types of logic. One of the simplest of these types of logic would be propositional logic where knowledge is encoded as sets of statements combined together with logical connectives [3]. One can evaluate more sophisticated logics such as first-order logic [4] or train a LLM for mathematical tasks including higher order logics [5]. One issue with these LLM approaches is that they can create false or hallucinated results.

Automated theorem provers are powerful tools to algorithmically produce step by step proofs that are able to be validated. These automated reasoners generate proofs are used as a sort of correct computational reasoning to show that some set of axioms deductively entail some set of conclusions. These automated theorem provers can be used to algorithmically create large amounts of synthetic data that a LLM can learn from and can be tailored to focus in on the types of logic that may be of interest. In this paper the sort of logic used is propositional, and an automated theorem prover can be used to produce valid proofs that a LLM can learn from.

2. Related Works

The large language models have been used for generating proofs before. Either in the production of proof steps [6] or to generate whole proofs generation in higher order logic [7]. Proof assistants such as Isabelle [8] for higher order logic were used to improve LLM performance on proof tasks like in [7]. Further work has built on using automated theorem provers to assist in proof generation through the use of tools which can do parts of the proof outside of the LLM [9]. Work has also been done in

HHAI-WS 2025: Workshops at the Fourth International Conference on Hybrid Human-Artificial Intelligence (HHAI), June 9–13, 2025, Pisa, Italy

*Corresponding author.

anmarch@okstate.edu (A. Marchiafava); asen@atriyasen.com/ (A. Sen); Hmuas11037@gmail.com (J. Moua)

https://www.atriyasen.com/ (A. Sen)

identifying reasoning errors and consistent mistakes in LLMs such as issues with in context learning [2].

3. Methodology

Training a large language model from start to finish is a computationally expensive and lengthy process [10]. Instead it is efficient to take an already existing foundational model and adjust this model to perform well on a specialized task through fine-tuning. The approach taken here is to test if training a specialized model on a propositional logic proof generation extends to better performance on a wider variety of reasoning tasks. To achieve this a set of propositional logic statements were generated, then input into an automated theorem prover which generated proofs, these proofs were then converted into natural language, and these natural language statements were used to fine-tune a pretrained large language model. The now fine-tuned large language model was evaluated on a number of benchmark reasoning tasks and these benchmarks were then compared to the performance of the pre-fine-tuned equivalent model.

3.1. Propositional Dataset Generation

Synthetically generating a large number of propositional logic statements was done algorithmically. The dataset was generated using a normal grammar in conjunctive normal form (CNF). Propositional logic statements can be expressed as combinations of conjunctions, where the conjunctions are combinations of conjunctions 'and'ed with disjunctions, and disjunctions are the combination of disjunctions 'or'ed with literals [11]. Here the dataset was generated with four literals and their negations $a, b, c, d, \neg a, \neg b, \neg c, \neg d$. These literals were then combined together to form disjunctions such as $a \vee b$, and these disjunctions were combined together to form conjunctions such as $(a \vee b) \wedge c$. This set of CNF statements were then treated first as a list of axioms and then as a separate list of conjectures. Each axiom in the list of axioms was then paired with each conjecture (where the axiom and the conjecture were not the same statement) in the Thousand Problems for Theorem Provers (TPTP) syntax [12]. Then the list of combined axiom and conjecture was shuffled. This follows a similar pattern to previous work where a synthetically generated dataset was used to train a classifier to differentiate between logic statements whose proofs were found or if there was no proof that was detected [13].

An individual combination of axiom and conjecture were then input into the automated theorem prover E which produced a proof showing if the axioms entailed the conjecture [14]. This process of feeding input into the automated theorem prover continued until a number, here 10,000, suitable proofs were produced. Each of these saved proofs showed that the conjecture was entailed by the axiom and would be used for fine-tuning while the results which showed there was no entailment were not used.

The E theorem prover utilized proof by contradiction solve to produce the proofs which were used to create the synthetic dataset here. This process begins by asserting the negation of the conclusion and attempting to derive a contradiction. If the negation of the conclusion entails a contradiction then the original conclusion is entailed. The E theorem prover created these proofs algorithmically and reports them instruction by instruction. These results were then converted into a natural language equivalent.

3.2. Natural Language Equivalents

Obtaining the natural language equivalence of the dataset was also done algorithmically. It does so by analyzing each problem's structure of various components like axioms, conjectures, refutation steps, inferences, and its label. Each component has their own sub-components acting as describable attributes. They are essential for the automated theorem prover E to algorithmically solve given problems. However, some were not necessary in obtaining the natural language equivalence of the results. Instead, core components were obtained that were used to achieve the result. As an example, a problem containing the axiom $(b) \wedge (\neg b \vee \neg c)$ with conjecture $(\neg a \vee b) \wedge (b \vee c)$ consists of multiple refutation steps which could consist of a single inference or nested inferences. Its first refutation step

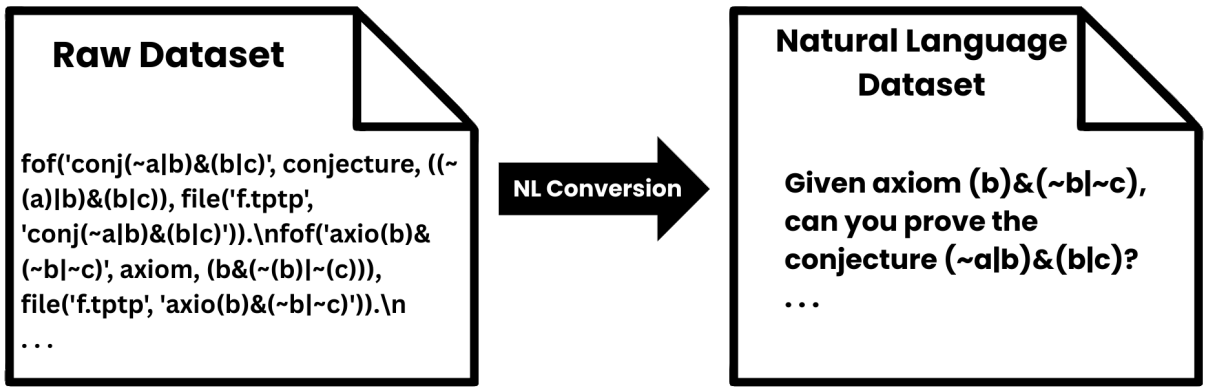

Figure 1: Natural Language Equivalence of Raw Dataset

would have two inferences where the first is to perform an operation of assuming the negation of the conjecture. Then, the second inference would be to simplify it. This produced the sub-result ¬((¬a ∨b) ∧(b ∨c)) which the second refutation step takes in. This operation continues until the last refutation step is reached. Throughout the entire process, the algorithm checks for particular key words that indicates what operations were performed on each step and the derived sub-results and final result. For each check that is made, a natural language equivalent is created and stored.

3.3. Large Language Model Fine Tuning

The seven billion parameter version Qwen2 for instruction behavior was chosen as the starting point for fine-tuning. Quen2 is a set of foundational models where the most complex version with 72 billion parameters performs well on multiple varieties of tasks [15]. Qwen2 was specifically chosen for this purpose due to its open source nature and the multiple versions with differing parameter sizes. Hardware and computational limitations prevented the inclusion of larger models. However, should performance improve on the smaller models that would give evidence that further performance gains may be available on larger instances of those same models.

Low-Rank Adaptation (LORA) of large language models is a technique for training models in such a way that it decreases the number of trainable parameters through reparameterization [16]. This allows larger models to be effectively trained on reduced hardware and is a type of parameter-efficient fine-tuning [17]. LORA was used here to efficiently fine-tune Quen2-7B-Instruct on the dataset of proofs converted to natural language. These were done using the Transformers library [18] and LLAMA-Factory [19]. The model was trained for 3 epoch with a learning rate of $1.0e-4$, a batch size of 1, and using PyTorch's ADAMW optimizer [20] with $\beta = (0.9, 0.999)$ and $\epsilon = 1e-08$. When all was complete these models were considered fine-tuned and suitable for evaluation on the chosen benchmark tasks.

4. Evaluation Benchmarks

The performance of these fine-tuned models is compared against the performance of the unspecialized language models on which they are based. Reasoning can be construed broadly from the more direct logic proof generation this paper used for training to things such as mathematical reasoning, mechanical reasoning, and spacial reasoning among many others. To illustrate this broad perspective this paper's benchmarks were chosen to model a wide variety of reasoning tasks. These were the Grade School Math 8K (GSM8K) [21], Massive Multitask Language Understanding-Redux (MMLU-Redux) [22], Code Reasoning, Understanding, and eXecution Evaluation (CRUXEval) [23], and ZebraLogic [24] benchmark tasks. ZeroEval [25] was used as the framework for evaluating the various models performance on these tasks. Two variations of the Qwen2 model were chosen to evaluate the performance of this paper's approach; both are instruct versions which are designed to follow prompted instructions. The smaller version is Qwen2 1.5B-Instruct which has roughly 1.5 billion parameters and the larger version

is Qwen2 7B-Instruct which has roughly 7 billion parameters. These were chosen due to hardware constraints and are suitable for fine-tuning. This approach can scale to different language models or larger models with more parameters without major alteration.

4.1. Grade School Math 8K

GSM8K is a collection of mathematical reasoning problems designed to be solvable with simple arithmetic operations. Each problem takes two to eight steps to solve and are designed not to follow a template for problem construction [21]. These problems also require an understanding of words and their meanings which differ from the purely symbolic fine-tuning data that was used for this paper's methodology.

4.2. Massive Multitask Language Understanding-Redux

MMLU-Redux is a curated subset of MMLU [26] which is large dataset of questions that span a variety of topics. MMLU-Redux includes 5,700 manually re-annotated problems which span 57 topic areas including logical fallacies, formal logic, economics, history, and medicine [22]. This benchmark requires the language model to internalize a large amount of background information while also including many problems which require formal reasoning skills.

4.3. Code Reasoning, Understanding, and eXecution Evaluation

CRUXEval is a benchmark dataset made up of 800 Python functions and input-output pairs [23]. Each problem can either specify an input and the function and request the language model to produce the correct output or request the language model to produce what input would produce the specified output from the function. Answering these questions indicates how the language model understands and reasons about programming constructs and the logic necessary to evaluate provided functions.

4.4. ZebraLogic

Zebra Logic is benchmark made up of Constraint satisfaction problems [24]. The language model is provided a set of houses and a set of owners. The model is also given some constraints about which house has what items, which people own which items, and some information relating one item to another house. The final goal being the matching of each house to each owner. These problems require the language model to identify a complex relationship of facts, a large number of constraints, and apply some logic to find the solution.

5. Results

Overall using synthetically generated proofs by contradiction does not improve model performance. To some degree the specialization can decrease performance for smaller models without substantially improving performance for larger models. For each case the number of non-answers is the number of questions that did not include sufficient information to check if an answer is correct or not divided by the total number of questions. The accuracy of each example is the total number of correctly answered questions divided by the total number of overall questions. In the case of GSM8K the base models outperformed this paper's fine-tuned models either by 1.97 for the larger models or a difference as great as 20.92 for the smaller models as seen in Table 1.

The number of No Answers is similar when comparing both versions of models. For the MMLU-Redux problems the fine-tuned model again underperformed the base models with a slight difference of only 0.54 for the larger models as seen in Table2. The number of No Answers is similarly close. As for CRUXEval task the results are similar with only a difference of 1 between the two larger models accuracy as seen in Table3. ZebraLogic also has a similar result where the base model outperforms the fine-tuned model by the small amount of 1.3 as seen in Table4.

Table 1
Grade School Math 8K

Model	Accuracy	No Answer	Total Problems
Qwen2 7B-Instruct	80.06	0	1319
Qwen2 7B Proposition FT	78.09	0.08	1319
Qwen2 1.5B-Instruct	43.29	4.78	1319
Qwen2 1.5B Proposition FT	22.37	4.25	1319

Table 2
Massive Multitask Language Understanding-Redux

Model	Accuracy	No Answer	Total Problems
Qwen2 7B-Instruct	67.53	0.72	2778
Qwen2 7B Proposition FT	66.99	1.04	2778
Qwen2 1.5B-Instruct	41.11	7.74	2778
Qwen2 1.5B Proposition FT	38.52	13.32	2778

Table 3
Code Reasoning, Understanding, and eXecution Evaluation

Model	Accuracy	No Answer	Total Problems
Qwen2 7B-Instruct	37.88	0.12	800
Qwen2 7B Proposition FT	36.88	0.38	800
Qwen2 1.5B-Instruct	9.88	0.62	800
Qwen2 1.5B Proposition FT	11.25	3.62	800

Table 4
Zebra Grid Evaluation

Model	Accuracy	Small Puzzles	Medium Puzzles	No Answer	Total Problems
Qwen2 7B-Instruct	8.4	26.25	0	24.4	1000
Qwen2 7B Proposition FT	7.1	21.88	0.36	20.7	1000
Qwen2 1.5B-Instruct	2.8	8.75	0	16.1	1000
Qwen2 1.5B Proposition FT	1.4	4.38	0	18.9	1000

6. Conclusion

Learning propositional logic proofs based on contradiction does not sufficiently translate to overall reasoning skills. This is not an entirely unexpected result as many of the benchmark tasks require the use of first order logic which is a more sophisticated than the propositional logic used to train the fine-tuned models here.

Declaration on Generative AI

The core methodology of this research involved the use of Qwen for fine tuning and answering reasoning based question benchmarks, as described in the paper. No generative AI tools were employed for the writing or editing of this manuscript beyond the research itself. The author takes full responsibility for the publication's content.

References

[1] Y. Wan, W. Wang, Y. Yang, Y. Yuan, J.-t. Huang, P. He, W. Jiao, M. R. Lyu, Logicasker: Evaluating and improving the logical reasoning ability of large language models, arXiv preprint arXiv:2401.00757 (2024).

[2] P. Han, P. Song, H. Yu, J. You, In-context learning may not elicit trustworthy reasoning: A-not-b errors in pretrained language models, arXiv preprint arXiv:2409.15454 (2024).

[3] C. Franks, Propositional Logic, in: E. N. Zalta, U. Nodelman (Eds.), The Stanford Encyclopedia of Philosophy, Winter 2024 ed., Metaphysics Research Lab, Stanford University, 2024.

[4] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, W. Zhou, J. Coady, D. Peng, Y. Qiao, L. Benson, L. Sun, A. Wardle-Solano, H. Szabó, E. Zubova, M. Burtell, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, A. Fabbri, W. M. Kryscinski, S. Yavuz, Y. Liu, X. V. Lin, S. Joty, Y. Zhou, C. Xiong, R. Ying, A. Cohan, D. Radev, FOLIO: Natural language reasoning with first-order logic, in: Y. Al-Onaizan, M. Bansal, Y.-N. Chen (Eds.), Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Miami, Florida, USA, 2024, pp. 22017–22031. URL: https://aclanthology.org/2024.emnlp-main.1229/. doi:10.18653/v1/2024.emnlp-main.1229.

[5] Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. M. McAleer, A. Q. Jiang, J. Deng, S. Biderman, S. Welleck, Llemma: An open language model for mathematics, in: The Twelfth International Conference on Learning Representations, 2024. URL: https://openreview.net/forum?id=4WnqRR915j.

[6] S. Polu, I. Sutskever, Generative language modeling for automated theorem proving, 2020. URL: https://arxiv.org/abs/2009.03393. arXiv:2009.03393.

[7] E. First, M. N. Rabe, T. Ringer, Y. Brun, Baldur: Whole-proof generation and repair with large language models, in: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 1229–1241. URL: https://doi.org/10.1145/3611643.3616243. doi:10.1145/3611643.3616243.

[8] T. Nipkow, L. C. Paulson, M. Wenzel, Isabelle/HOL — A Proof Assistant for Higher-Order Logic, volume 2283 of *LNCS*, Springer, 2002.

[9] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Miłoś, Y. Wu, M. Jamnik, Thor: Wielding hammers to integrate language models and automated theorem provers, Advances in Neural Information Processing Systems 35 (2022) 8360–8373.

[10] A. Singh, N. P. Patel, A. Ehtesham, S. Kumar, T. T. Khoei, A survey of sustainability in large language models: Applications, economics, and challenges, in: 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC), 2025, pp. 00008–00014. doi:10.1109/CCWC62904.2025.10903774.

[11] E. Rich, Automata, Computability and Complexity: Theory and Applications, Pearson Prentice Hall, 2008.

[12] G. Sutcliffe, The tptp problem library and associated infrastructure, J. Autom. Reason. 59 (2017) 483–502. URL: https://doi.org/10.1007/s10817-017-9407-7. doi:10.1007/s10817-017-9407-7.

[13] A. Marchiafava, A. Sen, Towards determining how deep neural models learn to reason, in: 2025 AAAI-Make Spring Symposium Series, Association for the Advancement of Artificial Intelligence, 2025.

[14] S. Schulz, E – a brainiac theorem prover, Journal of AI Communications 15 (2002) 111–126.

[15] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Fan, Qwen2 technical report, arXiv preprint arXiv:2407.10671 (2024).

[16] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, 2021. URL: https://arxiv.org/abs/2106.09685. arXiv:2106.09685.

[17] N. Ding, Y. Qin, G. Yang, F. Wei, Y. Zonghan, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H.-T. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, M. Sun, Parameter-efficient fine-tuning of large-scale pre-trained language models, Nature Machine Intelligence 5 (2023) 1–16. doi:10.1038/s42256-023-00626-4.

[18] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, A. M. Rush, Huggingface's transformers: State-of-the-art natural language processing, 2020. URL: https://arxiv.org/abs/1910.03771. arXiv:1910.03771.

[19] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, Z. Feng, Y. Ma, Llamafactory: Unified efficient fine-tuning of 100+ language models, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), Association for Computational Linguistics, Bangkok, Thailand, 2024. URL: http://arxiv.org/abs/2403.13372.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, 2019. URL: https://arxiv.org/abs/1912.01703. arXiv:1912.01703.

[21] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, J. Schulman, Training verifiers to solve math word problems, 2021. URL: https://arxiv.org/abs/2110.14168. arXiv:2110.14168.

[22] A. P. Gema, J. O. J. Leang, G. Hong, A. Devoto, A. C. M. Mancino, R. Saxena, X. He, Y. Zhao, X. Du, M. R. Ghasemi Madani, C. Barale, R. McHardy, J. Harris, J. Kaddour, E. Van Krieken, P. Minervini, Are we done with MMLU?, in: L. Chiruzzo, A. Ritter, L. Wang (Eds.), Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Association for Computational Linguistics, Albuquerque, New Mexico, 2025, pp. 5069–5096. URL: https://aclanthology.org/2025.naacl-long.262/.

[23] A. Gu, B. Rozière, H. Leather, A. Solar-Lezama, G. Synnaeve, S. I. Wang, Cruxeval: A benchmark for code reasoning, understanding and execution, 2024. URL: https://arxiv.org/abs/2401.03065. arXiv:2401.03065.

[24] B. Y. Lin, R. L. Bras, Y. Choi, Zebralogic: Benchmarking the logical reasoning ability of language models, 2024. URL: https://huggingface.co/spaces/allenai/ZebraLogic.

[25] B. Y. Lin, ZeroEval: A Unified Framework for Evaluating Language Models, 2024. URL: https://github.com/WildEval/ZeroEval.

[26] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, J. Steinhardt, Measuring massive multitask language understanding, 2021. URL: https://arxiv.org/abs/2009.03300. arXiv:2009.03300.

A. Training Data Example

Problem: $(b) \wedge (\neg b \vee \neg c) \models (\neg a \vee b) \wedge (b \vee c)$

System Message: You are Qwen, created by Alibaba Cloud. You are a helpful assistant.
Input: Given axiom axio(b)(b| c), can you prove the conjecture conj(a|b)(b|c)?
Output: step 1) Begin by assuming the negation of the conjecture which is simplified to ¬(((¬a|b)∧(b|c))). \nstep 2) Converting the result of step 1 to negated normal form Converting the result of step 1 to negated normal form which is distributed to (((¬b|a)∧(¬c|a))∧((¬b|¬b)∧(¬c|¬b))). \nstep 3) Applying simplification to the result of step 2 provides us with (b∧(¬b|¬c)). \step 4) Using split conjunction on to the result of step 3 produces (¬b|¬b). \nstep 5) Applying negated normal form to the result of step 4 renders (b∧(¬b|¬c)). \step 6) Now we apply clause normalization to step 5 results in (¬b). \nstep 7) Using split conjunction on to the result of step 6 produces (b). \nstep 8) Given step 6 produced (¬b) and step 7 (b), we have derived a contradiction.