LLMSR@XLLM25: SWRV: Empowering Self-Verification of Small Language Models through Step-wise Reasoning and Verification

Danchun Chen[1] **Yixin Cao**[1] **Liangming Pan**[2]
[1]Fudan University [2]University of Arizona
{dcchen20, yxcao}@fudan.edu.cn liangmingpan@arizona.edu

Abstract

Large language models (LLMs) have demonstrated impressive reasoning capabilities through Chain-of-Thought (CoT) prompting. However, their reasoning processes remain inexplicable and uncontrollable. In this paper, we introduce a ***Step-Wise Reasoning and Verification*** (SWRV) framework, designed as a two-stage *Parser–Verifier* pipeline, that decomposes the reasoning process into discrete inference steps and rigorously validates each one. Our *Parser* extracts problem constraints and the sequence of reasoning steps from the LLM's output, and our *Verifier*, either LM-based or powered by a symbolic solver, checks the logical correctness of every step. To ensure robust parsing, we fine-tune a compact LM on a small, high-quality annotation set generated by a more capable LM. Experiments on the *LLMSR* dataset built atop *LogiQA* show significant gains over baselines, illustrating the effectiveness of our method for step-wise analysis of LLM reasoning[1].

1 Introduction

Large language models (LLMs) have propelled significant advances in natural language processing, yet they continue to struggle with tasks that demand precise multi-step logical reasoning—especially those involving multiple constraints, nested subproblems, or domain-specific knowledge. While recent developments in Chain-of-Thought (CoT) prompting have enhanced LLM inference and reasoning abilities, the resulting chains of reasoning often lack reliability and interpretability. This can severely hinder downstream performance, as models fail to consistently apply logical rules or verify intermediate steps (Paul et al., 2024). Recent work has demonstrated that very large LLMs (OpenAI, 2024) are capable of self-correcting their outputs through iterative refinement, ushering in a new paradigm for improving model reliability. However, such performance typically depends on massive model sizes and extensive training data, rendering these methods impractical in resource-constrained scenarios. More pertinently, Zhang et al. 2024 found that, for smaller models, overall reasoning performance is bottlenecked not by the ability to refine answers but by the weakness of the verifier module itself. This insight underscores the pressing need for robust, fine-grained self-verification, particularly in settings where only compact models and small-scale annotated data are available. Despite this, most existing approaches focus on validating the entire CoT holistically, without systematically analyzing the validity of individual reasoning steps.

This drives our exploration of step-level reasoning verification methods for small language models. In this work, we propose SWRV, a stronger step-wise self-verification framework as shown in Figure 1 for small LMs (i.e., *Llama-3-8B-Instruct*) using minimal data. In the framework, we perform fine-grained question and CoT analysis to verify each individual inference step within the following pipeline. We first prompt *Llama-3-8B-Instruct* to produce CoTs for *LogiQA* (Liu et al., 2020) questions, then conduct fine-grained parsing of each question and each CoT reasoning step, followed by rigorous verification. For the parser module, we employ rule-based prompting of the small LM and fine-tuning using annotations generated by LM. For verifier, in addition to LM-based inference, we integrate a deterministic symbolic solver Z3 by translating the problem and reasoning steps into symbolic formulas and performing formal inference. Our method aims to deliver a fine-grained, precise self-verifier that improves the self-correction and reasoning ability of small LMs and enables more granular process-level reward modeling.

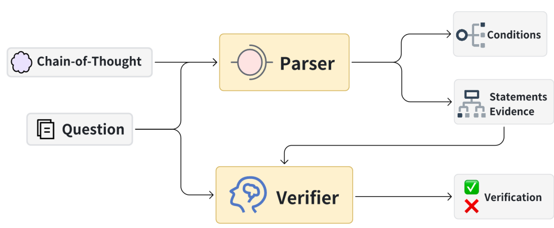

Figure 1: Overview of our SWRV framework.

[1]Code is publicly available at https://github.com/Teganone/XLLM_LLMSR.

Proceedings of the 1st Joint Workshop on Large Language Models and Structure Modeling (XLLM 2025), pages 322–335
August 1, 2025

2 Problem Formulation of Self-Verification

As illustrated in Figure 1, we equip a language model with fine-grained self-verification by decomposing its reasoning process into two complementary modules: i) Step-wise Reasoning Parsing, in which both the question and its CoT are parsed into elementary units; and ii) Step-wise Reasoning Verification, in which each inferred unit is rigorously checked for logical validity.

Step-wise Reasoning Parsing The step-wise reasoning parsing mainly focus on parsing the question and CoT. Given a reasoning question *Q* and its CoT, the parser extract all necessary conditions from the question and reasoning steps from the CoT. The parsing for the question and the CoT can be integrated or processed separately, allowing for flexible adjustment of parameters to enhance parsing accuracy.

Step-wise Reasoning Verification For step-by-step reasoning process in CoT, the verifier checks its correctness and marks it as True or False. A verifier, which can either be intrinsic (the LM itself) or extrinsic (an external signal), then decides whether the statement in the step is adequately supported by the corresponding evidence. If the verifier is unable to draw a conclusion, it defaults to considering the reasoning step as correct.

Decoupling parsers from the verifier offers significant advantages over an "all-in-one" design. Firstly, we can freely parameterize each module, for instance, by using fine-tuning or prompting. Secondly, it reduces the difficulty of training each module since the model only needs to focus on one specific capability, either task-specific parsing or step-level reasoning verification. Finally, it makes it possible to integrate deterministic verifiers that incorporate external symbolic solvers.

3 Methodology

As shown in Figure 2, the inputs for our framework consist of a natural language logical reasoning question *Q*, along with the Chain-of-Thought *CoT* genearted by prompting LM to solve the question *Q*. Our goal is to parse *Q* into separate conditions *QP* and the entire *CoT* into separate reasoning steps, each step composing a statement and evidence *R*, and verify the correctness v of each step.

Hence, the final output of each pair of inputs consists of the following elements:

- parsed question conditions

$$\mathrm{QP} = \{c_1, \ldots, c_n\},$$

- parsed & verified reasoning steps

$$CP = \{(s_1, e_1, v_1), \ldots, (s_m, e_m, v_m)\},$$

where s_i is a statement, e_i its supporting evidence and v_i the verification result of (s_i, e_i), each $v_i \in \{True, False\}$.

To achieve this end-to-end pipeline, our SWRV framework is organized into two complementary components. The Parser module comprises i) a *Question Parser* submodule that ingests Q and extracts the individual conditions c_i, and ii) a *CoT Parser* submodule that partitions the CoT into discrete inference steps (s_i, e_i). The Verifier module then takes these parsed outputs, uses a *Symbolic Formulator* to translate both the set of conditions $\{c_i\}$ and each pair (s_i, e_i) into formal logical expressions, and employs an *SMT-Based Checker* (implemented with the *Z3 solver*) to deterministically determine the validity of each inference. By pipelining parsing and verification, our framework delivers a finely grained, formally certified assessment of the entire reasoning trajectory.

3.1 Parser

The parsering module is divided into two stages: prompting a LM to generate question parsing and CoT parsing, and then optionally using the obtained question parsing and CoT parsing together with the original question and CoT as input, with the target question parsing and CoT parsing as labels to supervise the fine-tuning of the LM.

Stage 1: Generating Question Parsing and CoT Parsing A logical reasoning problem typically comprises four parts: problem description, constraints, query, and options. The LM is tasked with extracting all the conditions present in the first three parts. For each question, we instruct the LM to generate the CoT that outlines the problem-solving process. From this CoT, we extract the statement and evidence for each reasoning step. Considering the nature of logical reasoning problems, the arguments mainly stem from the conditions given in the problem and may be related to the options; meanwhile, the evidence often consists of the problem conditions or intermediate conclusions.

Divided by analytical content, we have set up three parsers: a question parser, a CoT parser, and a combined parser, which, respectively, parse the question, the CoT, and both simultaneously. This separate design allows us to individually adjust the parameters for each parser. The parsing process can flexibly choose any combination of these three parsers. In our prompt, we enforce the corresponding rules that require the output of the parsing to remain as semantically consistent as possible with the original question or CoT. We also use one-shot examples to format the output structure and few-shot examples to enrich the diversity of logical question cases. For detailed prompt information, please refer to the Appendix B.2 and B.3.

Stage 2: Supervised Fine-tuning of the Parser To effectively fine-tune LM with minimal data, we supplement the original question and CoT with the question

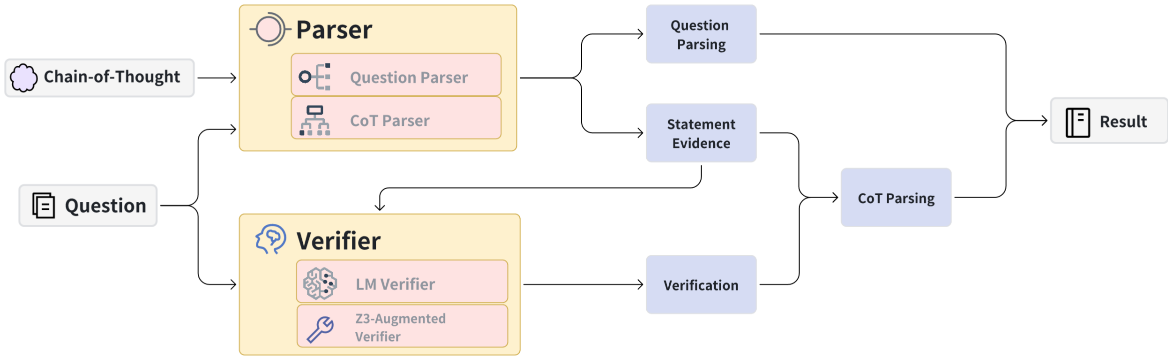

Figure 2: Overview of more detailed SWRV framework.

parsing and CoT parsing obtained from *Stage 1* as inputs. This indicates the LM only need to learn how to refine the existing parsing results instead of having to learn parsing rules from scratch. For detailed prompt information, please refer to the Appendix B.4.

3.2 Verifier

We employ two approaches for verifying the step-wise reasoning: one using LM inference, and the other aided by a deterministic symbolic solver (i.e. *Z3*).

3.2.1 LLM Verifier

For each reasoning step parsed into statements and evidence by the parser, we prompt the LM, with the original or parsed question along with the statements and evidence, to directly verify whether a given statement can be deduced from the evidence. The correct conclusion is "true" and an incorrect deduction is "false". Detailed system prompt and user prompt could be seen in the Appendix B.5.

3.2.2 Z3-Augmented Verifier

Z3 is a high-performance SMT (Satisfiability Modulo Theories) solver developed by Microsoft[2]. It can decide the satisfiability of first-order logic formulas over a rich set of theories—such as linear arithmetic, bit-vectors, arrays, and uninterpreted functions—and is widely used for program verification, symbolic reasoning, and formal analysis.

Inspired by Pan et al. 2023, our *Z3-Augmented Verifier* formulates each reasoning step as a symbolic problem and then invokes Z3 Prover for deterministic validation. Concretely, we define a custom intermediate representation (IR) that bridges natural language and formal logic. The LM is prompted to translate both the original question and each parsed inference step into this IR, which is then translated into executable code. By running through Z3 solver, we obtain a definitive “true” or “false” verdict on the logical correctness of each step.

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

Problem-and-Reasoning Formulator As shown in figure 3, given a natural language logical reasoning question Q and its step-wise reasoning R, we prompt a LM to translate them into self-defined intermediate representations. These are then converted into a formal, SMT-compatible representation by a code translator. This encoding captures both the question description and the step-wise reasoning in a symbolic language understandable by Z3.

Symbolic Reasoner We invoke Z3 as a deterministic SMT solver over the encoded problem. Z3 efficiently checks the satisfiability, performs the required logical inferences, and produces a symbolic answer. Because Z3’s algorithms are sound and complete for the supported theories, the correctness of the answer is guaranteed when the initial encoding is faithful.

Self-Refiner For complex problems and intricate reasoning, it is challenging for the LM to generate correct logical expressions immediately. Therefore, we introduce a self-refinement module that returns syntax errors from the Z3 solver back to the LM, guiding it to generate correct logical programs. This iterative refinement continues until a valid program is generated or the maximum number of attempts is reached.

Result Interpreter Finally, we use a rule-based interpreter to map the symbolic output back to natural language, providing the final answer.

Appendix B.6 shows detailed prompts of *Problem & Reasoning Symbolic Formulation*.

4 Experimental Setup

4.1 Datasets

A typical sample is stored in JSON format. Detailed examples are given in Appendix A.

4.2 Model Architecture and Fine-tuning

We use *Llama-3-8B-Instruct* as our base model for both the parser and the LM verifier. Considering *Llama-3-8B-Instruct*’s lacking in groundtruth of Z3 syntax, we employ *O3-mini-high* as the base model to generate Z3

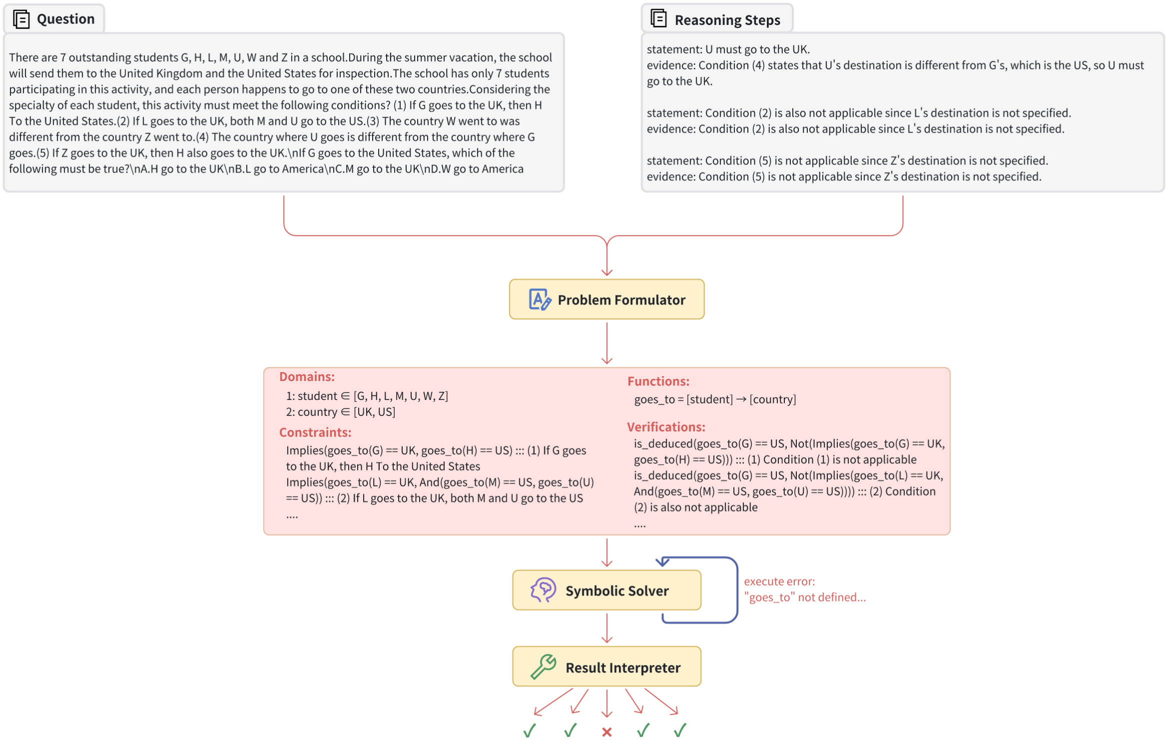

Figure 3: Framework of Z3-Augmented Verifier

formulations.

We set the temperature as $0.2 - 0.3$ in the Question Parser, $0.5 - 0.6$ in the Combined and Cot Parser. For Verification module, we set temperature as $0.1 - 0.2$ in the *Llama Verifier* and the iteration of self-refinement as 3.

We use the 24 samples in the datasets to fine-tune the base *Llama-3-8B-Intruct* with *LoRA*. We set the low-rank dimension as 32, the learning rate as $2e{-}5$, training epochs as 6, batch size as 2. All our experiments can be conducted on 2×H20 GPU with 96GB of memory.

4.3 Evaluation Metrics and Baseline

Question_Macro_F1 the macro-averaged F1 score computed over the set of all atomic conditions that must be extracted from the input question. Each distinct condition constitutes its own class; true positives, false positives, and false negatives are counted per class, and then F1 is averaged uniformly across classes. This metric thus captures the model's ability to recover every necessary constraint for downstream reasoning, regardless of class frequency.

Statement_Macro_F1 denotes the macro-averaged F1 score for segmenting and identifying individual reasoning statements and their associated evidence spans within the chain of thought. We treat each span type (statement vs. evidence) as a separate class and evaluate extraction quality via both lexical and semantic overlap against ground truth. Precision and recall are computed per class and averaged, ensuring balanced evaluation across all span categories.

Statement_Evidence_Macro_F1 measures the macro-averaged F1 over pairwise links between extracted statements and their corresponding evidence. Each possible statement–evidence pairing is treated as a binary classification task (linked vs. unlinked). We compute class-wise precision and recall for the "linked" label and average the resulting F1 scores across all statement–evidence candidates to assess the model's ability to reconstruct the intended argumentative structure.

Reasoning_F1 the macro-averaged F1 score for the final entailment verification between each correctly extracted statement–evidence pair. We frame logical deduction as a binary entailment decision (entails vs. does not entail). For all validated pairs, we compute precision and recall on the "entails" class and average the F1 scores uniformly, thereby evaluating end-to-end correctness of the tool-augmented reasoning pipeline.

Baseline We invoke *Llama-3-8B-Instruct* to directly parse and verify the data, without any rule setting or fine-tuning, as our baseline.

5 Results

Table 1 and Table 2 presents the primary evaluation results for our base and fine-tuned LM respectively. The results include four performance metrics detailed in 4.3. And we have Four major findings.

5.1 Main Findings

1) Using a lower temperature for *Question Parsing* ensures deterministic and robust outputs. We found that setting a relatively low temperature (around 0.2) for *Question Parsing* and subsequent reasoning yields more deterministic and robust outputs. A lower temperature helps reduce randomness during question generation, ensuring that the parsing results remain logically consistent and rigorous.

2) Striking a balance in temperature during *CoT Parsing* leads to reasoning that is both comprehensive and accurate. For *CoT Parsing*, it is crucial to strike a balance — the temperature should not be too low nor too high. A recommended range is between 0.5 and 0.6. If the temperature is too low, the generated content might be overly fixed and could miss out on important details or the richer process of reasoning. Conversely, if it is too high, the outputs can become too divergent, making it more difficult to extract relevant evidence and accurately perform subsequent verification. Thus, balanced temperature settings are essential to obtain both comprehensive and accurate reasoning chains.

3) Fine-tuned parsers do not outperform the rule-based prompting base parser. The main reason might be limited training set (only 24 examples) combined with the rich, complex rules specified in the prompt, which making it insufficient for a small LM to internalize those rules.
Besides, LM Fine-tuned by Larger LM (e.g. *O3-mini-high*) has better performance in all metrics.

4) Integrating the Z3 solver significantly enhances the accuracy and overall performance of step-wise reasoning verification. In our experiments, the Z3 solver's success rate is around 72%, leading to a *Reasoning_F1* score of 0.078. It is expected that if the parsing and logical expression generation achieved a 100% success rate, the *Reasoning_F1* score could approach the reasoning performance demonstrated by *O3-mini-high*, around 85.07% better than the *LM Verifier* itself. This implies that the symbolic solvers could enhance the step-wise reasoning verification.

5.2 Case Study

Figure 4 and Figure 5 together exemplify the comprehensive reasoning and verification workflow of our SWRV framework. In Figure 4, we show how a complex recruit-assignment puzzle is first decomposed by the *Parser*: the problem statement yields a set of conditions, and the accompanying CoT is split into individual inference steps, each paired with its evidence. Building on this, Figure 5 presents the symbolic instantiation of the same case, in which entities and constraints extracted from the natural-language quetsion description are formalized into logical expressions. Each inference step is translated into a corresponding verification formula, enabling the Z3 solver to deterministically check its validity. By moving from free-form text to symbolic logic, our framework not only attains stronger correctness guarantees than pure language-model reasoning but also furnishes interpretable, step-wise feedback. This structured decomposition thus enhances both the accuracy of self-verification and the transparency of the reasoning process, addressing a key limitation of existing CoT analysis approaches.

6 Conclusion

The paper presents SWRV, a two-stage *Parser–Verifier* framework specifically designed for small LMs and limited data to parse and verify every step in the reasoning process. By decomposing problems in a fine-grained manner and complementing this with a verification mechanism based on symbolic solving, the method significantly enhances the accuracy and robustness of small LMs in logical reasoning tasks.

Future work may explore the expansion of annotated datasets, thereby optimizing the fine-tuning of the parser. Besides, further refinement of natural-to-symbolic translation methods, could also be explored to fortify verification module in diverse reasoning scenarios.

Acknowledgements

This work is supported by my respectful advisors Prof. Cao and Prof. Pan. We also acknowledge support from the hosts (Jia et al., 2025).

References

Zixia Jia, Zilong Zheng, Yang Liu, Jiaqi Li, and Jun Bai. 2025. Llm for structural reasoning. https://xllms.github.io/LLMSR/#. Date: 2025-02-10.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning.

OpenAI. 2024. Gpt-4 technical report.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2024. Refiner: Reasoning feedback on intermediate representations.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024. Small language models need strong verifiers to self-correct reasoning.

Question: There are seven recruits recruited at a university? F, G, H, I, W, X, and Y.One of them is a communications soldier, three are engineering soldiers, and the other three are transport soldiers.The conditions for the assignment of recruits to the army are as follows? H and Y must be assigned to the same arm.F and G cannot be assigned to the same arm.If X is assigned as a transport soldier, then W is assigned as an engineering soldier.Assign F as an engineering soldier.

If X is not assigned as an engineer, which of the following statements may be true?
A.W and G are assigned as engineering soldiers
B.H and W are assigned as transport soldiers.
C.F and Y are assigned as engineering soldiers
D.H and W are assigned as engineering soldiers"

CoT: Since X is not assigned as an engineer, it must be assigned as a communications soldier or a transport soldier. If X is a transport soldier, then W is an engineering soldier, and H and Y must be assigned to the same arm. However, F and G cannot be assigned to the same arm, so F must be an engineering soldier. This means that H and Y are assigned to the same arm, which is the transport arm. Therefore, H and W are assigned as transport soldiers.

Parsed Conditions (Question Parsing):

```
[
    "There are seven recruits recruited at a university? F, G, H, I, W, X, and Y.One of them is a communications soldier, three are engineering soldiers, and the other three are transport soldiers",
    "H and Y must be assigned to the same arm",
    "F and G cannot be assigned to the same arm",
    "If X is assigned as a transport soldier, then W is assigned as an engineering soldier",
    "Assign F as an engineering soldier",
    "If X is not assigned as an engineer"
]
```

Parsed Reasoning Steps:

```
{
    "statement": "X is assigned as a communications soldier or a transport soldier.",
    "evidence": "Since X is not assigned as an engineer, it must be assigned as a communications soldier or a transport soldier.",
},
{
    "statement": "If X is a transport soldier, then W is assigned as an engineering soldier, and H and Y must be assigned to the same arm.",
    "evidence": "If X is a transport soldier, then W is assigned as an engineering soldier, and H and Y must be assigned to the same arm.",
},
{
    "statement": "F must be an engineering soldier.",
    "evidence": "However, F and G cannot be assigned to the same arm, so F must be an engineering soldier.",
}
```

Figure 4: Example of parsed question and reasoning steps for a logical reasoning problem.

Question: There are seven recruits recruited at a university? F, G, H, I, W, X, and Y.One of them is a communications soldier, three are engineering soldiers, and the other three are transport soldiers.The conditions for the assignment of recruits to the army are as follows? H and Y must be assigned to the same arm.F and G cannot be assigned to the same arm.If X is assigned as a transport soldier, then W is assigned as an engineering soldier.Assign F as an engineering soldier.

If X is not assigned as an engineer, which of the following statements may be true?
A.W and G are assigned as engineering soldiers
B.H and W are assigned as transport soldiers.
C.F and Y are assigned as engineering soldiers
D.H and W are assigned as engineering soldiers"

Generated Symbolic Problem Formulations:

Declarations:

```
recruits = EnumSort([F, G, H, I, W, X, Y])
arms = EnumSort([communications, engineering, transport])
arm_of = Function([recruits] -> [arms])
```

Constraints:

```
Count([r: recruits], arm_of(r) == communications) == 1 ::: (Count of communications soldiers)
Count([r: recruits], arm_of(r) == engineering) == 3 ::: (Count of engineering soldiers)
Count([r: recruits], arm_of(r) == transport) == 3 ::: (Count of transport soldiers)
arm_of(H) == arm_of(Y) ::: (1) H and Y must be assigned to the same arm
arm_of(F) != arm_of(G) ::: (2) F and G cannot be assigned to the same arm
Implies(arm_of(X) == transport, arm_of(W) == engineering) ::: (3) If X is assigned as a transport soldier, then W is assigned as an engineering soldier
arm_of(F) == engineering ::: (4) F is assigned as an engineering soldier
arm_of(X) != engineering ::: (Assumption: X is not assigned as an engineer)
```

Reasoning Step:

Statement: X is assigned as a communications soldier or a transport soldier.

Evidence: Since X is not assigned as an engineer, it must be assigned as a communications soldier or a transport soldier.

Generated Symbolic Verification Formulations:

```
is_deduced(arm_of(X) != engineering, Or(arm_of(X) == communications, arm_of(X) == transport)) ::: (1) X must be assigned as a communications soldier or a transport soldier
```

Predicted Verification: True

Reasoning Step:

Statement: F must be an engineering soldier.

Evidence: However, F and G cannot be assigned to the same arm, so F must be an engineering soldier.

Generated Symbolic Verification Formulations:

```
is_deduced(True, arm_of(F) == engineering) ::: (3) F must be an engineering soldier
```

Predicted Verification: False

other step-wise reasoning examples...

Figure 5: Example of symbolic problem and reasoning formulation and verification for a logical logical problem.

Approach	Question_F1	Statement_F1	Statement_Evidence_F1	Reasoning_F1
Baseline	0.5702	0.3341	0.0852	0.0326
Base Question Parser	0.7299	/	/	/
Base Combined Parser	0.7187	0.4247	0.168	/
Llama Verifier	/	/	/	0.067
O3-mini-high Verifier	/	/	/	0.124
Z3 Verifier (72%)	/	/	/	0.078

Table 1: Performance of various parsers and verifiers. “/” indicates that a given configuration was not designed or evaluated for that subtask (e.g., the Base Question Parser only performs question parsing, so its *Statement_F1*, *Evidence_F1*, and *Reasoning_F1* cells are marked “/”).

Preprocessor	Question_F1	Statement_F1	Statement_Evidence_F1
Baseline	0.7187	0.4247	0.168
Llama-3-8B-Instruct	0.6078	0.3979	0.1417
O3-mini-high	0.7171	0.4046	0.1862

Table 2: Performance of parsers fine-tuned on data preprocessed by *Llama-3-8B-Instruct* and *O3-mini-high*. "Baseline" denotes the original model without fine-tuning.

A Dataset

question: There are 7 outstanding students G, H, L, M, U, W and Z in a school. During the summer vacation, the school will send them to the United Kingdom and the United States for inspection. The school has only 7 students participating in this activity, and each person happens to go to one of these two countries. Considering the specialty of each student, this activity must meet the following conditions? (1) If G goes to the UK, then H To the United States. (2) If L goes to the UK, both M and U go to the US. (3) The country W went to was different from the country Z went to. (4) The country where U goes is different from the country where G goes. (5) If Z goes to the UK, then H also goes to the UK.
If G goes to the United States, which of the following must be true?
A. H go to the UK
B. L go to America
C. M go to the UK
D. W go to America
question_parsing: [
"The school has only 7 students participating in this activity, and each person happens to go to one of these two countries",
"If G goes to the UK, then H goes to the United States",
"If L goes to the UK, both M and U go to the US",
"The country W went to was different from the country Z went to",
"The country where U goes is different from the country where G goes",
"If Z goes to the UK, then H also goes to the UK",
"G goes to the United States"
]

answer: b
id: 162
cot: Since G goes to the United States, we need to analyze the conditions that follow. Condition (1) is not applicable since G is going to the US. Condition (2) is also not applicable since L's destination is not specified. Condition (3) does not provide any information about H, M, U, or W. Condition (4) states that U's destination is different from G's, which is the US, so U must go to the UK. Condition (5) is not applicable since Z's destination is not specified.
cot_parsing: [
{
"statement": "Condition (1) is not applicable",
"evidence": "G is going to the US",
"Verification": "true"
},
{
"statement": "Condition (2) is also not applicable",
"evidence": "L's destination is not specified",
"Verification": "true"
},
{
"statement": "Condition (3) does not provide any information about H, M, U, or W",
"evidence": "Condition (3)",
"Verification": "false"
},
{
"statement": "U must go to the UK",
"evidence": "Condition (4) states that U's destination is different from G's, which is the US",
"Verification": "true"
},

{
"statement": "Condition (5) is not applicable",
"evidence": "Z's destination is not specified",
"Verification": "true"
}
]
sel_idx: 92

B Prompt

B.1 Prompt For Question Parser

Given a question. The task is to generate "question_parsing" results based on the content of "question". The "question" could be logically divided into four parts: 1) problem description, 2) conditions/constraints, 3) query, and 4) options. The question parsing process involves extracting all conditions necessary for solving the question.
Evaluation: This task focuses on question parsing. Question parsing involves extracting all relevant conditions required to solve the problem. The Macro F1 score metric is used to evaluate question parsing performance.
Please generate output in JSON format based on the requirements below. The output must include one key-value pair:
"question_parsing": an array of strings used to extract all necessary constraints provided in the question for solving the problem. Only extract conditions from parts 1) problem description, 2) conditions/constraints, and any additional conditions present in part 3) query. Do not extract any details from the explicit query statement (e.g., phrases like "which is . . . ") or from the options. The problem itself should be extracted as the first condition.
Rules: Each constraint or condition in the original question should be treated as a whole; do not split or break down a single constraint or condition into smaller parts. Use the exact descriptions given in the original question without synonym substitution or additional embellishment, ensuring high consistency in both semantics and wording with the original text.
EXAMPLE 1:
question:
There are 7 outstanding students G, H, L, M, U, W and Z in a school. During the summer vacation, the school will send them to the United Kingdom and the United States for inspection. The school has only 7 students participating in this activity, and each person happens to go to one of these two countries. Considering the specialty of each student, this activity must meet the following conditions? (1) If G goes to the UK, then H To the United States. (2) If L goes to the UK, both M and U go to the US. (3) The country W went to was different from the country Z went to. (4) The country where U goes is different from the country where G goes. (5) If Z goes to the UK, then H also goes to the UK.
If G goes to the United States, which of the following must be true?
A. H go to the UK
B. L go to America
C. M go to the UK
D. W go to America
Example Output:
question_parsing:
"The school has only 7 students participating in this activity, and each person happens to go to one of these two countries",
"If G goes to the UK, then H goes to the United States",
"If L goes to the UK, both M and U go to the US",
"The country W went to was different from the country Z went to",
"The country where U goes is different from the country where G goes",
"If Z goes to the UK, then H also goes to the UK",
"G goes to the United States"
] output Example "question_parsing": ["The school has only 7 students participating in this activity, and each person happens to go to one of these two countries", "If G goes to the UK, then H To the United States", "If L goes to the UK, both M and U go to the US", "The country W went to was different from the country Z went to", "The country where U goes is different from the country where G goes", "If Z goes to the UK, then H also goes to the UK", "G goes to the United States"]
NOW ANALYZE THIS NEW QUESTION:
question:
{{question}}
Remember to analyze the specific content above, not the examples. Your output should be a valid JSON object with a "question_parsing" key.

B.2 Prompt For Combined Parser

Given a question and cot. The task is to generate "question_parsing" and "cot_parsing" results based on the content of "question" and "cot". The "question" could be logically divided into four parts: 1) problem description, 2) conditions/constraints, 3) query, and 4) options. The question parsing process involves extracting all conditions necessary for solving the question. The cot parsing process identifies all "statements" and their corresponding "evidence" within the context of the question conditions and the given cot content.
Evaluation: This task consists of two parts: Question parsing and cot parsing. Question parsing involves extracting all relevant conditions required

to solve the problem. The Macro F1 score metric is used to evaluate question parsing performance. The process of extracting statements and evidence is similar to Discourse Parsing. Correct extraction of statements or evidence from the cot is crucial at the outset. Next, the pairwise relationship between a specific statement and its corresponding evidence is assessed (a statement should be followed by its related evidence from the cot). Both semantic and lexical similarity are used to evaluate the accuracy of statements and evidence predictions. The final evaluation metric is the Macro F1 score, applied to both statement parsing and statement-evidence pair extraction. Whether the "statement" can be deduced from the "evidence" logically, answer with only with "true" or "false". Please generate output in JSON format based on the requirements below. The output must include two key-value pairs:
"question_parsing": an array of strings used to extract all necessary constraints provided in the question for solving the problem. Only extract conditions from parts 1) problem description, 2) conditions/constraints, and any additional conditions present in part 3) query. Do not extract any details from the explicit query statement (e.g., phrases like "which is ...") or from the options. The problem itself should be extracted as the first condition.
"question_parsing": an array of strings used to extract all necessary constraints provided in the question for solving the problem. Only extract conditions from parts 1) problem description, 2) conditions/constraints, and any additional conditions present in part 3) query. Do not extract any details from the explicit query statement (e.g., phrases like "which is ...") or from the options. The problem itself should be extracted as the first condition.
Rules: Each constraint or condition in the original question should be treated as a whole; do not split or break down a single constraint or condition into smaller parts. Use the exact descriptions given in the original question without synonym substitution or additional embellishment, ensuring high consistency in both semantics and wording with the original text.
EXAMPLE 1:
question:
There are 7 outstanding students G, H, L, M, U, W and Z in a school. During the summer vacation, the school will send them to the United Kingdom and the United States for inspection. The school has only 7 students participating in this activity, and each person happens to go to one of these two countries. Considering the specialty of each student, this activity must meet the following conditions? (1) If G goes to the UK, then H To the United States. (2) If L goes to the UK, both M and U go to the US. (3) The country W went to was different from the country Z went to. (4) The country where U goes is different from the country where G goes. (5) If Z goes to the UK, then H also goes to the UK.
If G goes to the United States, which of the following must be true?
A. H go to the UK
B. L go to America
C. M go to the UK
D. W go to America
Example Output:
question_parsing:
"The school has only 7 students participating in this activity, and each person happens to go to one of these two countries",
"If G goes to the UK, then H goes to the United States",
"If L goes to the UK, both M and U go to the US",
"The country W went to was different from the country Z went to",
"The country where U goes is different from the country where G goes",
"If Z goes to the UK, then H also goes to the UK",
"G goes to the United States"
] output Example "question_parsing": ["The school has only 7 students participating in this activity, and each person happens to go to one of these two countries", "If G goes to the UK, then H To the United States", "If L goes to the UK, both M and U go to the US", "The country W went to was different from the country Z went to", "The country where U goes is different from the country where G goes", "If Z goes to the UK, then H also goes to the UK", "G goes to the United States"]
NOW ANALYZE THIS NEW QUESTION:
question:
{{question}}
Remember to analyze the specific content above, not the examples. Your output should be a valid JSON object with a "question_parsing" key.

B.3 Prompt For Combined Parser

Given a question and cot. The task is to generate "question_parsing" and "cot_parsing" results based on the content of "question" and "cot". The "question" could be logically divided into four parts: 1) problem description, 2) conditions/constraints, 3) query, and 4) options. The question parsing process involves extracting all conditions necessary for solving the question. The cot parsing process identifies all "statements" and their corresponding "evidence" within the context of the question conditions and the given cot content.
Evaluation: This task consists of two parts: Question parsing and cot parsing. Question parsing

involves extracting all relevant conditions required to solve the problem. The Macro F1 score metric is used to evaluate question parsing performance. The process of extracting statements and evidence is similar to Discourse Parsing. Correct extraction of statements or evidence from the cot is crucial at the outset. Next, the pairwise relationship between a specific statement and its corresponding evidence is assessed (a statement should be followed by its related evidence from the cot). Both semantic and lexical similarity are used to evaluate the accuracy of statements and evidence predictions. The final evaluation metric is the Macro F1 score, applied to both statement parsing and statement-evidence pair extraction. Whether the "statement" can be deduced from the "evidence" logically, answer with only with "true" or "false" Please generate output in JSON format based on the requirements below. The output must include two key-value pairs:
"question_parsing": an array of strings used to extract all necessary constraints provided in the question for solving the problem. Only extract conditions from parts 1) problem description, 2) conditions/constraints, and any additional conditions present in part 3) query. Do not extract any details from the explicit query statement (e.g., phrases like "which is . . . ") or from the options. The problem itself should be extracted as the first condition.
"cot_parsing": an array where each element is an object containing three keys: - "statement": The final inference result from the cot, or an intermediate logical reasoning step. - "evidence": The corresponding supporting evidence directly for the statement as it appears in the cot. - "Verification": A Boolean indicator ("true" or "false") stating whether the "statement" can be deduced from the "evidence" logically.
Rules:
Regarding Question Parsing: Each constraint or condition in the original question should be treated as a whole; do not split or break down a single constraint or condition into smaller parts. Use the exact descriptions given in the original question without synonym substitution or additional embellishment, ensuring high consistency in both semantics and wording with the original text.
Regarding cot Parsing: Extract from the cot text all "statements" and their corresponding "evidences". Each cot entry must be based directly on the text as it appears, ensuring that unnecessary synonym replacements or categorical expansions are avoided. Each "statement" must be immediately followed by its corresponding "evidence". The output should match the original text as closely as possible in both wording and semantics. If the cot text includes explanations about a constraint or a condition being "not applicable" or "irrelevant," these must be captured in full without splitting or merging. Additional note: In the cot text, the selected statements often contain information that is different from or additional to the constraints (especially conditions) extracted from the question. Even if these statements do not match the constraints extracted in "question_parsing," please extract them as they appear and preserve the original wording from the cot text. For each logical reasoning, extract the final deduction as statement and extract intermediate reasoning steps as well as condition as its corresponding evidence. Verify for each "statement" whether it can be logically deduced from the provided "evidence". For this verification, answer only with either "true" or "false".

EXAMPLE 1:
question:
There are 7 outstanding students G, H, L, M, U, W and Z in a school. During the summer vacation, the school will send them to the United Kingdom and the United States for inspection. The school has only 7 students participating in this activity, and each person happens to go to one of these two countries. Considering the specialty of each student, this activity must meet the following conditions? (1) If G goes to the UK, then H To the United States. (2) If L goes to the UK, both M and U go to the US. (3) The country W went to was different from the country Z went to. (4) The country where U goes is different from the country where G goes. (5) If Z goes to the UK, then H also goes to the UK.
If G goes to the United States, which of the following must be true?
A. H go to the UK
B. L go to America
C. M go to the UK
D. W go to America

cot:
Since G goes to the United States, we need to analyze the conditions that follow. Condition (1) is not applicable since G is going to the US. Condition (2) is also not applicable since L's destination is not specified. Condition (3) does not provide any information about H, M, U, or W. Condition (4) states that U's destination is different from G's, which is the US, so U must go to the UK. Condition (5) is not applicable since Z's destination is not specified.

Example Output:
question_parsing:
"The school has only 7 students participating in this activity, and each person happens to go to one

of these two countries",
"If G goes to the UK, then H To the United States",
"If L goes to the UK, both M and U go to the US",
"The country W went to was different from the country Z went to",
"The country where U goes is different from the country where G goes",
"If Z goes to the UK, then H also goes to the UK",
"G goes to the United States"

cot_parsing:
{
"statement": "Condition (1) is not applicable",
"evidence": "G is going to the US",
},
{
"statement": "Condition (2) is also not applicable",
"evidence": "L's destination is not specified",
},
{
"statement": "Condition (3) does not provide any information about H, M, U, or W",
"evidence": "Condition (3)",
},
{
"statement": "U must go to the UK",
"evidence": "Condition (4) states that U's destination is different from G's, which is the US",
},
{
"statement": "Condition (5) is not applicable",
"evidence": "Z's destination is not specified",
}

NOW ANALYZE THIS NEW QUESTION AND COT:
question:
{{question}}
cot:
{{cot}}
Remember to analyze the specific content above, not the examples. Your output should be a valid JSON object with "question_parsing" and "cot_parsing" keys.

B.4 Prompt For Fine-tuning

SYSTEM_PROMPT
You are an expert in logical parsing and reasoning analysis, specializing in analyzing problem conditions and chain-of-thought reasoning processes. Given a question, a cot and preprocessed question_parsing and cot_parsing provided by the given question and the cot. Your task is to generate accurate question question_parsing and cot_parsing results based on the given question and cot.
USER_PROMPT
Based on the following question and chain of thought reasoning process, generate question_parsing and cot_parsing results.
Question:
{question}
Cot:
{cot}
Preprocessed Question Parsing:
{preprocessed_qp}
Preprocessed Cot Parsing:
{preprocessed_cp}
Please provide improved parsing results in the following format: {
"question_parsing": [
"condition 1",
"condition 2",
...
],
"cot_parsing": [
{
"statement": "statement 1",
"evidence": "evidence 1",
"Verification": "true or false"
},
{
"statement": "statement 2",
"evidence": "evidence 2",
"Verification": "true or false"
},
...
]
}
Generate the improved JSON:

B.5 Prompt For LLM Verifier

SYSTEM:
Whether the "statement" can be deduced from the "evidence" logically, answer with only with True or False, do not output other contents.
USER:
question:
question
statement:
statement
evidence:
evidence

B.6 Prompt For Problem Fomulator of Z3 Verifier

Given a question and a cot_parsing. The task is to formulate the problem as a logic program (All the self-defined syntax could be seen in the following examples), consisting three parts: Declarations, Constraints, and Verification. Please strictly follow the samples below to generate the result, do not generate any other irrelevant contents. Declarations: Declare the variables and functions from the question. Constraints: Write the constraints or conditions in the question as logic formulas. Verifications: Write the verification of statement and evidence in the cot_parsing as logic formulas.

IMPORTANT RULES:

1. When using boolean values, always use capitalized True and False, not lowercase true and false. For example, use "is_playing(m) == True" instead of "is_playing(m) == true".
2. Ensure that all variable names used in Constraints and Verifications are declared in the Declarations section.
3. Make sure all names in the Declarations section are consistent with those used in the Constraints and Verifications sections.
4. Do not add any irrelevant comments, such as comments starting with // or #, except # Declarations, # Constraints, # Verifications.
5. Only use logic expressions or syntax patterns that appear in the examples. Do not create your own syntax.

EXAMPLE 1:

question:

There are 7 outstanding students G, H, L, M, U, W and Z in a school. During the summer vacation, the school will send them to the United Kingdom and the United States for inspection. The school has only 7 students participating in this activity, and each person happens to go to one of these two countries.Considering the specialty of each student, this activity must meet the following conditions? (1) If G goes to the UK, then H To the United States.(2) If L goes to the UK, both M and U go to the US.(3) The country W went to was different from the country Z went to.(4) The country where U goes is different from the country where G goes.(5) If Z goes to the UK, then H also goes to the UK.G goes to the United States, which of the following must be true?.H go to the UK.L go to America.M go to the UK.W go to America

cot_parsing:

{
"statement": "Condition (1) is not applicable",
"evidence": "G is going to the US",
"Verification": "true"
},
{
"statement": "Condition (2) is also not applicable",
"evidence": "L's destination is not specified",
"Verification": "true"
},
{
"statement": "Condition (3) does not provide any information about H, M, U, or W",
"evidence": "Condition (3)",
"Verification": "false"
},
{
"statement": "U must go to the UK",
"evidence": "Condition (4) states that U's destination is different from G's, which is the US",
"Verification": "true"
},
{
"statement": "Condition (5) is not applicable",
"evidence": "Z's destination is not specified",
"Verification": "true"
}

Example Output:

Declarations

students = EnumSort([G, H, L, M, U, W, Z])
countries = EnumSort([UK, US])
goes_to = Function([students] -> [countries])

Constraints

Implies(goes_to(G) == UK, goes_to(H) == US) ::: (1) If G goes to the UK, then H To the United States
Implies(goes_to(L) == UK, And(goes_to(M) == US, goes_to(U) == US)) ::: (2) If L goes to the UK, both M and U go to the US
goes_to(W) != goes_to(Z) ::: (3) The country W went to was different from the country Z went to
goes_to(U) != goes_to(G) ::: (4) The country where U goes is different from the country where G goes
Implies(goes_to(Z) == UK, goes_to(H) == UK) ::: (5) If Z goes to the UK, then H also goes to the UK
goes_to(G) == US ::: If G goes to the United States

Verifications

is_deduced(goes_to(G) == US, Not(Implies(goes_to(G) == UK, goes_to(H) == US))) ::: (1) Condition (1) is not applicable
is_deduced(goes_to(G) == US, Not(Implies(goes_to(L) == UK, And(goes_to(M)

== US, goes_to(U) == US)))) ::: (2) Condition (2) is also not applicable
is_deduced(goes_to(W) != goes_to(Z), False) ::: (3) Condition (3) does not provide any information about H, M, U, or W
is_deduced(And(goes_to(U) != goes_to(G), goes_to(G) == US), goes_to(U) == UK) ::: (4) U must go to the UK
is_deduced(goes_to(G) == US, Not(Implies(goes_to(Z) == UK, goes_to(H) == UK))) ::: (5) Condition (5) is not applicable

NOW ANALYZE THIS NEW QUESTION AND COT_PARSING:
question:
{{question}}
cot_parsing:
{{cot_parsing}}
Remember to analyze the specific content above, not the examples. Your output should include Declarations, Constraints, and Verifications sections.