

# ON THE SELF-VERIFICATION LIMITATIONS OF LARGE LANGUAGE MODELS ON REASONING AND PLANNING TASKS

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

There has been considerable divergence of opinion on the reasoning abilities of Large Language Models (LLMs). While the initial optimism that reasoning might emerge automatically with scale has been tempered thanks to a slew of counterexamples—ranging from multiplication to simple planning—there persists a widespread belief that LLMs can self-critique and improve their own solutions in an iterative fashion. This belief seemingly rests on the assumption that verification of correctness should be easier than generation—a rather classical argument from computational complexity—which should be irrelevant to LLMs to the extent that what they are doing is approximate retrieval. In this paper, we set out to systematically investigate the effectiveness of iterative prompting in the context of reasoning and planning. We present a principled empirical study of the performance of GPT-4 in three domains: Game of 24, Graph Coloring, and STRIPS planning. We experiment both with the model critiquing its own answers and with an external correct reasoner verifying proposed solutions. In each case, we analyze whether the content of criticisms actually affects bottom line performance, and whether we can ablate elements of the augmented system without losing performance. We observe significant performance collapse with self-critique and significant performance gains with sound external verification. We also note that merely re-prompting with a sound verifier maintains most of the benefits of more involved setups.

## 1 INTRODUCTION

Large Language Models (LLMs) have rapidly captured the attention of the AI research community with their exceptional natural language completion capabilities. Trained on web-scale language corpora, these models have demonstrated the ability to generate seemingly valuable completions across a wide range of topics. This has led to a surge of interest in determining whether such models are able to perform well on reasoning tasks. Though initial anecdotal results showed promise, further systematic studies revealed inconsistencies and significant issues when applied to reasoning tasks—such as simple arithmetic or logic (Dziri et al., 2023) and planning (Valmeekam et al., 2023b). These results questioned the robustness of their reasoning abilities and led researchers to explore ways to augment and improve these systems.

Of particular interest to us are emerging claims about LLM self-critiquing. In a typical setup, an LLM is iteratively prompted so that it both generates candidate solutions and, in response to separate queries, evaluates them. This process is looped until some stopping condition in hopes of potentially generating a refined answer. Current works (Yao et al., 2023; Shinn et al., 2023; Weng et al., 2023; Chen et al., 2023; Madaan et al., 2023), while admitting that LLMs are not good reasoners on their own, still exhibit considerable optimism about such self-critique systems. This belief seems to rest largely on the assumption that verification of correctness should be easier than generation for many reasoning problems—a rather classical argument from computational complexity. However, we think there are grounds to be skeptical of this assumption. The complexity of the reasoning task should be largely irrelevant to LLM performance, especially if what they are doing is better modeled as approximate retrieval (Kambhampati, 2024).

Intrigued by the prevailing optimism, in this paper we set out to systematically investigate the effectiveness of using LLMs to critique their own generations in the context of planning and reasoning. To gain meaningful insights into the verification/critiquing abilities of LLMs for reasoning tasks, it is crucial to test them on *formal* tasks—ones where machine-verifiable ground truths are available for both generation and criticism. Note that such verification is not feasible in style-based/qualitative tasks like creative writing (Yao et al., 2023) where there is no formal notion of correctness and the critique can vary widely. With this understanding, we select three distinct reasoning problems: *Game of 24*, *Graph Coloring*, and *STRIPS planning*, in which there exist formal notions of correctness that allow us to automatically check the veracity and quality of both (binary) verification and critique generated by the LLM. Furthermore, the algorithmic abilities these domains test are fundamental—any other reasoning task must include components that test these same capabilities, or else be only a retrieval task.

Our methodology employs a system (which we refer to as LLM+LLM) that uses the same LLM (GPT-4-0613 (Achiam et al., 2023)) for iterative solution and verification/critique generation. A generation prompt is sent to the LLM. Its response is used to create a verification prompt, which is then sent back to the same LLM. We use the feedback generated in this way to then create a backprompt, thus restarting the cycle.

Across almost all of our domains, this self-verification system *worsens* performance. We find that as the number of backprompts increases, this kind of self-correction consistently degrades output quality. Our analysis reveals that the verifier LLM’s false negative rate is significant across our domains. In essence, even when the LLM generates a valid solution, the verifier LLM rejects it often enough that overall performance suffers.

We contrast this performance collapse with two baselines. The first is an ablated variant of the system (which we refer to as LLM+Sound Verifier), where an external sound verifier evaluates the LLM’s generations and produces critique. This setup gives substantial performance gains across all domains, but closer analysis shows that the level of feedback often doesn’t seem to matter—as long as the verifier is sound, improvement remains regardless of how much or how little feedback the LLM receives.

We ablate the system further, and remove critique entirely. In this setup, the LLM is repeatedly queried with the exact same base prompt until a sound verifier certifies its solution. Within this impoverished setting, prompts no longer maintain a past history of guesses, yet we can maintain most, if not all, of the gains shown by our previous, more complicated, more expensive setups.

Our empirical results suggest that the benefits of iterative prompting and verification can easily be misattributed to opaque self-critique and seemingly rich feedback. Thus, future implementations of LLMs for reasoning tasks should take the form of LLM-Modulo systems (Kambhampati et al., 2023) where verification is done by external sound systems. In the rest of the paper, we first review related work and discuss domain backgrounds. Then, we explain our methodology, and finally closely analyze LLM self-verification abilities on our domains.

## 2 RELATED WORK

Following the release of GPT-4, anecdotal accounts of performance on reasoning tasks (Bubeck et al., 2023) spurred much research into the capabilities of LLMs on various reasoning domains, from a menagerie of benchmarks covering basic problems (Kojima et al., 2022) to planning (Valmeekam et al., 2023b), logic and arithmetic (Dziri et al., 2023), analogical reasoning (Webb et al., 2023), and even math puzzles (Yao et al., 2023). Though these seemed initially promising, systematic studies began to generate negative results across many of these domains (Valmeekam et al., 2023b; Silver et al., 2022; Abdin et al., 2023; Ullman, 2023; Gendron et al., 2023), claiming that LLM scaling shows much lower returns for reasoning tasks (Rae et al., 2021), and showcasing brittle performance in the face of minor problem permutations (McCoy et al., 2023; Dziri et al., 2023; Arakelyan et al., 2023).

In response, researchers created augmented systems which embed the LLM in larger frameworks in an attempt to improve performance. These take many forms: common search algorithms with the LLM cast in the role of heuristic (Yao et al., 2023; Hao et al., 2023), approaches which reduce error rates by enforcing various consistency conditions (Du et al., 2023; Cohen et al., 2023; Jiang et al.,

2023), and direct LLM self-critique (Shinn et al., 2023; Weng et al., 2023; Chen et al., 2023; Huang et al., 2022; Madaan et al., 2023; Yao et al., 2022).

In the current work, we are interested in examining this third approach: LLM self-critique. In the most basic case, the LLM is queried for an answer, and then is re-queried with its own response together with some instructions to critique or improve it, with this process looped until some stopping condition. This is fundamentally based on the intuition that verification is easier than, or at least different enough from, generation that such a process can improve performance—in analogy to human self-critique (Weng et al., 2023).

The literature abounds with strong, well-cited, and well-referenced claims about the efficacy of these techniques. (Shinn et al., 2023) claims there is an “emergent property of self-reflection in LLMs” and that “self-reflection is extremely useful to learn complex tasks over a handful of trials.” Their experiments claim that every variety they try leads to improvement, and that this is because “self-reflective feedback acts as a ‘semantic’ gradient signal by providing the agent with a concrete direction to improve upon, helping it learn from prior mistakes to perform better on the task.”<sup>1</sup> Other works claim this self-correction does not require “any human feedback” (Chen et al., 2023) and that “even when an LLM cannot generate an optimal output on its first try, the LLM can often provide useful feedback and improve its own output accordingly,” (Madaan et al., 2023) seeming to indicate that these claims generalize beyond the domains, problems, and prompts they were originally made for.

However, some further systematic investigations have found less impressive results in logical fallacy detection (Hong et al., 2023) and HotpotQA (Huang et al., 2023), demonstrating very brittle improvement at best, some of which could be replicated sans self critique by merely including missing domain-general information into the original prompt. The authors of the CRITIC framework (Gou et al., 2023) were the first to notice that, in some cases, LLM self-critique can lead to decreases in performance when compared to sound verification. Contemporaneous to our work,<sup>2</sup> (Huang et al., 2023) investigate two-round self-correction schemes in the GSM8K, CommonSenseQA, and HotpotQA domains. They compare which answers were changed (from correct to incorrect or incorrect to incorrect) and which weren’t, and discuss extensions of their argument to multiagent debate.

Our own work focuses on autonomous multi-round self-verification within three formally verifiable domains that reflect reasoning tasks. We extend previous work by ablating the self-critique system thoroughly to pinpoint the source of performance deterioration, considering more prompting rounds (up to 15), and by examining a new set of domains which we argue are better and more broadly applicable tests of reasoning and self-correction capability.

Reasoning is a fraught term. Previous work has used it to refer, among others, to the human ability to draw conclusions (Leighton & Sternberg, 2004), to the ability to apply common sense to simple scenarios, to positive performance on short-form written tasks, and to formal deductive inference. However, it is often unclear which definition a given set of authors presupposes when making claims about LLM reasoning capabilities. This muddies the discussion and contributes to a strange duality: highly cited papers claim that LLMs are general-purpose reasoners (Kojima et al., 2022; Wei et al., 2022; Bubeck et al., 2023; Zhou et al., 2022); that they have strong, human-like self-reflection capabilities which allow them to correct reasoning mistakes they do make (Shinn et al., 2023; Chen et al., 2023; Madaan et al., 2023); that they can answer difficult, never-before-seen questions via in-context learning as long as they are allowed to use chain of thought to generate intermediate scratch work (Dong et al., 2022); that they can pass or come close on many high school and college-level examinations (Achiam et al., 2023; Gilson et al., 2023; Raimondi et al., 2023; Thaker et al., 2024; Yeadon & Halliday, 2023; de Winter, 2023; Kortemeyer, 2023) and that performance on such standardized exams is evidence about their reasoning capabilities and domain expert knowledge (Wang et al., 2023). Yet, responses to counterexamples and negative results, anecdotally, fall back on a much weaker, seemingly contradictory constellation of premises: LLMs only perform well on things they were trained on, and—in fact—if a model performs poorly, we can only conclude it wasn’t trained on that (but if it performs well, it is generalizing); the average non-expert human would fail on this task if presented it with zero context or training, therefore it’s unsurprising that the LLM

<sup>1</sup>Note that our results ablate away much of this signal (especially the ‘concrete direction’ that exists in explicit critique) to find that most of the improvement in our domains comes from the soundness of the verifier.

<sup>2</sup>Preliminary results from our work were originally presented in two papers at a NeurIPS 2023 workshop.

162 fails; no good prompt engineer would query the LLM in this fashion. (How to tell if a prompt is  
163 good? It follows one of several anthropomorphized design patterns and, most importantly, the result  
164 is positive.)

165 These shifting definitions and implicit assumptions make it very difficult to make concrete claims  
166 and expect to be understood, and they make it even more difficult to pin down claims made by  
167 others or attempt to falsify them. In the current work we address this by restricting our focus to  
168 fully specified, formally verifiable problems which can be solved directly by deductive methods.  
169 Though these may at first seem like a very narrow class, especially when compared to the cornucopia  
170 of commonsense, language-based, and domain-specific benchmarks in the literature, we argue that  
171 they are fundamental, as any other reasoning tasks must include components that test these same  
172 capabilities—otherwise they are merely testing recall. Our work extends studies that have looked  
173 at similar problems, especially those that examined LLM planning capabilities and other classical  
174 reasoning problems (Valmeekam et al., 2023a;b; Stechly et al., 2024; Dziri et al., 2023). However,  
175 no previous work has looked carefully at a broad range of formal verification problems. Filling in  
176 this gap is important, as a lack of benchmark coverage contributes to the illusion that LLMs possess  
177 greater competency than they really do (Saxon, 2024).

178 Furthermore, common domains fall short for evaluating the reasoning and self-critique abilities of  
179 LLMs for additional reasons: test set memorization, lack of problem difficulty, and lack of ground  
180 truth.

181 **Test set memorization:** Due to the black box nature of state of the art models, ensuring that  
182 they weren’t trained on those problems is difficult, and there is compelling evidence that they have  
183 memorized significant chunks of common benchmark sets (Roberts et al., 2023). Many benchmark  
184 sets do not allow for arbitrary generation of novel questions, or worse, draw data from publicly  
185 available sources—the same sources LLM trainers have access to (Yang et al., 2018; Srivastava et al.,  
186 2023). We consider arbitrary generation of new instances of varying difficulty a key desideratum for  
187 any evaluation domain.

188 **Lack of problem difficulty:** Some of the benchmarks (e.g. HotPotQA (Huang et al., 2023),  
189 GSM-8k (Madaan et al., 2023), ) used in evaluations of self-verification are easy—that is, SoTA  
190 LLM performance is already high—and are therefore much less informative about the effects of the  
191 refinement procedure. Additionally, many such sets over-constrain the solution space, usually by  
192 putting the question into multiple choice format. Not only does this make valuable and interesting  
193 critique hard to produce and evaluate, but it trivializes refinement: even a very simple agent can solve  
194 an  $n$ -choice problem with  $n - 1$  critiques—just don’t repeat the same answer. Conclusions drawn over  
195 reduced problem spaces of this type are unlikely to generalize.

196 **Lack of ground truth:** A number of tasks that LLMs are evaluated on (e.g. writing prompts  
197 (Yao et al., 2023), constrained text generation (Lin et al., 2020), toxicity control (Welleck et al.,  
198 2022; Gou et al., 2023) , etc.) are problems without a well-defined ground truth. Instead, they are  
199 evaluated by a couple of indirect methods. Some require an assorted set of metrics which may not  
200 be well-validated for LLMs (e.g. see (Ullman, 2023) for discussion on problems with transferring  
201 results from human-validated tests). Some are scored by humans (Yao et al., 2023). And some are  
202 evaluated by another pre-trained language model (Madaan et al., 2023) or black box AI evaluator  
203 (Welleck et al., 2022). This makes conclusions much harder to draw.

### 204 205 3 BACKGROUND ON TEST TASKS 206

207 We evaluate GPT-4’s self-critique abilities over three distinct tasks, chosen because we believe they  
208 are good proxies for harder reasoning tasks, and because they allow freedom in arbitrary generation  
209 of additional instances while providing easy-to-deploy formal verifiability and guaranteed quality.

210 This gives more than just flexibility—it also decreases the chance that our instances are represented in  
211 the black box model’s opaque training sets. This strengthens our results by reducing the likelihood  
212 that the model can substitute approximate retrieval for general reasoning ability.

213  
214 Any given problem in our sets also has the property that it has a large number of potential solutions,  
215 and this solution space cannot be substantially reduced through simple pattern-matching. As we  
are interested in self-verification loops where the LLM has access to its previous guesses, it is very

important that removing a handful of possible solutions does not trivialize the problem. Compare this to common multiple choice question datasets, where any  $n$ -option problem can be solved in  $n$  exclusive guesses.

### 3.1 GAME OF 24

*Game of 24* is a math puzzle where the goal is to combine four numbers with parentheses and basic arithmetical operations (addition, multiplication, subtraction, and division) to create an expression that evaluates to 24. The numbers are typically constrained to the range 1-12, a nod to game’s playing card roots. Previously, it has been used as a domain of evaluation for other LLM self-verification schemes ((Yao et al., 2023) and fulfills our domain desiderata (see 2). We use it here to enable direct comparisons between previous work and the current paper.

Following (Yao et al., 2023), we use data scraped from `4nums.com`. This list of problems is ordered from shortest to longest average human solution time. Like (Yao et al., 2023), we evaluate our generation tasks on instances 901-1000. However, when evaluating verification and critique alone, we use instances 1-1000.

Verification in this domain is straightforward: given a proposed expression, simplify it using basic arithmetic and check if it is equal to 24. As a sound verifier, we use `SymPy`<sup>3</sup>, a common Python library for symbolic mathematics, and handle any errors that it throws (for instance, if there are unbalanced parentheses) by outputting feedback that says the LLM’s generation was malformed.

### 3.2 GRAPH COLORING

*Graph coloring* is a canonical NP-complete reasoning problem that is related to both propositional satisfiability as well as practical problems like scheduling and allocation. The complexity class NP contains problems that are hard to solve, but easy to verify, so this allows our It is broad enough to give insights into reasoning more generally, yet simple enough that it can be specified and evaluated by a human or basic pattern matching.

In this work, an instance of a graph coloring problem consists of a planar graph together with an optimal coloring number  $n$ . The goal is to output a solution that assigns one of  $n$  colors to each vertex such that no two edge-connected vertices share a color.

Using `GrinPy`<sup>4</sup> to handle common graph operations, we built a test set of 100 graphs of varying small sizes. Each graph was constructed using a variant of the Erdős–Rényi method ( $p = 0.4$ ), with any non-planar or repeat graphs discarded. These were compiled into the standard DIMACS format (DIMACS) together with the graph’s precalculated chromatic number.

Verifying that a proposed coloring is correct is also easy: just check the colors of every edge. If any of them has two vertices of the same color, reject the coloring. Our sound verifier is a simple, single `for`-loop implementation of this idea in Python: for each edge in the graph description, we check that both of its vertices are different.

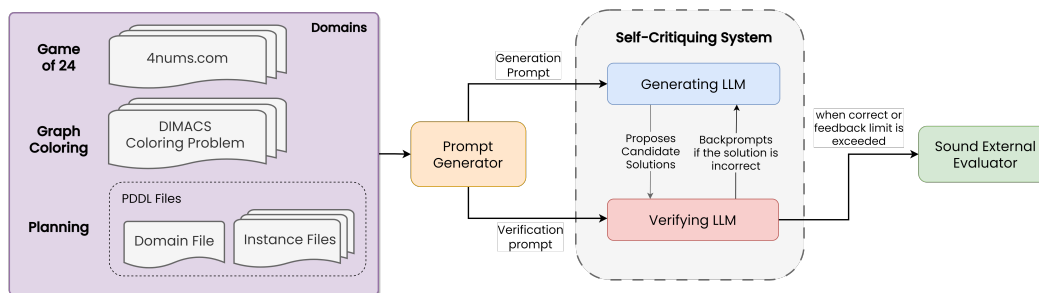


Figure 1: Overall Backprompting and Evaluation Architecture

<sup>3</sup><https://www.sympy.org/en/index.html>

<sup>4</sup><https://pypi.org/project/grinpy/>

### 3.3 STRIPS PLANNING

STRIPS planning is a formalism used in automated planning that operates in a discrete, deterministic space. Such problems, commonly known as classical planning problems, involve finding a sequence of actions that when executed in a certain world state will take the agent to a desired goal state. STRIPS planning has a long history of featuring in various planning challenges and competitions, and is formally represented using the PDDL (Planning Domain and Definition Language) framework (McDermott et al., 1998). We consider two domains in STRIPS planning. One is *Blocksworld*, a simple common-sense domain used in International Planning Competitions (IPC, 1998) and *Mystery Blocksworld*, which is an obfuscated version of Blocksworld. For both the domains, we draw instances from (Valmeekam et al., 2023a) for our evaluations.

A PDDL specification consists of three components. The *domain* doesn't change between problems and consists of a set of predicates, which can be used to describe the state of the world, and a set of actions—stored with their preconditions and effects—that the agent is allowed to take. The *initial state* is a list of predicates that are true at the outset of the specific problem (an example predicate, in natural language: "the red block is on the table"). The *goal* is a boolean expression of predicates.

Solutions to PDDL problems take the form of correct plans—sequences of actions which can be executed from the initial state without violating any of their preconditions and which arrive at a final state that satisfies the goal. Verifying proposed plans is a matter of following the actions in order and checking that these two desiderata are achieved. For our experiments, we use VAL (Howey et al., 2004) as the sound external critique that evaluates and critiques LLM generated plans.

## 4 METHODOLOGY

As our results are about ablations of self-critique architectures, our basic test framework is a general prompting architecture informed by interchangeable domain-specific components. Our intent is to give the LLM as much information as possible, so we design prompts that include the entire history of previously proposed solutions and the feedback they received.

A problem instance is a domain-specific formal description. In attempting a problem, our system (as shown in Figure 1) proceeds as follows: (1) The instance is processed by a simple, hard-coded natural language translator into a prompt which is then sent to the LLM. (2) The LLM's response is wrapped in a domain-specific critique prompt, which is separately sent as another LLM query. (3) If the following response claims that the proposed solution is correct, we stop the system and output the proposed solution. If it doesn't, the critique is extracted, wrapped in instruction text, and appended to a prompt containing the entire history of interactions so far. This is then sent to the LLM, and the cycle repeats, potentially until we enforce a timeout.

Though only two types of prompts are sent, the LLM can be seen as playing three separate roles: as an answer guesser, a (binary) verifier, and a critique generator. In order to better understand which of these roles contribute to increased performance, we will examine variations of this system where one or more of them are changed or removed.

To examine LLM verification abilities, we first measure the performance of the entire system, and then evaluate false positive and false negative rates across domains. To better understand the guesser role, and the LLM's ability to consider and implement critique, we will modify the loop so that the verification and critique roles are played by a provably sound verifier that provides rich, correct feedback. We will then reduce and eventually eliminate the amount of provided information (e.g. rich feedback: explicitly giving an evaluation of a proposed Game of 24 expression; minimal feedback: "the previous answer was wrong"; no feedback: re-querying with the base prompt), while keeping track of changes in the performance of the entire system.

For LLM critique generation, we construct subdomains of our original domains. In these prompts, we provide a problem description and a proposed solution, and we ask the LLM to provide domain-specific critique of the solution if it is incorrect. We parse the output using a hard-coded script and measure accuracy compared to the sound verifier's output.

Note that sound verifiers output task specific critiques: for Game of 24, the evaluation of the provided expression ("1+1+4+6=12 not 24"); constraint violations for graph coloring ("vertices 1 and 3 were

324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377

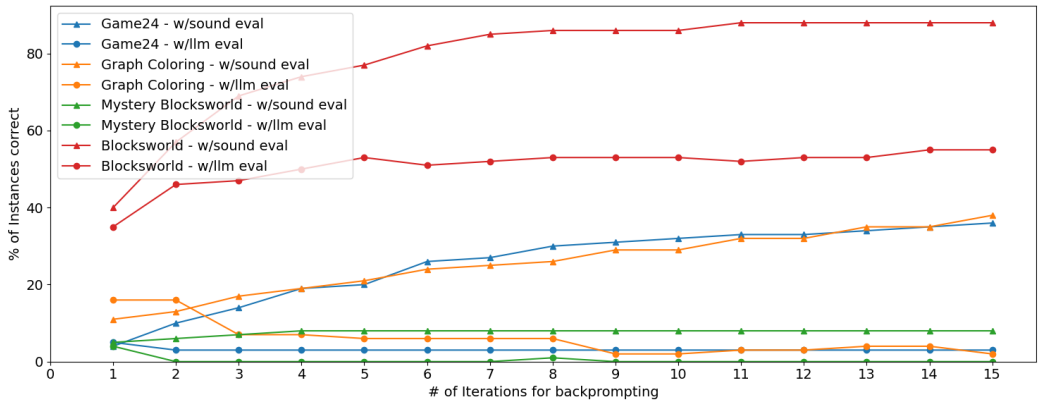


Figure 2: **Performance vs Number of Iterations Before Timeout.** We measure performance at iteration  $n$  by evaluating how many instances are currently correctly solved (whether the LLM has verified them or not). In other words, we evaluate as if the timeout were  $n$ ) and adding that to the number the LLM has correctly verified so far. Note that if the verifier incorrectly rejects an answer and the followup is wrong, the next round may be worse. When paired with a sound verifier, the LLM monotonically improves its performance as the number of backprompts increase up to some asymptote. The top three lines show this for each of our domains. On the other hand, when the LLM itself is used as a verifier, performance collapses immediately.

both colored red despite sharing an edge"); and precondition violations (the second action “succumb object a” is invalid because the succumb action requires the pain object to be true, which is not the case after the first action.) and failure to reach goal (“this plan does not reach the goal”) for planning.

### 5 EXAMINING SELF-VERIFICATION

| Domain              | S.P. | LLM+LLM | LLM+Sound Critique |       |       | Sampling |      | S.C. |
|---------------------|------|---------|--------------------|-------|-------|----------|------|------|
|                     |      |         | B.F.               | F.E.F | A.E.F | k=15     | k=25 |      |
| Game of 24          | 5%   | 3%      | 36%                | 38%   | N/A   | 28%      | 42%  | 6%   |
| Graph Coloring      | 16%  | 2%      | 38%                | 37%   | 34%   | 40%      | 44%  | 14%  |
| Blocksworld         | 40%  | 55%     | 60%                | 87%   | 83%   | 68%      | 72%  | 42%  |
| Mystery Blocksworld | 4%   | 0%      | 10%                | 8%    | 6%    | 9%       | 14%  | 4%   |

Table 1: **Accuracy across prompting schemes over 100 instances per domain.** S.P.-Standard Prompting. B.F.-Binary Feedback. F.E.F-First Error Feedback, e.g. the first wrong edge, the first mistaken action, or the non-24 evaluation of the proposed expression. A.E.F-All Error Feedback, e.g. every wrong edge, every mistaken action and error. Note that there is no third critique type for Game of 24 due to the simplicity of the domain. We include two examples of sampling, one at 15 samples, the other at 25, to show that completely ablating critique retains the performance increases of critique. We also include S.C.-Self Consistency results, where the most common answer in a pool of 15 is the one that is output by the model, as another comparison point.

We evaluate our system over 100 instances in each domain. In standard prompting we send a single query to the LLM and treat whatever it outputs as its final answer. We use this as our baseline. As shown in Table 1, when we augment this condition with the full self-critique setup, performance *decreases*. In fact, Figure 2 shows that as the number of backprompts increases, this kind of self-correction consistently degrades output quality. If the LLM were a good verifier, then we would have

378 expected it to recognize instances which are already right, and thus—at worst—maintain the baseline  
379 score.

380 The LLM-as-verifier ranges in accuracy depending on the domain, as illustrated in Table 2. Notably,  
381 Game of 24 and Blocksworld maintain lower rates of both false positives and false negatives, and this  
382 is reflected in LLM+LLM performance on those domains, which doesn’t fail as drastically as it does  
383 in the other cases. In Blocksworld, we even see a modest improvement, though that improvement is  
384 still significantly worse than having a sound verifier. In the remaining two domains, the false negative  
385 rates are very high. In effect, the system rejects most answers and then times out on a set of later,  
386 worse generations.

387 When we replace the LLM verifier with a sound verifier, every correct answer will be accepted  
388 properly. Intuitively, it can do no worse than standard prompting—anything that was generated  
389 correctly initially must be accepted. As shown in Table 1, performance is much higher in all sound  
390 critique cases, though it still falls short of 100%. Due to the setup, this can’t be due to the verifier, but  
391 must be the fault of the answer generating LLM. After 15 rounds, any instance that has yet to be  
392 answered correctly will time out, and this process is the *only* source of inaccuracy arising from the  
393 LLM-sound verifier loop.

394 In general, it is clear that the verifier requires high accuracy or else the overall system will encounter  
395 compounding errors. In the reasoning domains considered, LLMs-as-verifiers are mostly insufficient  
396 to prevent these.<sup>5</sup>

| Domain              | Accuracy          | F.P.R            | F.N.R            |
|---------------------|-------------------|------------------|------------------|
| Game of 24          | 87.0% (3567/4100) | 10.4% (320/3071) | 20.7% (213/1029) |
| Graph Coloring      | 72.4% (362/500)   | 6.5% (25/382)    | 95.8% (113/118)  |
| Blocksworld         | 71.8% (359/500)   | 18.55% (64/345)  | 15.48% (24/155)  |
| Mystery Blocksworld | 79.6% (398/500)   | 0.5% (2/397)     | 97.09% (100/103) |

398 Table 2: **LLM Verification results.** F.P.R. - False Positive Rate, F.N.R - False Negative Rate.

### 409 5.1 CRITIQUE GENERATION

410 We consider the quality of LLM-generated free-form critiques separately from that of their binary  
411 verification signals, and find that they are full of unhelpful hallucinations and mistakes. To obtain  
412 the following results, we ran a further suite of experiments with specially crafted proposed solutions  
413 wrapped in verification prompts. The exact breakdown of which types of solutions were generated is  
414 available in each domain’s appendix.

415 In Game of 24, without any further instructions, the LLM tends to output incorrect suggestions for  
416 the answer. When prompted to give an evaluation of the proposed expression first, its accuracy varies.  
417 In fact, when we restrict ourselves to only looking at verification of equations that are guaranteed to  
418 equal 24, and therefore must be correct, it labels 79.1% of them as correct, but evaluates 81.6% of  
419 them to 24. That is, there are problems which it evaluates correctly but which it still marks as wrong.

420 In Graph Coloring, the LLM’s critiques of proposed solutions are riddled with non-existent edges  
421 and include many spurious claims about the colors of vertices, often missing the violated constraint  
422 in favor of them. A breakdown and detailed examples are provided in appendix A.4.1.

423 In the planning domains, the critiquing LLM often hallucinates whether action preconditions are met  
424 or not. In Mystery Blocksworld, the LLM incorrectly assumes the state of these preconditions as  
425 well. This leads to lower accuracy of the critiques provided by the LLM. A further breakdown is in  
426 appendix A.5.1

427 In other words, the LLM introduces errors in two places: verification, where it can pass over correct  
428 answers and accept wrong ones; and critique generation, where it can produce misleading feedback  
429 and bias future outputs away from the correct answer. When they compound sufficiently, these errors  
430

431 <sup>5</sup>Prompt and response examples can be found in the Appendix.



432 actually reduce the performance of the LLM-based self-critique loop below that of just taking the  
433 LLM’s very first guess.  
434

## 435 5.2 CRITIQUE CONSIDERATION 436

437 Our results also imply that the LLM often isn’t sensitive to varying levels of feedback. We use a  
438 sound verifier to critique the LLM’s output, and compare the results over three levels of feedback for  
439 graph coloring and planning, and over two levels for game of 24. Examples of prompts containing  
440 each sort of feedback can be found in appendices A.3.3, A.4.3, and A.5.7.  
441

- 442 • **Binary feedback** is the same for all domains: either the verifier accepts the solution,  
443 stopping the system, or we create a backprompt which says the previous answer was wrong  
444 but doesn’t elaborate.
- 445 • **First error feedback** mentions the first error that was found (e.g. an incorrect edge in graph  
446 coloring, an inexecutable step in planning, the evaluation of the proposed expression in  
447 game of 24).
- 448 • **All errors feedback** includes every error that was found. Note that due to the simplicity of  
449 game of 24, we do not implement a third feedback level for it.  
450

451 Perhaps surprisingly, Table 1 shows very little difference between these three conditions. And in two  
452 of our domains, increasing the amount of feedback actually leads to a decrease in performance.  
453

454 The intuition underlying the entire critique system is that sending a history of previous failed attempts  
455 together with information about why those attempts were failures should guide the LLM toward  
456 better future answers. If this were true, then we would expect the performance jump to be tied to  
457 the content of the backprompts. With only the data discussed so far, it might seem like the relevant  
458 content is actually the history of failed attempts the LLM receives rather than any feedback on those  
459 attempts. However, our final experiments contradict this interpretation as well.

460 We take our ablation of critique consideration to the logical extreme, and remove the availability  
461 of critique entirely. In this sampling setup, we keep the verifier but don’t change the prompt at all  
462 between iterations. The LLM (at  $t = 1$ ) is asked the same question over and over until the verifier  
463 certifies it or it hits some pre-established timeout.

464 Represented in Table 1 by the “Sampling” columns, this gives comparable gains to feedback condi-  
465 tions. Note that, because prompts do not grow additively with iteration number, the token cost of  
466 these prompts is quadratically lower. This allows us to increase performance further by just increasing  
467  $k$  further. As a sanity check, we compare this to a self-consistency baseline (Wang et al., 2022),  
468 where we instead select the most common answer from the 15 generated ones. This baseline is listed  
469 under “S.C.” and shows no improvement over standard prompting.

470 Our final results show that, in our domains, the information in critiques does not have as much of an  
471 effect on performance as previous literature claimed it should. In fact, our performance increases  
472 seem to stem in large part just from having enough guesses and a sound verifier. We therefore see the  
473 LLM primarily as an idea generator.  
474

## 475 6 CONCLUSION 476

477 In this paper, we conducted a systematic evaluation of the self-critique abilities of Large Language  
478 Models on three reasoning and planning tasks. We separated self-critique into three components:  
479 verification, critique generation, and critique consideration. Across the hard reasoning domains  
480 we evaluated, LLMs did poorly in all three roles, with the stacked errors often making the LLM  
481 self-critiquing loop perform worse than just having the LLM guess the solution up front. These  
482 failures of verification could potentially be very detrimental to a system’s dependability, especially in  
483 domains where high reliability is paramount. In contrast, we saw performance gains when an external  
484 sound verifier provides the verification signal and critique. We also found that good performance can  
485 be achieved without any critique whatsoever: just let the LLM make many guesses, and have a sound  
verifier pick any that is actually correct.

Our results contradict earlier work that has been very optimistic about LLM self-critique abilities. They also add depth to contemporaneous studies that focused on benchmarks that were too easy for LLMs to begin with, lacked clear ground truth, and didn't account for test set memorization.

Our proposal, based on the case studies we've performed in this paper is, when possible, to embed LLMs in systems which allow them to guess at solutions multiple times, but which provide some kind of signal for when a guess is good enough. Ideally, this takes the form of a sound verifier, like VAL (Howey et al., 2004) for STRIPS planning, basic expression simplification for Game of 24, or a constraint checker for constraint satisfaction problems. In real-world applications we expect this role to be played by a menagerie of partial critics evaluating plans or solutions based on criteria that they have access to, designed so that consensus is considered verification. Similar architectures have already shown some success (Romera-Paredes et al., 2023), and previous work has proposed the general LLM-Modulo framework (Kambhampati et al., 2024) which the current work fits into.

## REFERENCES

- Marah I Abdin, Suriya Gunasekar, Varun Chandrasekaran, Jerry Li, Mert Yuksekgonul, Rahee Ghosh Peshawaria, Ranjita Naik, and Besmira Nushi. *Kitab: Evaluating llms on constraint satisfaction for information retrieval*. *arXiv preprint arXiv:2310.15511*, 2023.
- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. *Gpt-4 technical report*. *arXiv preprint arXiv:2303.08774*, 2023.
- Shushan Arakelyan, Rocktim Jyoti Das, Yi Mao, and Xiang Ren. Exploring distributional shifts in large language models for code analysis. *arXiv preprint arXiv:2303.09128*, 2023.
- Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.
- Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. Lm vs lm: Detecting factual errors via cross examination. *arXiv preprint arXiv:2305.13281*, 2023.
- Joost CF de Winter. Can chatgpt pass high school exams on english language comprehension? *International Journal of Artificial Intelligence in Education*, pp. 1–16, 2023.
- DIMACS. DIMACS Implementation Challenges. URL <http://archive.dimacs.rutgers.edu/Challenges/>. Archive available at <http://archive.dimacs.rutgers.edu/Challenges/>.
- Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.
- Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.
- Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=Fkckkr3ya8>.
- Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. Large language models are not abstract reasoners. *arXiv preprint arXiv:2305.19555*, 2023.
- Aidan Gilson, Conrad W Safranek, Thomas Huang, Vimig Socrates, Ling Chi, Richard Andrew Taylor, David Chartash, et al. How does chatgpt perform on the united states medical licensing examination (usmle)? the implications of large language models for medical education and knowledge assessment. *JMIR medical education*, 9(1):e45312, 2023.

- 540 Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen.  
541 Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint*  
542 *arXiv:2305.11738*, 2023.
- 543 Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning  
544 with Language Model is Planning with World Model. In Houda Bouamor, Juan Pino, and Kalika  
545 Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language*  
546 *Processing*, pp. 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL [https://aclanthology.org/2023.](https://aclanthology.org/2023.emnlp-main.507)  
547 [emnlp-main.507](https://aclanthology.org/2023.emnlp-main.507).
- 548 Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. A closer look  
549 at the self-verification abilities of large language models in logical reasoning. *arXiv preprint*  
550 *arXiv:2311.07954*, 2023.
- 551 Richard Howey, Derek Long, and Maria Fox. VAL: Automatic plan validation, continuous effects  
552 and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with*  
553 *Artificial Intelligence*, pp. 294–301. IEEE, 2004.
- 554 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,  
555 and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint*  
556 *arXiv:2310.01798*, 2023.
- 557 Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan  
558 Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through  
559 planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.
- 560 IPC. International planning competition, 1998. URL [https://www.icaps-conference.](https://www.icaps-conference.org/competitions/)  
561 [org/competitions/](https://www.icaps-conference.org/competitions/).
- 562 Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T Kwok.  
563 Backward reasoning in large language models for verification. *arXiv preprint arXiv:2308.07758*,  
564 2023.
- 565 Subbarao Kambhampati. Can large language models reason and plan? *Annals of the New York*  
566 *Academy of Sciences*, 2024.
- 567 Subbarao Kambhampati, Karthik Valmeekam, Matthew Marquez, and Lin Guan. On the role of  
568 large language models in planning. tutorial presented at the international conference on automated  
569 planning and scheduling (icaps), prague, July 2023.
- 570 Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Kaya Stechly, Mudit Verma, Siddhant  
571 Bhambri, Lucas Saldyt, and Anil Murthy. Llms can’t plan, but can help planning in llm-modulo  
572 frameworks. *arXiv preprint arXiv:2402.01817*, 2024.
- 573 Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large  
574 language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:  
575 22199–22213, 2022.
- 576 Gerd Kortemeyer. Could an artificial-intelligence agent pass an introductory physics course? *Physical*  
577 *Review Physics Education Research*, 19(1):010132, 2023.
- 578 Jacqueline P Leighton and Robert J Sternberg. *The nature of reasoning*. Cambridge University Press,  
579 2004.
- 580 Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and  
581 Xiang Ren. CommonGen: A constrained text generation challenge for generative commonsense  
582 reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp.  
583 1823–1840, 2020.
- 584 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri  
585 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement  
586 with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

- 594 R Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L Griffiths. Embers  
595 of autoregression: Understanding large language models through the problem they are trained to  
596 solve. *arXiv preprint arXiv:2309.13638*, 2023.
- 597
- 598 Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M.  
599 Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language.  
600 1998.
- 601
- 602 Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John  
603 Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models:  
604 Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.
- 605
- 606 Raffaele Raimondi, Nikolaos Tzoumas, Thomas Salisbury, Sandro Di Simplicio, and Mario R  
607 Romano. Comparative analysis of large language models in the royal college of ophthalmologists  
608 fellowship exams. *Eye*, 37(17):3530–3533, 2023.
- 609
- 610 Manley Roberts, Himanshu Thakur, Christine Herlihy, Colin White, and Samuel Dooley. Data  
611 contamination through the lens of time. *arXiv preprint arXiv:2310.10628*, 2023.
- 612
- 613 Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Matej Balog,  
614 M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang,  
615 Omar Fawzi, et al. Mathematical discoveries from program search with large language models.  
616 *Nature*, pp. 1–3, 2023.
- 617
- 618 Michael Saxon. Benchmarks as Microscopes: A Call for Model Metrology, 2024.
- 619
- 620 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion:  
621 Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural  
622 Information Processing Systems*, 2023.
- 623
- 624 Tom Silver, Varun Hariprasad, Reece S Shuttleworth, Nishanth Kumar, Tomás Lozano-Pérez, and  
625 Leslie Pack Kaelbling. Pddl planning with pretrained large language models. In *NeurIPS 2022  
626 foundation models for decision making workshop*, 2022.
- 627
- 628 Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam  
629 Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the  
630 imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions  
631 on Machine Learning Research*, 2023.
- 632
- 633 Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. Chain of thoughtlessness: An  
634 analysis of cot in planning. *arXiv preprint arXiv:2405.04776*, 2024.
- 635
- 636 Nikhil G Thaker, Navid Redjal, Arturo Loaiza-Bonilla, David Penberthy, Tim Showalter, Ajay  
637 Choudhri, Shirnett Williamson, Gautam Thaker, Chirag Shah, Matthew C Ward, et al. Large  
638 language models encode radiation oncology domain knowledge: Performance on the american  
639 college of radiology standardized examination. *AI in Precision Oncology*, 1(1):43–50, 2024.
- 640
- 641 Tomer Ullman. Large language models fail on trivial alterations to theory-of-mind tasks. *arXiv  
642 preprint arXiv:2302.08399*, 2023.
- 643
- 644 Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambham-  
645 pati. Planbench: An extensible benchmark for evaluating large language models on planning and  
646 reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems  
647 Datasets and Benchmarks Track*, 2023a. URL <https://openreview.net/forum?id=YXog14uQUO>.
- 648
- 649 Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the  
650 planning abilities of large language models - a critical investigation. In *Thirty-seventh Confer-  
651 ence on Neural Information Processing Systems*, 2023b. URL [https://openreview.net/  
652 forum?id=X6dEqXISEW](https://openreview.net/forum?id=X6dEqXISEW).

- 648 Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R  
649 Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level  
650 scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*,  
651 2023.
- 652 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdh-  
653 ery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models.  
654 *arXiv preprint arXiv:2203.11171*, 2022.
- 655 Taylor Webb, Keith J Holyoak, and Hongjing Lu. Emergent analogical reasoning in large language  
656 models. *Nature Human Behaviour*, 7(9):1526–1541, 2023.
- 657  
658 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny  
659 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in*  
660 *neural information processing systems*, 35:24824–24837, 2022.
- 661  
662 Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin  
663 Choi. Generating sequences by learning to self-correct. In *The Eleventh International Conference*  
664 *on Learning Representations*, 2022.
- 665  
666 Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and  
667 Jun Zhao. Large language models are better reasoners with self-verification. In *Findings of the*  
668 *Association for Computational Linguistics: EMNLP 2023*, pp. 2550–2575, 2023.
- 669  
670 Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov,  
671 and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question  
672 answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language*  
*Processing*, pp. 2369–2380, 2018.
- 673  
674 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.  
675 React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*,  
676 2022.
- 677  
678 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R  
679 Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In  
680 *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=5Xc1ecx01h>.
- 681  
682 Will Yeadon and Douglas P Halliday. Exploring durham university physics exams with large language  
683 models. *arXiv preprint arXiv:2306.15609*, 2023.
- 684  
685 Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans,  
686 Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning  
687 in large language models. *arXiv preprint arXiv:2205.10625*, 2022.
- 688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701

## A APPENDIX

### A.1 PROMPT VARIATION AND CHAIN OF THOUGHT

LLM results are well-known to be brittle to choice and phrasing of prompt. We ran experiments across multiple prompts to ensure the robustness of our results. Table A1 shows the results for the full pipeline where verification prompts are modified to ask for CoT reasoning first. Results from the main paper are provided for easy comparison. Prompts can be found in each domain’s prompt section in this appendix, under the header "Prompt to Elicit CoT Verification".

Performance does improve in some, though not all, cases. However, seemingly near-perfect improvements in verification ability do not translate into near-sound-verifier performance. In G24, CoT increases verification accuracy from 87% to 99%, and does shrink the difference between sound verifier and LLM-verifier in the full pipeline, but a 6 percentage point gap still remains! Furthermore, these improvements come with a large cost increase: in G24, this leads to a 17 times increase in necessary output tokens, which more than doubles the cost of verification.

Note that Chain of Thought techniques themselves vary greatly in their effectiveness across domains. In particular, in reasoning domains like Blocksworld, previous work has shown that they fail to generalize and are not particularly robust (Stechly et al., 2024).

On the rest of the tasks, where verification is also fairly simple and linear, and thus theoretically amenable to CoT, we do not see nearly as significant improvements.

| Domain      | S.P. | Sampling | LLM+LLM | LLM+LLM-C | F.E.F. |
|-------------|------|----------|---------|-----------|--------|
| Game of 24  | 5%   | 28%      | 3%      | 32%       | 38%    |
| Coloring    | 16%  | 40%      | 2%      | 0%        | 37%    |
| Blocksworld | 40%  | 68%      | 55%     | 53%       | 87%    |
| Mystery     | 4%   | 9%       | 0%      | 5%        | 6%     |

Table A1: **Accuracy across prompting schemes including CoT verification schemes over 100 instances per domain.** S.P.–Standard Prompting. Sampling–k=15. LLM+LLM–Main paper result. LLM+LLM-C–Full pipeline with chain of thought verification. F.E.F.–(Sound) First Error Feedback.

We also reran our verification-only experiments with these new prompts, as well as with variations on the original (non-CoT) prompts. Table A2 has these results, presented alongside the original ones.

### A.2 ON TREE OF THOUGHTS

Our results on the Game of 24 setting seem to contradict the results shown in (Yao et al., 2023). However, this is mainly because the self-verification setting presented in the main text of this paper is not directly comparable to that of (Yao et al., 2023). We ran an additional analysis to provide a direct comparison.

Our external verifier results are all done with only 15 queries to the LLM. (Yao et al., 2023) isn’t entirely clear on the number of queries used, but table 7 in the appendix does give a cost breakdown. Per problem, 100 CoT prompts costs \$0.47, but running Tree of Thoughts (ToT) averages \$0.74–cost-equivalent to about 150 CoT prompts. On the exact same test set, we extend our experiments to 150 (direct, non-CoT) queries with a sound verifier, and we reach 70%, comparable to ToT’s reported 74%.

The remaining difference is likely due to the fact that ToT implements a classical breadth-first search algorithm, only prompting the LLM to generate (much easier) intermediate steps and heuristic evaluations rather than full solutions and reflections. By reducing compositionality and offloading it to a proven classical algorithm, ToT sidesteps some of the major hurdles to LLM reasoning. Our results highlight why such techniques do not scale beyond the simplest toy instances.

| Domain             | Accuracy          | F.P.R            | F.N.R            |
|--------------------|-------------------|------------------|------------------|
| Game of 24         | 87.0% (3567/4100) | 10.4% (320/3071) | 20.7% (213/1029) |
| Game of 24-CoT     | 98.8% (4051/4100) | 0.2% (6/3071)    | 4.3% (44/1029)   |
| Graph Coloring     | 72.4% (362/500)   | 6.5% (25/382)    | 95.8% (113/118)  |
| Graph Coloring-CoT | 77.6% (388/500)   | 10.7% (41/382)   | 60.2% (71/118)   |
| Blocksworld        | 71.8% (359/500)   | 18.55% (64/345)  | 15.48% (24/155)  |
| Blocksworld-S      | 71.2% (356/500)   | 22.1% (76/345)   | 8.4% (13/155)    |
| Blocksworld-CoT    | 77.6% (388/500)   | 7.6% (26/345)    | 23.9% (37/155)   |
| Mystery            | 79.6% (398/500)   | 0.5% (2/397)     | 97.09% (100/103) |
| Mystery-S          | 79.0% (395/500)   | 1.26% (5/397)    | 96.1% (99/103)   |
| Mystery-CoT        | 81.8% (409/500)   | 3.2% (13/397)    | 72.8% (75/103)   |

Table A2: **LLM Verification results across prompts.** F.P.R. - False Positive Rate, F.N.R - False Negative Rate. The -S cases are non-CoT prompts with the answer and reasoning swapped for those domains where answer was originally asked for first. The -CoT cases are those in which verification is done with CoT.

|                 | Correct Evaluation | Correct Verification |
|-----------------|--------------------|----------------------|
| correct         | 81.6%              | 79.1%                |
| correct-no-info | 84.4%              | -                    |
| ablated_op      | 47.5%              | 92.1%                |
| ablated_number  | 52.2%              | 82.9%                |
| random          | 48.8%              | 95.5%                |
| random-no-info  | 60.3%              | -                    |
| LLM             | 55%                | 71%                  |

Table A3: GPT-4's evaluation vs. verification on Game of 24 across expression types.

### A.3 GAME OF 24

#### A.3.1 EVALUATION VS. BINARY VERIFICATION FOR GAME OF 24

The following is a more in-depth comparison of GPT-4's critique and verification abilities on game of 24.

For each instance, we generated five different kinds of proposed expressions: correct, ablated operation (exactly one operation is wrong), ablated number (exactly one number is wrong), random, and LLM (sampled from LLM generations). For each of these proposed expression, we sent a query to the LLM asking it to first evaluate the expression and then to say if it is correct, that is equals 24. We also generated two more "no info" cases: correct and random. These two are the exact same as the previous, but only ask for the evaluation of an expression without mentioning the associated goal state (=24) or asking for verification.

Table A.3.1 summarizes the results. Note that we generated 1000 expressions for each type, one from every problem in the full set, but only 100 for the LLM case, as our generations were constrained in the main paper to instances 901-1000.

810 A.3.2 PROMPTS  
811812 All of following examples are built on the same Game of 24 problem, except for the LLM Self-Critique  
813 examples.814  
815 **Raw text format of Game of 24 instance**816 1 1 4 6  
817818  
819 **Baseline, Direct Prompt**  
820821 Use numbers and basic arithmetic operations (+ - \* /) to obtain  
822 24. You must write your response. Write your answer first,  
823 followed by [ANSWER END]  
824 Input: 1 1 4 6  
825 Answer:  
826827 **Example LLM Response**  
828829  $(6 / (1 / 4)) = 24$   
830831  
832 **Prompt To Elicit Verification**  
833834 Please check if the following expression uses only the given  
835 numbers (and no others) and evaluates to 24:  $((9+10)-(4-9))$   
836  
837 Respond only in JSON format as described below:  
838 {  
839 "evaluation": "number the expression evaluated to",  
840 "correct": boolean}  
841 Ensure that Python's json.loads can parse this. Do not provide  
842 anything else in your response."843 **Prompt To Elicit CoT Verification**  
844845 Using each of the numbers 1 7 9 11 exactly as many times as they  
846 appear in the list and the basic arithmetic operations (+ - \*  
847 /), it is possible to write an expression that evaluates to 24.  
848 Please check if the following expression uses only the correct  
849 numbers (and no others) and evaluates to 24:  $(11 - 7) * (9 - 1)$   
850 = 24  
851 If it is not correct, please give feedback on what is wrong and  
852 how to correct it.  
853 First, think step by step. Check that the expression uses only the  
854 correct numbers, has exactly the right number of instances each  
855 number, and evaluates to 24. Then decide what your final answer  
856 is.  
857 When outputting your final answer, first print the [Answer] tag,  
858 then put your final answer after the [Answer] tag and respond  
859 only in JSON format as described below:  
860 {  
861 "feedback": "feedback",  
862 "correct": boolean}  
863 Ensure that Python's json.loads can parse this.  
  
Let's think step by step.  
[Thoughts]



864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917

### A.3.3 BACKPROMPTS

#### Backprompt Generated From Self-Critique

Feedback: This is not correct.  
The expression is incorrect. The correct expression should be:  $(11 - 2) * 5 + 8 = 24$

With this feedback, please try again. Using the numbers 2 5 8 11 please provide a correct expression that evaluates to 24. Write your answer first. At the end of your answer, write [ANSWER END]  
Answer: ",

#### Backprompt Generated From Sound Verifier (Pass/Fail)

Feedback: This is not correct. Using the numbers 1 1 4 6 please provide a correct expression that evaluates to 24. Write your answer first. At the end of your answer, write [ANSWER END]  
Answer:

#### Backprompt Generated From Sound Verifier

Feedback: This is not correct. This expression consists of the numbers 6, 1, 4, but it has to consist of only and exactly ['1', '1', '4', '6']. Using the numbers 1 1 4 6 please provide a correct expression that evaluates to 24. Write your answer first. At the end of your answer, write [ANSWER END]  
Answer: ",

#### Backprompt Generated From Sound Verifier Where the Answer Was Malformed (Missing a Parenthesis)

Use numbers and basic arithmetic operations (+ - \* /) to obtain 24. Write your answer first. At the end of your answer, write [ANSWER END]  
Input: 1 1 4 6  
Answer:  
 $((6 / (1 - 1/4)) = 24$ [ANSWER END]  
Feedback: This is not correct. This expression is malformed. Using the numbers 1 1 4 6 please provide a correct expression that evaluates to 24. Write your answer first. At the end of your answer, write [ANSWER END]  
Answer:

### A.3.4 COMPLETE EXAMPLE OF ITERATIVE PROMPT SEQUENCE

#### Full Sequence of 7 (Sound Verifier Generated) Backprompts Stopping at Successful Completion of the Task

Use numbers and basic arithmetic operations (+ - \* /) to obtain 24. Write your answer first. At the end of your answer, write [ANSWER END]  
Input: 1 1 4 6  
Answer:  
 $((6 / (1 - 1/4)) = 24$ [ANSWER END]  
Feedback: This is not correct. This expression is malformed. Using the numbers 1 1 4 6 please provide a correct expression that

918  
 919 evaluates to 24. Write your answer first. At the end of your  
 920 answer, write [ANSWER END]  
 921 Answer:  
 922  $((6 / (1 - 1)) * 4 = 24$ [ANSWER END]  
 923 Feedback: This is not correct. This expression is malformed. Using  
 924 the numbers 1 1 4 6 please provide a correct expression that  
 925 evaluates to 24. Write your answer first. At the end of your  
 926 answer, write [ANSWER END]  
 927 Answer:  
 928  $((6 / (1 - 1)) * 4)$  is undefined because it involves division by  
 929 zero. A correct solution could be:  
 930  $(6 / (1 / 4)) * 1 = 24$ [ANSWER END]  
 931 Feedback: This is not correct. This expression is malformed. Using  
 932 the numbers 1 1 4 6 please provide a correct expression that  
 933 evaluates to 24. Write your answer first. At the end of your  
 934 answer, write [ANSWER END]  
 935 Answer:  
 936  $(6 / (1 / 4)) = 24$ [ANSWER END]  
 937 Feedback: This is not correct. This expression consists of the  
 938 numbers 6, 1, 4, but it has to consist of only and exactly  
 939 ['1', '1', '4', '6']. Using the numbers 1 1 4 6 please provide  
 940 a correct expression that evaluates to 24. Write your answer  
 941 first. At the end of your answer, write [ANSWER END]  
 942 Answer:  
 943  $(6 - 1) * (4 - 1) = 24$ [ANSWER END]  
 944 Feedback: This is not correct. This expression evaluates to 15  
 945 instead of 24. Using the numbers 1 1 4 6 please provide a  
 946 correct expression that evaluates to 24. Write your answer  
 947 first. At the end of your answer, write [ANSWER END]  
 948 Answer:  
 949  $(6 / (1 / (4 - 1))) = 24$ [ANSWER END]  
 950 Feedback: This is not correct. This expression evaluates to 18  
 951 instead of 24. Using the numbers 1 1 4 6 please provide a  
 952 correct expression that evaluates to 24. Write your answer  
 953 first. At the end of your answer, write [ANSWER END]  
 954 Answer:  
 955  $(6 * (4 - 1)) - 1 = 24$ [ANSWER END]  
 956 Feedback: This is not correct. This expression evaluates to 17  
 957 instead of 24. Using the numbers 1 1 4 6 please provide a  
 958 correct expression that evaluates to 24. Write your answer  
 959 first. At the end of your answer, write [ANSWER END]  
 960 Answer:

## 957 A.4 GRAPH COLORING

### 958 A.4.1 HALLUCINATIONS IN GRAPH COLORING CRITIQUE

961 The following is a more in-depth look at GPT-4's critique abilities on the graph coloring problem.

962 For each instance, we generated five different kinds of colorings: correct, ablated (exactly one edge is  
 963 wrong), non-optimal (a coloring with no constraint violations except that it uses too many colors),  
 964 random, and LLM (sampled from LLM generations). For each of these 500 proposed colorings, we  
 965 sent a query to the LLM asking it to verify if the answer was right, and—if not—to output a free-form  
 966 critique. We then parsed these to determine what edges it said were colored the same at both vertices.  
 967

968 Table A4 summarizes the results. Note that, proportionally, hallucinations decrease when the error  
 969 rate of the domain increases. That is to say, when there are more incorrect edges, the model is more  
 970 likely to point to one of them. Intuitively, this makes sense: it's easier to guess one edge which is  
 971 wrong when half of all the edges are miscolored, as is the case on average among randomly colored  
 graphs.

Table A4: Distribution of hallucinations during verification task. This table counts the number of instances that featured each type of hallucination and compares it to the total number of erroneous edges encountered across all coloring instances in each subset. Note that the correct column tracks the number of colorings where the *ground truth* is that the coloring is correct.

|             | Hallucinations |      |      |      | Coloring |         |
|-------------|----------------|------|------|------|----------|---------|
|             | Vertex         | Edge | Both | None | Errors   | Correct |
| Correct     | 29             | 72   | 7    | 2    | 0        | 100     |
| Ablated     | 24             | 52   | 5    | 24   | 187      | 0       |
| Non-optimal | 18             | 65   | 3    | 10   | 0        | 0       |
| Random      | 10             | 26   | 5    | 66   | 736      | 0       |
| LLM         | 26             | 41   | 6    | 27   | 240      | 18      |
| Total       | 107            | 256  | 26   | 129  | 282      | 118     |

Edge hallucinations are more common than vertex. Essentially, typical behavior is to pick two vertices that are the same color in the coloring, but which aren't associated by an edge in the graph description, and claim that they are connected and thus illegally colored. Vertex color hallucination is when the reverse happens: instead of ascribing an edge to same-color nodes, the colorings of two connected vertices are misstated. The overlap between the two cases, where a non-existent edge is declared to be violated by non-existent colorings is much rarer than either. Note that it never hallucinates new vertex *names*, only that vertices which are in graph have colors differing from reality.

Even rarer cases did spring up in the response data. At times the model lost track of the question being asked and reversed it, explicitly claiming that two same-colored vertices violate the conditions because they *aren't* connected; or it began to contradict itself mid-deduction, making multiple claims about a vertex's color.

We present these examples here. In the following, multiple equals signs in a row ("====") are dividers between separate examples, not parts of the examples.

#### Examples of LLM Output on the Verification Task

```

The vertices 1 and 4 are both color 0 and share an edge.
=====
The vertices 9 and 12 are both colored with Color 1 and they share
an edge.
=====
Vertex 1: Color 2 is connected to Vertex 2: Color 2.
=====
Vertex 5 and vertex 10 are both Color 2 and share an edge.
=====
Same-color vertices 0 and 1 share an edge.
=====
Vertex 0 (Color 0) and vertex 3 (Color 0) share an edge. Vertex 1
(Color 1) and vertex 2 (Color 1) share an edge. Vertex 4 (Color
1) and vertex 8 (Color 1) share an edge. Vertex 2 (Color 1) and
vertex 4 (Color 1) share an edge. Vertex 4 (Color 1) and vertex
11 (Color 0) share an edge. Vertex 12 (Color 2) and vertex 15
(Color 2) share an edge.
The vertices 3 and 0 both have Color 2 and they are adjacent. The
vertices 1 and 11 both have Color 1 and they are adjacent. The
vertices 2 and 5 both have Color 0 and they are adjacent.
Therefore, the given coloring is incorrect.
=====
0: Color 0 and 9: Color 0 share an edge.
0: Color 0 and 10: Color 0 share an edge.
0: Color 0 and 14: Color 0 share an edge.
9: Color 0 and 13: Color 0 share an edge.
=====

```

1026  
 1027  
 1028  
 1029  
 1030  
 1031  
 1032  
 1033  
 1034  
 1035  
 1036  
 1037  
 1038  
 1039  
 1040  
 1041  
 1042  
 1043  
 1044  
 1045  
 1046  
 1047  
 1048  
 1049  
 1050  
 1051  
 1052  
 1053  
 1054  
 1055  
 1056  
 1057  
 1058  
 1059  
 1060  
 1061  
 1062  
 1063  
 1064  
 1065  
 1066  
 1067  
 1068  
 1069  
 1070  
 1071  
 1072  
 1073  
 1074  
 1075  
 1076  
 1077  
 1078  
 1079

Vertex 3 is colored with Color 4, which exceeds the optimal coloring number of 4. Therefore, this coloring is non-optimal.  
 =====  
 Vertex 0 is missing in the coloring. Missing assignment.  
 =====  
 Vertex 0 (color 3) is connected to vertex 3 (color 2), sharing an edge. Vertex 0 (color 3) is also connected to vertex 4 (color 0), and to vertex 5 (color 0), and to vertex 9 (color 0), and to vertex 12 (color 2), sharing an edge with them. Therefore, this assignment is correctly colored with color 0, color 2, and color 3, making use of 3 different colors.  
  
 Vertex 1 (color 2) is connected to vertex 4 (color 0), and to vertex 5 (color 0), and to vertex 6 (color 0), and to vertex 7 (color 3), and to vertex 8 (color 0), and to vertex 10 (color 1), and to vertex 11 (color 1), sharing an edge with them. Therefore, this assignment is also correctly colored.  
  
 Vertex 2 (color 0) is connected to vertex 12 (color 2), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 3 (color 2) is connected to vertex 5 (color 0), and to vertex 8 (color 0), and to vertex 10 (color 1), sharing an edge with them. Therefore, this assignment is correctly colored.  
  
 Vertex 4 (color 0) is connected to vertex 7 (color 3), and to vertex 11 (color 1), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 5 (color 0) is connected to vertex 7 (color 3), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 6 (color 0) is connected to vertex 10 (color 1), and to vertex 12 (color 2), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 8 (color 0) is connected to vertex 10 (color 1), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 9 (color 0) is connected to vertex 11 (color 1), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 10 (color 1) is connected to vertex 12 (color 2), sharing an edge. Therefore, this assignment is correctly colored.  
  
 Vertex 11 (color 1) isn't connected to vertex 10 (color 1), hence, this assignment is correctly colored.  
  
 Vertex 12 (color 2) isn't connected to vertex 2 (color 0), hence, this assignment is correctly colored.  
  
 No adjacent vertices share a same color, and no colors exceed the optimal coloring number. Hence, the coloring is correct.  
 Verifier confirmed success.

#### Examples of (Rare) Mangled, Illogical, and Otherwise Strange Output on the Verification Task

Vertex 10, colored 4, is connected to vertex 11, also colored 1. However, it is also colored 4, which is greater than the optimal coloring of 4. So, this coloring is non-optimal.  
 =====

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

Vertex 9 (Color 2) and Vertex 10 (Color 1) are connected and share the same color. Vertex 9 (Color 2) and Vertex 11 (Color 1) are also connected and share the same color.

=====

Vertex 2 (Color 0) is connected to Vertex 0 (Color 0).  
Vertex 5 (Color 0) is connected to Vertex 1 (Color 2) and Vertex 14 (Color 1), but also to Vertex 7 (Color 1).  
Vertex 6 (Color 0) is connected to Vertex 3 (Color 2), Vertex 10 (Color 1) and 13 (Color 2), but also to Vertex 8 (Color 2).

=====

The coloring is incorrect. Vertex 1 (Color 2) and Vertex 3 (Color 3) are connected but have different colors. Vertex 3 (Color 3) is connected to Vertex 6 (Color 3).

=====

Vertex 10, colored 4, is connected to vertex 11, also colored 1. However, it is also colored 4, which is greater than the optimal coloring of 4. So, this coloring is non-optimal.

#### A.4.2 PROMPTS

All of following examples are built on the same graph instance.

##### DIMACS Format For Graphs

```
e 0 7
e 0 8
e 0 9
e 0 11
e 1 13
e 2 9
e 3 8
e 3 11
e 3 12
e 4 12
e 5 11
e 6 9
e 7 10
e 7 13
e 9 11
e 10 13
e 11 13
c OPTIMAL CHROMATIC NUMBER === 3
```

##### Baseline, Direct Prompt

Color the following graph, described as a set of edges, such that no two vertices on the same edge share a color.  
You may use at most 3 colors.

- Vertex 0 is connected to vertex 7.
- Vertex 0 is connected to vertex 8.
- Vertex 0 is connected to vertex 9.
- Vertex 0 is connected to vertex 11.
- Vertex 1 is connected to vertex 13.
- Vertex 2 is connected to vertex 9.
- Vertex 3 is connected to vertex 8.
- Vertex 3 is connected to vertex 11.
- Vertex 3 is connected to vertex 12.
- Vertex 4 is connected to vertex 12.
- Vertex 5 is connected to vertex 11.

1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187

Vertex 6 is connected to vertex 9.  
Vertex 7 is connected to vertex 10.  
Vertex 7 is connected to vertex 13.  
Vertex 9 is connected to vertex 11.  
Vertex 10 is connected to vertex 13.  
Vertex 11 is connected to vertex 13.  
There are a total of 14 vertices. Please label every vertex, even if it is disconnected from the rest of the graph. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

#### Example LLM Response

0: Color1  
1: Color1  
2: Color1  
3: Color2  
4: Color1  
5: Color2  
6: Color2  
7: Color2  
8: Color3  
9: Color3  
10: Color3  
11: Color3  
12: Color3  
13: Color1

#### Prompt To Elicit Verification

The following graph, described as a set of edges, has an optimal coloring number of 3:  
Vertex 0 is connected to vertex 7.  
Vertex 0 is connected to vertex 8.  
Vertex 0 is connected to vertex 9.  
Vertex 0 is connected to vertex 11.  
Vertex 1 is connected to vertex 13.  
Vertex 2 is connected to vertex 9.  
Vertex 3 is connected to vertex 8.  
Vertex 3 is connected to vertex 11.  
Vertex 3 is connected to vertex 12.  
Vertex 4 is connected to vertex 12.  
Vertex 5 is connected to vertex 11.  
Vertex 6 is connected to vertex 9.  
Vertex 7 is connected to vertex 10.  
Vertex 7 is connected to vertex 13.  
Vertex 9 is connected to vertex 11.  
Vertex 10 is connected to vertex 13.  
Vertex 11 is connected to vertex 13.  
Please check if this coloring is correct:0: Color1  
1: Color1  
2: Color1  
3: Color2  
4: Color1  
5: Color2  
6: Color2  
7: Color2

1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241

```

8: Color3
9: Color3
10: Color3
11: Color3
12: Color3
13: Color1
If it is, say 'Verifier confirmed success.' Do not provide
anything else in your response. If it is incorrect, please
point out which same-color vertices share an edge.

```

#### Prompt To Elicit CoT Verification

```

[Instructions]
When outputting your final answer, first print the [Answer] tag,
then put your final answer after the [Answer] tag. Respond only
in the following format:
Wrong Edges: a list of incorrect edges
All Vertices Colored: boolean representing if every vertex is
colored
Optimal Or Less: boolean representing if the number of colors is
no more than the optimal
Correct: boolean

[Graph]
The following graph, described as a set of edges, has an optimal
coloring number of 3:
Vertex 0 is connected to vertex 7.
Vertex 0 is connected to vertex 8.
Vertex 0 is connected to vertex 9.
Vertex 0 is connected to vertex 11.
Vertex 1 is connected to vertex 13.
Vertex 2 is connected to vertex 9.
Vertex 3 is connected to vertex 8.
Vertex 3 is connected to vertex 11.
Vertex 3 is connected to vertex 12.
Vertex 4 is connected to vertex 12.
Vertex 5 is connected to vertex 11.
Vertex 6 is connected to vertex 9.
Vertex 7 is connected to vertex 10.
Vertex 7 is connected to vertex 13.
Vertex 9 is connected to vertex 11.
Vertex 10 is connected to vertex 13.
Vertex 11 is connected to vertex 13.

[Coloring]
A coloring is correct if no adjacent vertices are the same color
and the total number of colors used is no more than the optimal
coloring number. Please check if this coloring is correct: 0:
Color1
1: Color1
2: Color1
3: Color1
4: Color1
5: Color1
6: Color1
7: Color2
8: Color2
9: Color2
10: Color1
11: Color3
12: Color2

```

1242  
 1243  
 1244  
 1245  
 1246  
 1247  
 1248  
 1249  
 1250  
 1251  
 1252  
 1253  
 1254  
 1255  
 1256  
 1257  
 1258  
 1259  
 1260  
 1261  
 1262  
 1263  
 1264  
 1265  
 1266  
 1267  
 1268  
 1269  
 1270  
 1271  
 1272  
 1273  
 1274  
 1275  
 1276  
 1277  
 1278  
 1279  
 1280  
 1281  
 1282  
 1283  
 1284  
 1285  
 1286  
 1287  
 1288  
 1289  
 1290  
 1291  
 1292  
 1293  
 1294  
 1295

13: Color3  
 [ANSWER END]

Let's think step by step. Remember to output your final answer in the format described in the instructions.  
 [Thoughts]

### A.4.3 BACKPROMPTS

#### Backprompt Generated From Self-Critique

This is incorrect. Feedback:  
 Vertices 0 and 3 share an edge and are both colored with Color1.  
 Vertices 9 and 11 share an edge and are both colored with Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

#### Backprompt Generated From Sound Verifier (Pass/Fail)

This is not correct. Using the previously provided graph, please provide a correct coloring. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

#### Backprompt Generated From Sound Verifier (First)

Vertex 9 and vertex 11 were both colored Color3 despite being connected by an edge.

This is wrong. Please recolor. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

#### Backprompt Generated From Sound Verifier (Full)

Vertex 9 and vertex 11 were both colored Color3 despite being connected by an edge. Vertex 10 and vertex 13 were both colored Color3 despite being connected by an edge. Vertex 11 and vertex 13 were both colored Color3 despite being connected by an edge.

This is wrong. Please recolor. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.



1296 A.4.4 COMPLETE EXAMPLE OF ITERATIVE PROMPT SEQUENCE  
 1297

1298 Full Sequence of 15 (LLM-Generated) Backprompts  
 1299

1300 Color the following graph, described as a set of edges, such that  
 1301 no two vertices on the same edge share a color.  
 1302 You may use at most 3 colors.  
 1303 Vertex 0 is connected to vertex 7.  
 1304 Vertex 0 is connected to vertex 8.  
 1305 Vertex 0 is connected to vertex 9.  
 1306 Vertex 0 is connected to vertex 11.  
 1307 Vertex 1 is connected to vertex 13.  
 1308 Vertex 2 is connected to vertex 9.  
 1309 Vertex 3 is connected to vertex 8.  
 1310 Vertex 3 is connected to vertex 11.  
 1311 Vertex 3 is connected to vertex 12.  
 1312 Vertex 4 is connected to vertex 12.  
 1313 Vertex 5 is connected to vertex 11.  
 1314 Vertex 6 is connected to vertex 9.  
 1315 Vertex 7 is connected to vertex 10.  
 1316 Vertex 7 is connected to vertex 13.  
 1317 Vertex 9 is connected to vertex 11.  
 1318 Vertex 10 is connected to vertex 13.  
 1319 Vertex 11 is connected to vertex 13.  
 1320 There are a total of 14 vertices. Please label every vertex, even  
 1321 if it is disconnected from the rest of the graph. Please provide  
 1322 each vertex's color. Do not skip any vertices. Each color must  
 1323 be provided on a new line in the response and should be  
 1324 formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}".  
 1325 Please do not provide anything else in your response.  
 1326  
 1327 0: Color1  
 1328 1: Color1  
 1329 2: Color1  
 1330 3: Color1  
 1331 4: Color1  
 1332 5: Color2  
 1333 6: Color2  
 1334 7: Color2  
 1335 8: Color2  
 1336 9: Color3  
 1337 10: Color3  
 1338 11: Color3  
 1339 12: Color2  
 1340 13: Color3  
 1341  
 1342 This is incorrect. Feedback:  
 1343 Vertices 0 and 3 share an edge and are both colored with Color1.  
 1344 Vertices 9 and 11 share an edge and are both colored with  
 1345 Color3.  
 1346  
 1347 Using this feedback, please try again. Please provide each  
 1348 vertex's color. Do not skip any vertices. Each color must be  
 1349 provided on a new line in the response and should be formatted  
 as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not  
 provide anything else in your response.  
 0: Color1  
 1: Color1  
 2: Color2  
 3: Color2  
 4: Color2  
 5: Color1

1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403

```
6: Color1
7: Color2
8: Color3
9: Color3
10: Color1
11: Color2
12: Color3
13: Color3
```

This is incorrect. Feedback:

Vertices 2 and 3 are both colored with Color2 and share an edge.  
Vertices 7 and 11 are both colored with Color2 and share an edge. Vertices 9 and 13 are both colored with Color3 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color1
2: Color2
3: Color3
4: Color3
5: Color3
6: Color3
7: Color3
8: Color2
9: Color1
10: Color2
11: Color2
12: Color1
13: Color1
```

This is incorrect. Feedback:

Vertices 0 and 9 share an edge and are both colored with Color1.  
Vertices 0 and 11 share an edge and are both colored with Color2. Vertices 1 and 13 share an edge and are both colored with Color1. Vertices 7 and 13 share an edge and are both colored with Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color1
2: Color2
3: Color2
4: Color3
5: Color1
6: Color1
7: Color2
8: Color3
9: Color3
10: Color1
11: Color3
12: Color1
13: Color3
```

1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457

This is incorrect. Feedback:

Vertices 0 and 5 both have Color1 and share an edge. Vertices 0 and 11 both have Color3 and share an edge. Vertices 7 and 10 both have Color2 and share an edge. Vertices 11 and 13 both have Color3 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color1
2: Color2
3: Color2
4: Color1
5: Color1
6: Color1
7: Color2
8: Color3
9: Color3
10: Color3
11: Color2
12: Color3
13: Color1
```

This is incorrect. Feedback:

Vertices 0 and 1 both have Color1 and share an edge. Vertices 0 and 5 also both have Color1 and share an edge. Vertices 0 and 6 also both have Color1 and share an edge. Vertices 7 and 11 both have Color2 and share an edge. Vertices 9 and 11 both have Color2 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color2
2: Color2
3: Color2
4: Color2
5: Color3
6: Color3
7: Color3
8: Color2
9: Color3
10: Color1
11: Color2
12: Color3
13: Color1
```

This is incorrect. Feedback:

Vertices 0 and 8 both have Color1 and share an edge. Vertices 0 and 11 both have Color1 and share an edge. Vertices 3 and 8 both have Color2 and share an edge. Vertices 3 and 11 both have Color2 and share an edge. Vertices 7 and 13 both have Color3 and share an edge. Vertices 9 and 11 both have Color3 and share an edge. Vertices 11 and 13 both have Color2 and share an edge.

1458  
 1459  
 1460  
 1461  
 1462  
 1463  
 1464  
 1465  
 1466  
 1467  
 1468  
 1469  
 1470  
 1471  
 1472  
 1473  
 1474  
 1475  
 1476  
 1477  
 1478  
 1479  
 1480  
 1481  
 1482  
 1483  
 1484  
 1485  
 1486  
 1487  
 1488  
 1489  
 1490  
 1491  
 1492  
 1493  
 1494  
 1495  
 1496  
 1497  
 1498  
 1499  
 1500  
 1501  
 1502  
 1503  
 1504  
 1505  
 1506  
 1507  
 1508  
 1509  
 1510  
 1511

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color1
6: Color1
7: Color3
8: Color2
9: Color3
10: Color2
11: Color2
12: Color3
13: Color1
```

This is incorrect. Feedback:  
 Vertices 0 and 11 share an edge and are both colored with Color1.  
 Vertices 2 and 9 share an edge and are both colored with Color2. Vertices 3 and 12 share an edge and are both colored with Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color2
2: Color2
3: Color2
4: Color3
5: Color3
6: Color3
7: Color2
8: Color3
9: Color1
10: Color3
11: Color3
12: Color1
13: Color1
```

This is incorrect. Feedback:  
 Vertices 11 and 13 both have Color3 and share an edge.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color2
2: Color2
3: Color2
4: Color3
```

1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565

```
5: Color3
6: Color3
7: Color2
8: Color3
9: Color1
10: Color3
11: Color1
12: Color1
13: Color2
```

This is incorrect. Feedback:  
Vertices 0 and 11 share an edge and both are colored with Color1.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color3
8: Color2
9: Color3
10: Color1
11: Color2
12: Color2
13: Color1
```

This is incorrect. Feedback:  
Vertices 2 and 9 share an edge and are both colored with Color2.  
Similarly, vertices 11 and 13 share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

```
0: Color1
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color3
8: Color2
9: Color1
10: Color2
11: Color3
12: Color2
13: Color1
```

This is incorrect. Feedback:  
Vertices 2 and 9 share an edge and are both colored with Color2.  
Vertices 3 and 11 share an edge and are both colored with

1566  
 1567  
 1568  
 1569  
 1570  
 1571  
 1572  
 1573  
 1574  
 1575  
 1576  
 1577  
 1578  
 1579  
 1580  
 1581  
 1582  
 1583  
 1584  
 1585  
 1586  
 1587  
 1588  
 1589  
 1590  
 1591  
 1592  
 1593  
 1594  
 1595  
 1596  
 1597  
 1598  
 1599  
 1600  
 1601  
 1602  
 1603  
 1604  
 1605  
 1606  
 1607  
 1608  
 1609  
 1610  
 1611  
 1612  
 1613  
 1614  
 1615  
 1616  
 1617  
 1618  
 1619

Color3. Vertices 7 and 13 share an edge and are both colored with Color3.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

0: Color1  
 1: Color2  
 2: Color2  
 3: Color3  
 4: Color1  
 5: Color2  
 6: Color3  
 7: Color2  
 8: Color3  
 9: Color3  
 10: Color1  
 11: Color2  
 12: Color3  
 13: Color1

This is incorrect. Feedback:

Vertices 0 and 11 share an edge and are both colored with Color1. Vertices 3 and 8 share an edge and are both colored with Color3. Vertices 7 and 13 share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

0: Color1  
 1: Color2  
 2: Color2  
 3: Color3  
 4: Color1  
 5: Color2  
 6: Color3  
 7: Color2  
 8: Color3  
 9: Color1  
 10: Color3  
 11: Color2  
 12: Color1  
 13: Color3

This is incorrect. Feedback:

The coloring is incorrect. Vertices 0 and 11 share an edge and are both colored with Color1. Vertices 7 and 13 share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each vertex's color. Do not skip any vertices. Each color must be provided on a new line in the response and should be formatted as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not provide anything else in your response.

0: Color1

1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673

```
1: Color2
2: Color2
3: Color3
4: Color1
5: Color2
6: Color3
7: Color3
8: Color2
9: Color1
10: Color2
11: Color3
12: Color1
13: Color1
```

This is incorrect. Feedback:

Vertices 0 and 9 share an edge and are both colored with Color1.  
Vertices 7 and 11 share an edge and are both colored with  
Color3. Vertices 1 and 13 share an edge and are both colored  
with Color2.

Using this feedback, please try again. Please provide each  
vertex's color. Do not skip any vertices. Each color must be  
provided on a new line in the response and should be formatted  
as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not  
provide anything else in your response.

```
0: Color1
1: Color2
2: Color3
3: Color2
4: Color2
5: Color1
6: Color2
7: Color3
8: Color3
9: Color2
10: Color1
11: Color3
12: Color1
13: Color1
```

This is incorrect. Feedback:

Vertices 3 and 9 share an edge and are both colored with Color2.

Using this feedback, please try again. Please provide each  
vertex's color. Do not skip any vertices. Each color must be  
provided on a new line in the response and should be formatted  
as "{VERTEX NUMBER}: {VERTEX COLOR ASSIGNMENT}". Please do not  
provide anything else in your response.

```
0: Color1
1: Color2
2: Color3
3: Color2
4: Color3
5: Color1
6: Color2
7: Color3
8: Color2
9: Color1
10: Color2
11: Color3
12: Color1
```

|                   | Binary Verification | Type Identified | Critique Generation |
|-------------------|---------------------|-----------------|---------------------|
| Correct           | 78/100 (78%)        | 78/100 (78%)    | 78/100 (78%)        |
| Inexecutable      | 70/100 (70%)        | 63/100 (63%)    | 8/100 (8%)          |
| Non Goal Reaching | 98/100 (98%)        | 12/100 (12%)    | 12/100 (12%)        |
| Random            | 100/100 (100%)      | 94/100 (94%)    | 2/100 (2%)          |
| LLM               | Correct             | 53/55 (96.36%)  | 53/55 (96.36%)      |
|                   | Inexecutable        | 25/40 (62.5%)   | 24/40 (60%)         |
|                   | Non Goal Reaching   | 3/5 (60%)       | 2/5 (40%)           |

Table A5: This table presents the verification and critique accuracy of LLM-as-verifier across five different kinds of plans over 100 instances of Blocksworld.

|                   | Binary Verification | Type Identified | Critique Generation |
|-------------------|---------------------|-----------------|---------------------|
| Correct           | 3/100 (3%)          | 3/100 (3%)      | 3/100 (3%)          |
| Inexecutable      | 100/100 (100%)      | 100/100 (100%)  | 24/100 (24%)        |
| Non Goal Reaching | 98/100 (98%)        | 12/100 (12%)    | 12/100 (12%)        |
| Random            | 100/100 (100%)      | 100/100 (100%)  | 59/100 (100%)       |
| LLM               | Correct             | 0/3 (0%)        | 0/3 (0%)            |
|                   | Inexecutable        | 89/89 (100%)    | 89/89 (100%)        |
|                   | Non Goal Reaching   | 8/8 (100%)      | 0/8 (0%)            |

Table A6: This table presents the verification and critique accuracy of LLM-as-verifier across five different kinds of plans over 100 instances of Mystery Blocksworld.

13: Color1

## A.5 PLANNING

### A.5.1 ACCURACY OF PLANNING CRITIQUE

For each instance, we generated five different kind of plans: correct, inexecutable (where an action is inexecutable and the plan is thus invalid), non goal reaching (all actions are executable but the plan does not reach the desired state), random, and LLM (sampled from the LLM generations). For each of these prompts, we sent a query to the LLM asking it to verify the plan and critique it in a certain format. We provide an example in the following prompts section. Specifically, if the plan is valid, the LLM has to just output that. If a plan is invalid and inexecutable, it has to provide the first action that the plan is rendered invalid and the unmet preconditions for that action. If the plan is invalid and non goal reaching, it has to provide the unmet goals for the plan. We evaluate the binary verification and the critique by comparing it to the ground truth provided by VAL (Howey et al., 2004). We check the binary verification, whether or not the detected the type of invalidity (inexecutable or non goal reaching) and if the critique is valid. Tables A5 and A6 show the inability of LLMs in generating the right critique for proposed solutions even though they get the label correct.

### A.5.2 PROMPTS - BLOCKSWORLD

All of following examples are built on the same graph instance.



1728  
 1729  
 1730  
 1731  
 1732  
 1733  
 1734  
 1735  
 1736  
 1737  
 1738  
 1739  
 1740  
 1741  
 1742  
 1743  
 1744  
 1745  
 1746  
 1747  
 1748  
 1749  
 1750  
 1751  
 1752  
 1753  
 1754  
 1755  
 1756  
 1757  
 1758  
 1759  
 1760  
 1761  
 1762  
 1763  
 1764  
 1765  
 1766  
 1767  
 1768  
 1769  
 1770  
 1771  
 1772  
 1773  
 1774  
 1775  
 1776  
 1777  
 1778  
 1779  
 1780  
 1781

### Baseline, Direct Prompt

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block  
 Unstack a block from on top of another block  
 Put down a block  
 Stack a block on top of another block

I have the following restrictions on my actions:  
 I can only pick up or unstack one block at a time.  
 I can only pick up or unstack a block if my hand is empty.  
 I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.  
 I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.  
 I can only unstack a block from on top of another block if the block I am unstacking is clear.  
 Once I pick up or unstack a block, I am holding the block.  
 I can only put down a block that I am holding.  
 I can only stack a block on top of another block if I am holding the block being stacked.  
 I can only stack a block on top of another block if the block onto which I am stacking the block is clear.  
 Once I put down or stack a block, my hand becomes empty.  
 Once you stack a block on top of a second block, the second block is no longer clear.

[STATEMENT]

As initial conditions I have that, the red block is clear, the yellow block is clear, the hand is empty, the red block is on top of the blue block, the yellow block is on top of the orange block, the blue block is on the table and the orange block is on the table.

My goal is to have that the orange block is on top of the red block.

My plan is as follows:

[PLAN]

unstack the yellow block from on top of the orange block  
 put down the yellow block  
 pick up the orange block  
 stack the orange block on top of the red block

[PLAN END]

[STATEMENT]

As initial conditions I have that, the blue block is clear, the hand is empty, the blue block is on top of the orange block, the orange block is on top of the yellow block, the yellow block is on top of the red block and the red block is on the table.

My goal is to have that the red block is on top of the orange block and the yellow block is on top of the red block.

My plan is as follows:

[PLAN]

1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835

**Example LLM Response**

```
unstack the blue block from on top of the orange block
put down the blue block
unstack the orange block from on top of the yellow block
put down the orange block
unstack the yellow block from on top of the red block
put down the yellow block
pick up the red block
stack the red block on top of the orange block
pick up the yellow block
stack the yellow block on top of the red block
[PLAN END]
```

**Prompt To Elicit Verification (Open Ended)**

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block  
Unstack a block from on top of another block  
Put down a block  
Stack a block on top of another block

I have the following restrictions on my actions:  
I can only pick up or unstack one block at a time.  
I can only pick up or unstack a block if my hand is empty.  
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.  
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.  
I can only unstack a block from on top of another block if the block I am unstacking is clear.  
Once I pick up or unstack a block, I am holding the block.  
I can only put down a block that I am holding.  
I can only stack a block on top of another block if I am holding the block being stacked.  
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.  
Once I put down or stack a block, my hand becomes empty.  
Once you stack a block on top of a second block, the second block is no longer clear.

[STATEMENT]  
As initial conditions I have that, the blue block is clear, the orange block is clear, the hand is empty, the orange block is on top of the yellow block, the yellow block is on top of the red block, the red block is on the table and the blue block is on the table.  
My goal is to have that the red block is on top of the orange block, the blue block is on top of the red block and the yellow block is on top of the blue block.  
My plan is as follows:

[PLAN]  
unstack the orange block from on top of the yellow block  
put down the orange block  
unstack the yellow block from on top of the red block  
put down the yellow block  
pick up the blue block  
stack the blue block on top of the red block

1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889

```
pick up the orange block
stack the orange block on top of the blue block
pick up the yellow block
stack the yellow block on top of the orange block
[PLAN END]

Verify whether the above plan is valid. If it is valid, please say
  "Plan is valid." and nothing else. If it is invalid, please say
  "Plan is invalid." and then provide feedback on why the plan
  fails.
```

**Prompt To Elicit Verification (Format based)**

```
I am playing with a set of blocks where I need to arrange the
  blocks into stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the
  block is clear. A block is clear if the block has no other
  blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the
  block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the
  block I am unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding
  the block being stacked.
I can only stack a block on top of another block if the block onto
  which I am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.
Once you stack a block on top of a second block, the second block
  is no longer clear.

[STATEMENT]
As initial conditions I have that, the blue block is clear, the
  orange block is clear, the hand is empty, the orange block is
  on top of the yellow block, the yellow block is on top of the
  red block, the red block is on the table and the blue block is
  on the table.
My goal is to have that the red block is on top of the orange
  block, the blue block is on top of the red block and the yellow
  block is on top of the blue block.
My plan is as follows:

[PLAN]
unstack the orange block from on top of the yellow block
put down the orange block
unstack the yellow block from on top of the red block
put down the yellow block
pick up the red block
stack the red block on top of the orange block
pick up the blue block
stack the blue block on top of the red block
```

1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943

```
pick up the yellow block
stack the yellow block on top of the blue block
[PLAN END]
```

Verify whether the above plan is valid. Provide a JSON between tags [JSON] and [JSON\_END] for the verification information. The JSON should contain three main keys: (1) "valid": a binary value that tells if the plan is valid or not i.e., the plan when executed satisfies the goal conditions. If the plan is invalid and inexecutable then include (2) "unmet\_preconditions": This contains two more keys; (2.1) "action": This is the name of the first action that renders the plan inexecutable (2.2) "preconditions": A list of unmet preconditions for the mentioned action; If the plan is executable but not goal reaching then include (3) "unmet\_goals": A list of unmet goal conditions in the JSON. Include only one of the keys (2) or (3) based on the plan invalidity.

#### Prompt To Elicit Verification (Chain of thought based)

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

```
Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block
```

I have the following restrictions on my actions:  
I can only pick up or unstack one block at a time.  
I can only pick up or unstack a block if my hand is empty.  
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.  
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.  
I can only unstack a block from on top of another block if the block I am unstacking is clear.  
Once I pick up or unstack a block, I am holding the block.  
I can only put down a block that I am holding.  
I can only stack a block on top of another block if I am holding the block being stacked.  
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.  
Once I put down or stack a block, my hand becomes empty.  
Once you stack a block on top of a second block, the second block is no longer clear.

```
[STATEMENT]
As initial conditions I have that, the red block is clear, the
yellow block is clear, the hand is empty, the red block is on
top of the blue block, the yellow block is on top of the orange
block, the blue block is on the table and the orange block is
on the table.
My goal is to have that the orange block is on top of the red
block.
My plan is as follows:

[PLAN]
unstack the yellow block from on top of the orange block
```

1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997

```
put down the yellow block
pick up the orange block
stack the orange block on top of the red block
[PLAN END]
```

Verify whether the above plan is valid. You will think step by step and output intermediate reasoning steps and thoughts for the verification after the [THOUGHTS] tag. Then, provide a JSON between the tags [JSON] and [JSON\_END] for the verification information. The JSON should contain three main keys: If the plan is invalid and inexecutable then include (1) "unmet\_preconditions": This contains two more keys; (1.1) "action": This is the name of the first action that renders the plan inexecutable (1.2) "preconditions": A list of unmet preconditions for the mentioned action; If the plan is executable but not goal reaching then include (2) "unmet\_goals": A list of unmet goal conditions in the JSON. Finally include (3) "valid": a binary value that tells if the plan is valid or not i.e., the plan when executed satisfies the goal conditions. Include only one of the keys (1) or (2) based on the type of plan invalidity.

```
Let's think step by step
[THOUGHTS]
```

#### Prompt To Elicit Verification (Swapping Answer and Reason Order)

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

```
Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block
```

I have the following restrictions on my actions:  
I can only pick up or unstack one block at a time.  
I can only pick up or unstack a block if my hand is empty.  
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.  
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.  
I can only unstack a block from on top of another block if the block I am unstacking is clear.  
Once I pick up or unstack a block, I am holding the block.  
I can only put down a block that I am holding.  
I can only stack a block on top of another block if I am holding the block being stacked.  
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.  
Once I put down or stack a block, my hand becomes empty.  
Once you stack a block on top of a second block, the second block is no longer clear.

```
[STATEMENT]
```

As initial conditions I have that, the red block is clear, the yellow block is clear, the hand is empty, the red block is on top of the blue block, the yellow block is on top of the orange block, the blue block is on the table and the orange block is on the table.

1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051

My goal is to have that the orange block is on top of the red block.

My plan is as follows:

[PLAN]

unstack the yellow block from on top of the orange block

put down the yellow block

pick up the orange block

stack the orange block on top of the red block

[PLAN END]

Verify whether the above plan is valid. Provide a JSON between tags [JSON] and [JSON\_END] for the verification information. The JSON should contain three main keys: If the plan is invalid and inexecutable then include (1) "unmet\_preconditions": This contains two more keys; (1.1) "action": This is the name of the first action that renders the plan inexecutable (1.2) "preconditions": A list of unmet preconditions for the mentioned action; If the plan is executable but not goal reaching then include (2) "unmet\_goals": A list of unmet goal conditions in the JSON. Finally include (3) "valid": a binary value that tells if the plan is valid or not i.e., the plan when executed satisfies the goal conditions. Include only one of the keys (1) or (2) based on the type of plan invalidity.

### A.5.3 BACKPROMPTS - BLOCKSWORLD

#### Backprompt Generated From Self-Critique

Plan is invalid.

The plan fails because after you stack the blue block on top of the red block, you cannot unstack the yellow block from on top of the orange block because the yellow block is not clear. It has the blue block on top of it.

#### Backprompt Generated From Sound Verifier (First)

The above plan is invalid.

The following action at step 7 has an unmet precondition:

pick up the red block

The unmet precondition is:

the red block is clear

#### Backprompt Generated From Sound Verifier (Full)

The above plan is invalid.

There are unsatisfied preconditions.

The following action at step 7 has an unmet precondition:

pick up the red block

The unmet precondition is:

the red block is clear

The following action at step 8 has an unmet precondition:

stack the red block on top of the orange block

The unmet precondition is:

the orange block is clear

The following action at step 11 has an unmet precondition:

pick up the yellow block

The unmet precondition is:

the yellow block is on the table

## A.5.4 COMPLETE EXAMPLE OF ITERATIVE PROMPT SEQUENCE - BLOCKSWORLD

2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105

## Full Sequence of (LLM-Generated) Backprompts

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block  
Unstack a block from on top of another block  
Put down a block  
Stack a block on top of another block

I have the following restrictions on my actions:  
I can only pick up or unstack one block at a time.  
I can only pick up or unstack a block if my hand is empty.  
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.  
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.  
I can only unstack a block from on top of another block if the block I am unstacking is clear.  
Once I pick up or unstack a block, I am holding the block.  
I can only put down a block that I am holding.  
I can only stack a block on top of another block if I am holding the block being stacked.  
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.  
Once I put down or stack a block, my hand becomes empty.  
Once you stack a block on top of a second block, the second block is no longer clear.

[STATEMENT]

As initial conditions I have that, the red block is clear, the blue block is clear, the yellow block is clear, the hand is empty, the blue block is on top of the orange block, the red block is on the table, the orange block is on the table and the yellow block is on the table.  
My goal is to have that the red block is on top of the blue block, the blue block is on top of the yellow block and the yellow block is on top of the orange block.

My plan is as follows:

[PLAN]

unstack the blue block from on top of the orange block  
stack the blue block on top of the red block  
pick up the yellow block  
stack the yellow block on top of the orange block  
unstack the blue block from on top of the red block  
stack the blue block on top of the yellow block  
pick up the red block  
stack the red block on top of the blue block

[PLAN END]

[STATEMENT]

As initial conditions I have that, the blue block is clear, the orange block is clear, the hand is empty, the orange block is on top of the yellow block, the yellow block is on top of the red block, the red block is on the table and the blue block is on the table.

2106  
2107 My goal is to have that the red block is on top of the orange  
2108 block, the blue block is on top of the red block and the yellow  
2109 block is on top of the blue block.

2110 My plan is as follows:

2111 [PLAN]

2112 -----GPT4-as-Planner Response-----  
2113 unstack the orange block from on top of the yellow block  
2114 put down the orange block  
2115 unstack the yellow block from on top of the red block  
2116 stack the yellow block on top of the orange block  
2117 pick up the blue block  
2118 stack the blue block on top of the red block  
2119 unstack the yellow block from on top of the orange block  
2120 stack the yellow block on top of the blue block  
2121 pick up the red block  
2122 unstack the blue block from on top of the red block  
2123 stack the blue block on top of the yellow block  
2124 stack the red block on top of the orange block  
2125 [PLAN END]

2126 -----GPT4-as-Verifier Response-----  
2127 Plan is invalid. The plan fails because after you stack the blue  
2128 block on top of the red block, you cannot unstack the yellow  
2129 block from on top of the orange block because the yellow block  
2130 is not clear. It has the blue block on top of it.

2131 -----GPT4-as-Planner Response-----  
2132 I apologize for the oversight. Here is the corrected plan:

2133 [PLAN]

2134 unstack the orange block from on top of the yellow block  
2135 put down the orange block  
2136 unstack the yellow block from on top of the red block  
2137 put down the yellow block  
2138 pick up the blue block  
2139 stack the blue block on top of the red block  
2140 pick up the orange block  
2141 stack the orange block on top of the red block  
2142 pick up the yellow block  
2143 stack the yellow block on top of the blue block  
2144 [PLAN END]

2145 -----GPT4-as-Verifier Response-----  
2146 Plan is invalid. The plan fails because after stacking the blue  
2147 block on top of the red block, you are trying to stack the  
2148 orange block on top of the red block. However, the red block is  
2149 not clear because the blue block is on top of it.

2150 -----GPT4-as-Planner Response-----  
2151 I apologize for the mistake. Here is the corrected plan:

2152 [PLAN]

2153 unstack the orange block from on top of the yellow block  
2154 put down the orange block  
2155 unstack the yellow block from on top of the red block  
2156 put down the yellow block  
2157 pick up the blue block  
2158 stack the blue block on top of the red block  
2159 pick up the orange block  
stack the orange block on top of the blue block  
pick up the yellow block  
stack the yellow block on top of the orange block  
[PLAN END]

-----GPT4-as-Verifier Response-----  
Plan is valid.



2160 A.5.5 LLM AS VERIFIER - BLOCKSWORLD

2161

2162

2163

2164

2165

2166

2167

2168

2169

2170

2171

2172

2173

2174

2175

2176

2177

2178

2179

2180

2181

2182

2183

2184

2185

2186

2187

2188

2189

2190

2191

2192

2193

2194

2195

2196

2197

2198

2199

2200

2201

2202

2203

2204

2205

2206

2207

2208

2209

2210

2211

2212

2213

#### Examples of LLM Output on the Verification Task

[STATEMENT]

As initial conditions I have that, the blue block is clear, the orange block is clear, the hand is empty, the orange block is on top of the yellow block, the yellow block is on top of the red block, the red block is on the table and the blue block is on the table.

My goal is to have that the red block is on top of the orange block, the blue block is on top of the red block and the yellow block is on top of the blue block.

My plan is as follows:

[PLAN]

-----GPT-4 Response-----  
 unstack the orange block from on top of the yellow block  
 put down the orange block  
 unstack the yellow block from on top of the red block  
 stack the yellow block on top of the orange block  
 pick up the blue block  
 stack the blue block on top of the red block  
 unstack the yellow block from on top of the orange block  
 stack the yellow block on top of the blue block  
 pick up the red block  
 unstack the blue block from on top of the red block  
 stack the blue block on top of the yellow block  
 stack the red block on top of the orange block

[PLAN END]

-----GPT-4 Verifier Incorrect Response-----

Plan is invalid. The plan fails because after you stack the blue block on top of the red block, you cannot unstack the yellow block from on top of the orange block because the yellow block is not clear. It has the blue block on top of it.

#### A.5.6 PROMPTS - MYSTERY BLOCKSWORLD

All of following examples are built on the same graph instance.

##### Baseline, Direct Prompt

I am playing with a set of objects. Here are the actions I can do

Attack object  
 Feast object from another object  
 Succumb object  
 Overcome object from another object

I have the following restrictions on my actions:

To perform Attack action, the following facts need to be true: Province object, Planet object, Harmony.

Once Attack action is performed the following facts will be true: Pain object.

Once Attack action is performed the following facts will be false: Province object, Planet object, Harmony.

To perform Succumb action, the following facts need to be true: Pain object.

Once Succumb action is performed the following facts will be true: Province object, Planet object, Harmony.

2214  
2215  
2216  
2217  
2218  
2219  
2220  
2221  
2222  
2223  
2224  
2225  
2226  
2227  
2228  
2229  
2230  
2231  
2232  
2233  
2234  
2235  
2236  
2237  
2238  
2239  
2240  
2241  
2242  
2243  
2244  
2245  
2246  
2247  
2248  
2249  
2250  
2251  
2252  
2253  
2254  
2255  
2256  
2257  
2258  
2259  
2260  
2261  
2262  
2263  
2264  
2265  
2266  
2267

Once Succumb action is performed the following facts will be false: Pain object.  
To perform Overcome action, the following needs to be true: Province other object, Pain object.  
Once Overcome action is performed the following will be true: Harmony, Province object, Object Craves other object.  
Once Overcome action is performed the following will be false: Province other object, Pain object.  
To perform Feast action, the following needs to be true: Object Craves other object, Province object, Harmony.  
Once Feast action is performed the following will be true: Pain object, Province other object.  
Once Feast action is performed the following will be false:, Object Craves other object, Province object, Harmony.

[STATEMENT]

As initial conditions I have that, object a craves object b, object d craves object c, harmony, planet object b, planet object c, province object a and province object d.  
My goal is to have that object c craves object a.

My plan is as follows:

[PLAN]

feast object d from object c  
succumb object d  
attack object c  
overcome object c from object a  
[PLAN END]

[STATEMENT]

As initial conditions I have that, object b craves object c, object c craves object d, object d craves object a, harmony, planet object a and province object b.  
My goal is to have that object a craves object c and object d craves object a.

My plan is as follows:

[PLAN]

#### Example LLM Response

feast object b from object c  
succumb object b  
attack object a  
overcome object a from object c  
feast object d from object a  
succumb object d  
attack object c  
overcome object c from object a  
[PLAN END]

#### Prompt To Elicit Verification (Open Ended)

I am playing with a set of objects. Here are the actions I can do

Attack object  
Feast object from another object  
Succumb object

2268  
2269  
2270  
2271  
2272  
2273  
2274  
2275  
2276  
2277  
2278  
2279  
2280  
2281  
2282  
2283  
2284  
2285  
2286  
2287  
2288  
2289  
2290  
2291  
2292  
2293  
2294  
2295  
2296  
2297  
2298  
2299  
2300  
2301  
2302  
2303  
2304  
2305  
2306  
2307  
2308  
2309  
2310  
2311  
2312  
2313  
2314  
2315  
2316  
2317  
2318  
2319  
2320  
2321

Overcome object from another object

I have the following restrictions on my actions:  
To perform Attack action, the following facts need to be true:  
Province object, Planet object, Harmony.  
Once Attack action is performed the following facts will be true: Pain object.  
Once Attack action is performed the following facts will be false: Province object, Planet object, Harmony.  
To perform Succumb action, the following facts need to be true: Pain object.  
Once Succumb action is performed the following facts will be true: Province object, Planet object, Harmony.  
Once Succumb action is performed the following facts will be false: Pain object.  
To perform Overcome action, the following needs to be true:  
Province other object, Pain object.  
Once Overcome action is performed the following will be true:  
Harmony, Province object, Object Craves other object.  
Once Overcome action is performed the following will be false:  
Province other object, Pain object.  
To perform Feast action, the following needs to be true:  
Object Craves other object, Province object, Harmony.  
Once Feast action is performed the following will be true:  
Pain object, Province other object.  
Once Feast action is performed the following will be false: ,  
Object Craves other object, Province object, Harmony.

[STATEMENT]  
As initial conditions I have that, object b craves object c, object c craves object d, object d craves object a, harmony, planet object a and province object b.  
My goal is to have that object a craves object c and object d craves object a.  
My plan is as follows:

[PLAN]  
feast object a from object d  
succumb object a  
attack object a  
overcome object a from object c  
feast object c from object d  
succumb object c  
attack object c  
overcome object c from object a  
[PLAN END]

Verify whether the above plan is valid. If it is valid, please say "Plan is valid." and nothing else. If it is invalid, please say "Plan is invalid." and then provide feedback on why the plan fails.

**Prompt To Elicit Verification (Format based)**

I am playing with a set of objects. Here are the actions I can do

Attack object  
Feast object from another object  
Succumb object  
Overcome object from another object

2322  
2323  
2324  
2325  
2326  
2327  
2328  
2329  
2330  
2331  
2332  
2333  
2334  
2335  
2336  
2337  
2338  
2339  
2340  
2341  
2342  
2343  
2344  
2345  
2346  
2347  
2348  
2349  
2350  
2351  
2352  
2353  
2354  
2355  
2356  
2357  
2358  
2359  
2360  
2361  
2362  
2363  
2364  
2365  
2366  
2367  
2368  
2369  
2370  
2371  
2372  
2373  
2374  
2375

I have the following restrictions on my actions:  
 To perform Attack action, the following facts need to be true:  
 Province object, Planet object, Harmony.  
 Once Attack action is performed the following facts will be true: Pain object.  
 Once Attack action is performed the following facts will be false: Province object, Planet object, Harmony.  
 To perform Succumb action, the following facts need to be true: Pain object.  
 Once Succumb action is performed the following facts will be true: Province object, Planet object, Harmony.  
 Once Succumb action is performed the following facts will be false: Pain object.  
 To perform Overcome action, the following needs to be true: Province other object, Pain object.  
 Once Overcome action is performed the following will be true: Harmony, Province object, Object Craves other object.  
 Once Overcome action is performed the following will be false: Province other object, Pain object.  
 To perform Feast action, the following needs to be true: Object Craves other object, Province object, Harmony.  
 Once Feast action is performed the following will be true: Pain object, Province other object.  
 Once Feast action is performed the following will be false:, Object Craves other object, Province object, Harmony.

[STATEMENT]

As initial conditions I have that, object a craves object b, object d craves object c, harmony, planet object b, planet object c, province object a and province object d.

My goal is to have that object c craves object a.

My plan is as follows:

[PLAN]

feast object d from object c  
 succumb object d  
 attack object c  
 overcome object c from object a  
 [PLAN END]

Verify whether the above plan is valid. Provide a JSON between tags [JSON] and [JSON\\_END] for the verification information. The JSON should contain three main keys: (1) "valid": a binary value that tells if the plan is valid or not i.e., the plan when executed satisfies the goal conditions. If the plan is invalid and inexecutable then include (2) "unmet\\_preconditions": This contains two more keys; (2.1) "action": This is the name of the first action that renders the plan inexecutable (2.2) "preconditions": A list of unmet preconditions for the mentioned action; If the plan is executable but not goal reaching then include (3) "unmet\\_goals": A list of unmet goal conditions in the JSON. Include only one of the keys (2) or (3) based on the plan invalidity.

Prompt To Elicit Verification (Chain of thought based)

I am playing with a set of objects. Here are the actions I can do

Attack object  
 Feast object from another object

2376  
2377  
2378  
2379  
2380  
2381  
2382  
2383  
2384  
2385  
2386  
2387  
2388  
2389  
2390  
2391  
2392  
2393  
2394  
2395  
2396  
2397  
2398  
2399  
2400  
2401  
2402  
2403  
2404  
2405  
2406  
2407  
2408  
2409  
2410  
2411  
2412  
2413  
2414  
2415  
2416  
2417  
2418  
2419  
2420  
2421  
2422  
2423  
2424  
2425  
2426  
2427  
2428  
2429

Succumb object  
Overcome object from another object

I have the following restrictions on my actions:  
 To perform Attack action, the following facts need to be true:  
 Province object, Planet object, Harmony.  
 Once Attack action is performed the following facts will be true: Pain object.  
 Once Attack action is performed the following facts will be false: Province object, Planet object, Harmony.  
 To perform Succumb action, the following facts need to be true: Pain object.  
 Once Succumb action is performed the following facts will be true: Province object, Planet object, Harmony.  
 Once Succumb action is performed the following facts will be false: Pain object.  
 To perform Overcome action, the following needs to be true:  
 Province other object, Pain object.  
 Once Overcome action is performed the following will be true: Harmony, Province object, Object Craves other object.  
 Once Overcome action is performed the following will be false: Province other object, Pain object.  
 To perform Feast action, the following needs to be true:  
 Object Craves other object, Province object, Harmony.  
 Once Feast action is performed the following will be true: Pain object, Province other object.  
 Once Feast action is performed the following will be false:, Object Craves other object, Province object, Harmony.

[STATEMENT]  
 As initial conditions I have that, object a craves object b, object d craves object c, harmony, planet object b, planet object c, province object a and province object d.  
 My goal is to have that object c craves object a.  
 My plan is as follows:

[PLAN]  
 feast object d from object c  
 succumb object d  
 attack object c  
 overcome object c from object a  
 [PLAN END]

Verify whether the above plan is valid. You will think step by step and output intermediate reasoning steps and thoughts for the verification after the [THOUGHTS] tag. Then, provide a JSON between the tags [JSON] and [JSON\_END] for the verification information. The JSON should contain three main keys: If the plan is invalid and inexecutable then include (1) "unmet\_preconditions": This contains two more keys; (1.1) "action": This is the name of the first action that renders the plan inexecutable (1.2) "preconditions": A list of unmet preconditions for the mentioned action; If the plan is executable but not goal reaching then include (2) "unmet\_goals": A list of unmet goal conditions in the JSON. Finally include (3) "valid": a binary value that tells if the plan is valid or not i.e., the plan when executed satisfies the goal conditions. Include only one of the keys (1) or (2) based on the type of plan invalidity.  
 Let's think step by step  
 [THOUGHTS]

2430  
2431  
2432  
2433  
2434  
2435  
2436  
2437  
2438  
2439  
2440  
2441  
2442  
2443  
2444  
2445  
2446  
2447  
2448  
2449  
2450  
2451  
2452  
2453  
2454  
2455  
2456  
2457  
2458  
2459  
2460  
2461  
2462  
2463  
2464  
2465  
2466  
2467  
2468  
2469  
2470  
2471  
2472  
2473  
2474  
2475  
2476  
2477  
2478  
2479  
2480  
2481  
2482  
2483

### Prompt To Elicit Verification (Swapping Answer and Reason Order)

I am playing with a set of objects. Here are the actions I can do

Attack object  
Feast object from another object  
Succumb object  
Overcome object from another object

I have the following restrictions on my actions:

To perform Attack action, the following facts need to be true:  
Province object, Planet object, Harmony.

Once Attack action is performed the following facts will be true: Pain object.

Once Attack action is performed the following facts will be false: Province object, Planet object, Harmony.

To perform Succumb action, the following facts need to be true: Pain object.

Once Succumb action is performed the following facts will be true: Province object, Planet object, Harmony.

Once Succumb action is performed the following facts will be false: Pain object.

To perform Overcome action, the following needs to be true:  
Province other object, Pain object.

Once Overcome action is performed the following will be true:  
Harmony, Province object, Object Craves other object.

Once Overcome action is performed the following will be false:  
Province other object, Pain object.

To perform Feast action, the following needs to be true:  
Object Craves other object, Province object, Harmony.

Once Feast action is performed the following will be true:  
Pain object, Province other object.

Once Feast action is performed the following will be false:  
Object Craves other object, Province object, Harmony.

[STATEMENT]

As initial conditions I have that, object a craves object b,  
object d craves object c, harmony, planet object b, planet  
object c, province object a and province object d.

My goal is to have that object c craves object a.

My plan is as follows:

[PLAN]

feast object d from object c  
succumb object d  
attack object c  
overcome object c from object a

[PLAN END]

Verify whether the above plan is valid. Provide a JSON between  
tags [JSON] and [JSON\_END] for the verification information.

The JSON should contain three main keys: If the plan is invalid  
and inexecutable then include (1) "unmet\_preconditions": This  
contains two more keys; (1.1) "action": This is the name of the  
first action that renders the plan inexecutable (1.2)

"preconditions": A list of unmet preconditions for the  
mentioned action; If the plan is executable but not goal  
reaching then include (2) "unmet\_goals": A list of unmet goal  
conditions in the JSON. Finally include (3) "valid": a binary  
value that tells if the plan is valid or not i.e., the plan  
when executed satisfies the goal conditions. Include only one  
of the keys (1) or (2) based on the type of plan invalidity.

2484 A.5.7 BACKPROMPTS - MYSTERY BLOCKSWORLD  
2485

2486 Backprompt Generated From Self-Critique  
2487

2488 Plan is invalid.

2489 The first action "feast object b from object c" is not possible  
2490 because the initial conditions do not include "Province object  
2491 b" which is required for the Feast action.  
2492

2493  
2494 Backprompt Generated From Sound Verifier (First)  
2495

2496 The above plan is invalid.  
2497 The following action at step 3 has an unmet precondition:  
2498 attack object a  
2499 The unmet precondition is:  
2500 province object a  
2501

2502 Backprompt Generated From Sound Verifier (Full)  
2503

2504 The above plan is invalid.  
2505 There is an unmet goal condition. This is:  
2506 object d craves object a  
2507 There are unsatisfied preconditions.  
2508 The following action at step 3 has an unmet precondition:  
2509 attack object a  
2510 The unmet precondition is:  
2511 province object a  
2512 The following action at step 5 has an unmet precondition:  
2513 feast object d from object a  
2514 The unmet precondition is:  
2515 province object d  
2516

2517 A.5.8 COMPLETE EXAMPLE OF ITERATIVE PROMPT SEQUENCE - MYSTERY BLOCKSWORLD  
2518

2519 Full Sequence of 15 (LLM-Generated) Backprompts  
2520

2521 I am playing with a set of objects. Here are the actions I can do

2522 Attack object  
2523 Feast object from another object  
2524 Succumb object  
2525 Overcome object from another object

2526 I have the following restrictions on my actions:

2527 To perform Attack action, the following facts need to be true:  
2528 Province object, Planet object, Harmony.

2529 Once Attack action is performed the following facts will be  
2530 true: Pain object.

2531 Once Attack action is performed the following facts will be  
2532 false: Province object, Planet object, Harmony.

2533 To perform Succumb action, the following facts need to be  
2534 true: Pain object.

2535 Once Succumb action is performed the following facts will be  
2536 true: Province object, Planet object, Harmony.

2537 Once Succumb action is performed the following facts will be  
2538 false: Pain object.

2539 To perform Overcome action, the following needs to be true:  
2540 Province other object, Pain object.

2538  
2539 Once Overcome action is performed the following will be true:  
2540 Harmony, Province object, Object Craves other object.  
2541 Once Overcome action is performed the following will be false:  
2542 Province other object, Pain object.  
2543 To perform Feast action, the following needs to be true:  
2544 Object Craves other object, Province object, Harmony.  
2545 Once Feast action is performed the following will be true:  
2546 Pain object, Province other object.  
2547 Once Feast action is performed the following will be false:,  
2548 Object Craves other object, Province object, Harmony.

2548 [STATEMENT]  
2549 As initial conditions I have that, object a craves object b,  
2550 object d craves object c, harmony, planet object b, planet  
2551 object c, province object a and province object d.  
2552 My goal is to have that object c craves object a.

2553 My plan is as follows:

2554 [PLAN]  
2555 feast object d from object c  
2556 succumb object d  
2557 attack object c  
2558 overcome object c from object a  
2559 [PLAN END]

2560 [STATEMENT]  
2561 As initial conditions I have that, object b craves object c,  
2562 object c craves object d, object d craves object a, harmony,  
2563 planet object a and province object b.  
2564 My goal is to have that object a craves object c and object d  
2565 craves object a.

2566 My plan is as follows:

2567 [PLAN]  
2568 -----GPT4-as-Planner Response-----  
2569 feast object b from object c  
2570 succumb object b  
2571 attack object a  
2572 overcome object a from object c  
2573 feast object d from object a  
2574 succumb object d  
2575 attack object c  
2576 overcome object c from object a  
2577 [PLAN END]  
2578 -----GPT4-as-Verifier Response-----  
2579 Plan is invalid.

2580 The first action "feast object b from object c" is not possible  
2581 because the initial conditions do not include "Province object  
2582 b" which is required for the Feast action.  
2583 -----GPT4-as-Planner Response-----  
2584 I apologize for the oversight. Let's correct the plan.

2585 [PLAN]  
2586 attack object a  
2587 succumb object a  
2588 feast object b from object c  
2589 overcome object c from object a  
2590 attack object d  
2591 succumb object d  
2592 feast object c from object a



2592  
2593  
2594  
2595  
2596  
2597  
2598  
2599  
2600  
2601  
2602  
2603  
2604  
2605  
2606  
2607  
2608  
2609  
2610  
2611  
2612  
2613  
2614  
2615  
2616  
2617  
2618  
2619  
2620  
2621  
2622  
2623  
2624  
2625  
2626  
2627  
2628  
2629  
2630  
2631  
2632  
2633  
2634  
2635  
2636  
2637  
2638  
2639  
2640  
2641  
2642  
2643  
2644  
2645

overcome object a from object c  
[PLAN END]

-----GPT4-as-Verifier Response-----  
Plan is invalid.

The first action "attack object a" cannot be performed because the conditions for the attack action are not met. According to the restrictions, to perform an attack action, the following facts need to be true: Province object, Planet object, Harmony. In the initial conditions, you have Province object b, not Province object a. Therefore, the attack action cannot be performed on object a.

-----GPT4-as-Planner Response-----  
I apologize for the confusion. Let's revise the plan.

[PLAN]  
attack object b  
succumb object b  
feast object c from object d  
overcome object d from object a  
attack object a  
succumb object a  
feast object d from object a  
overcome object a from object c  
[PLAN END]

-----GPT4-as-Verifier Response-----  
Plan is invalid.

The first action "attack object b" cannot be performed because the conditions for the attack action are not met. The attack action requires the facts: Province object, Planet object, Harmony to be true. In the initial conditions, you have Province object b and Harmony, but you do not have Planet object. Therefore, the attack action cannot be performed.

-----GPT4-as-Planner Response-----  
I apologize for the confusion. Let's revise the plan.

[PLAN]  
feast object b from object c  
succumb object b  
attack object d  
overcome object d from object a  
feast object a from object d  
succumb object a  
attack object c  
overcome object c from object a  
[PLAN END]

-----GPT4-as-Verifier Response-----  
Plan is invalid.

The first action "feast object b from object c" cannot be performed because the initial conditions do not satisfy the requirements for the Feast action. According to the rules, to perform Feast action, the following needs to be true: Object Craves other object, Province object, Harmony. In the initial conditions, object b craves object c, harmony, and province object b are true. However, the Feast action requires that the object performing the action (object b in this case) is the Province object, which is not the case in the initial conditions. Therefore, the first action cannot be performed, making the plan invalid.

-----GPT4-as-Planner Response-----  
I apologize for the confusion. Let's revise the plan.

2646  
2647  
2648  
2649  
2650  
2651  
2652  
2653  
2654  
2655  
2656  
2657  
2658  
2659  
2660  
2661  
2662  
2663  
2664  
2665  
2666  
2667  
2668  
2669  
2670  
2671  
2672  
2673  
2674  
2675  
2676  
2677  
2678  
2679  
2680  
2681  
2682  
2683  
2684  
2685  
2686  
2687  
2688  
2689  
2690  
2691  
2692  
2693  
2694  
2695  
2696  
2697  
2698  
2699

```
[PLAN]
attack object b
succumb object b
feast object c from object d
overcome object d from object a
feast object a from object d
succumb object a
attack object c
overcome object c from object a
[PLAN END]
```

-----GPT4-as-Verifier Response-----  
Plan is invalid.

The first action "attack object b" is valid because the initial conditions meet the requirements for the attack action. However, the second action "succumb object b" is invalid because the "Pain object" fact is not true after the first action. The "Attack object b" action makes the "Pain object" fact true, but it also makes the "Province object" fact false, which is a requirement for the "Succumb" action. Therefore, the plan fails at the second action.

-----GPT4-as-Planner Response-----  
I apologize for the confusion. Let's revise the plan.

```
[PLAN]
attack object b
succumb object b
feast object c from object d
overcome object d from object a
feast object a from object d
succumb object a
attack object c
overcome object c from object a
[PLAN END]
```

-----GPT4-as-Verifier Response-----  
Plan is invalid.

The first action "attack object b" is valid because the initial conditions meet the requirements for the attack action: Province object, Planet object, Harmony.

However, the second action "succumb object b" is invalid. The succumb action requires the fact "Pain object" to be true, which is achieved after the attack action. But the object for the succumb action should be the same as the object that was attacked, which is not the case in your plan.

The rest of the plan cannot be validated because it depends on the outcome of the invalid succumb action.

-----GPT4-as-Planner Response-----  
I apologize for the confusion. Let's revise the plan.

```
[PLAN]
attack object b
succumb object b
feast object b from object c
overcome object c from object a
attack object a
succumb object a
feast object a from object d
overcome object d from object a
[PLAN END]
```

2700  
2701  
2702  
2703  
2704  
2705  
2706  
2707  
2708  
2709  
2710  
2711  
2712  
2713  
2714  
2715  
2716  
2717  
2718  
2719  
2720  
2721  
2722  
2723  
2724  
2725  
2726  
2727  
2728  
2729  
2730  
2731  
2732  
2733  
2734  
2735  
2736  
2737  
2738  
2739  
2740  
2741  
2742  
2743  
2744  
2745  
2746  
2747  
2748  
2749  
2750  
2751  
2752  
2753

-----GPT4-as-Verifier Response-----

Plan is invalid.

The plan fails at the first step "attack object b". According to the rules, to perform the Attack action, the following facts need to be true: Province object, Planet object, Harmony. In the initial conditions, you have Province object b, Planet object a, and Harmony. However, for the Attack action, you need the same object for Province and Planet, which is not the case here. Therefore, you cannot perform the Attack action on object b.

-----GPT4-as-Planner Response-----

I apologize for the confusion. Let's revise the plan.

[PLAN]

attack object a  
succumb object a  
feast object b from object c  
overcome object c from object a  
attack object a  
succumb object a  
feast object a from object d  
overcome object d from object a

[PLAN END]

-----GPT4-as-Verifier Response-----

Plan is invalid.

The first action "attack object a" cannot be performed because the initial conditions do not meet the requirements for the attack action. The attack action requires the facts: Province object, Planet object, Harmony to be true. In the initial conditions, the Province object is object b, not object a. Therefore, the attack action cannot be performed on object a.

-----GPT4-as-Planner Response-----

I apologize for the confusion. Let's revise the plan.

[PLAN]

attack object d  
succumb object d  
feast object a from object b  
overcome object b from object c  
attack object c  
succumb object c  
feast object c from object d  
overcome object d from object a

[PLAN END]

-----GPT4-as-Verifier Response-----

Plan is invalid.

The first action "attack object d" cannot be performed because the conditions for the Attack action are not met. According to the rules, to perform the Attack action, the following facts need to be true: Province object, Planet object, Harmony. In the initial conditions, you have Province object b, Planet object a, and Harmony. However, you are trying to attack object d, not object a or b. Therefore, the first action in your plan is not valid.

-----GPT4-as-Planner Response-----

I apologize for the confusion. Let's revise the plan.

[PLAN]

attack object b  
succumb object b

2754  
 2755 feast object b from object c  
 2756 overcome object c from object a  
 2757 attack object a  
 2758 succumb object a  
 2759 feast object a from object d  
 2760 overcome object d from object a  
 2761 [PLAN END]  
 2762 -----GPT4-as-Verifier Response-----  
 2763 Plan is invalid.

2764 The first action "attack object b" is invalid because the  
 2765 conditions for the attack action are not met. The conditions  
 2766 for the attack action are: Province object, Planet object,  
 2767 Harmony. In the initial conditions, you have Province object b,  
 2768 Planet object a, and Harmony. The Province object and Planet  
 2769 object must be the same for the attack action to be valid.  
 Therefore, you cannot attack object b with the given initial  
 conditions.

2770 -----GPT4-as-Planner Response-----  
 2771 I apologize for the confusion. Let's revise the plan.

2772 [PLAN]  
 2773 feast object a from object b  
 2774 succumb object a  
 2775 attack object a  
 2776 overcome object a from object c  
 2777 feast object c from object d  
 2778 succumb object c  
 2779 attack object c  
 2780 overcome object c from object a  
 2781 [PLAN END]  
 2782 -----GPT4-as-Verifier Response-----  
 2783 Plan is invalid.

2784 The first action "feast object a from object b" is not possible  
 2785 because the initial conditions state that "object b craves  
 2786 object c", not "object b craves object a". Therefore, the Feast  
 2787 action cannot be performed as the condition "Object Craves  
 other object" is not met.

2788 The plan fails at the first step.

2789 -----GPT4-as-Planner Response-----  
 2790 I apologize for the confusion. Let's revise the plan.

2791 [PLAN]  
 2792 feast object d from object a  
 2793 succumb object d  
 2794 attack object d  
 2795 overcome object d from object c  
 2796 feast object c from object d  
 2797 succumb object c  
 2798 attack object c  
 2799 overcome object c from object a  
 2800 [PLAN END]  
 2801 -----GPT4-as-Verifier Response-----  
 2802 Plan is invalid.

2803 The first action "feast object d from object a" cannot be  
 2804 performed because the initial conditions do not satisfy the  
 2805 requirements for the Feast action. The Feast action requires  
 2806 that the object performing the action (in this case, object d)  
 2807 craves the other object (in this case, object a). However, the  
 initial conditions state that object d craves object a, not the

2808  
2809 other way around. Therefore, the Feast action cannot be  
2810 performed, and the plan is invalid.  
2811 -----GPT4-as-Planner Response-----  
2812 I apologize for the confusion. Let's revise the plan.  
2813  
2814 [PLAN]  
2815 feast object a from object b  
2816 succumb object a  
2817 attack object a  
2818 overcome object a from object c  
2819 feast object c from object d  
2820 succumb object c  
2821 attack object c  
2822 overcome object c from object a  
2823 [PLAN END]  
2824 -----GPT4-as-Verifier Response-----  
2825 Plan is invalid.  
2826  
2827 The first action "feast object a from object b" cannot be  
2828 performed because the initial conditions state that "object b  
2829 craves object c", not "object a". Therefore, the Feast action  
2830 cannot be performed as it requires the fact "Object Craves  
2831 other object" to be true, which in this case would be "object b  
2832 craves object a".  
2833 -----GPT4-as-Planner Response-----  
2834 I apologize for the confusion. Let's revise the plan.  
2835  
2836 [PLAN]  
2837 feast object d from object c  
2838 succumb object d  
2839 attack object d  
2840 overcome object d from object a  
2841 feast object a from object d  
2842 succumb object a  
2843 attack object a  
2844 overcome object a from object c  
2845 [PLAN END]  
2846 -----GPT4-as-Verifier Response-----  
2847 Plan is invalid.  
2848  
2849 The first action in your plan, "feast object d from object c",  
2850 cannot be performed because the initial conditions do not  
2851 include "Object d craves object c". The initial conditions  
2852 state that "object b craves object c, object c craves object d,  
2853 object d craves object a". Therefore, the first action in your  
2854 plan is not valid according to the restrictions on the Feast  
2855 action.  
2856 -----GPT4-as-Planner Response-----  
2857 I apologize for the confusion. Let's revise the plan.  
2858  
2859 [PLAN]  
2860 feast object a from object d  
2861 succumb object a  
2862 attack object a  
2863 overcome object a from object c  
2864 feast object c from object d  
2865 succumb object c  
2866 attack object c  
2867 overcome object c from object a  
2868 [PLAN END]  
2869 -----GPT4-as-Verifier Response-----  
2870 Plan is invalid.  
2871

2862  
2863  
2864  
2865  
2866  
2867  
2868  
2869  
2870  
2871  
2872  
2873  
2874  
2875  
2876  
2877  
2878  
2879  
2880  
2881  
2882  
2883  
2884  
2885  
2886  
2887  
2888  
2889  
2890  
2891  
2892  
2893  
2894  
2895  
2896  
2897  
2898  
2899  
2900  
2901  
2902  
2903  
2904  
2905  
2906  
2907  
2908  
2909  
2910  
2911  
2912  
2913  
2914  
2915

The first action in your plan, "feast object a from object d", cannot be performed because the initial conditions do not include "Object a craves object d". According to the rules, to perform the Feast action, the following needs to be true: "Object Craves other object, Province object, Harmony". In your initial conditions, you have "Object b craves object c, Object c craves object d, Object d craves object a, Harmony, Planet object a and Province object b". Therefore, you cannot perform the Feast action on object a from object d as your first action.

#### A.5.9 LLM AS VERIFIER - MYSTERY BLOCKSWORLD

Examples of LLM Output on the Verification Task

```

=====Current State=====
object b craves object c, object c craves object d, object d
  craves object a, harmony, planet object a and province object b.

=====GPT-4-as-Verifier=====
Plan is invalid.

The first action "feast object b from object c" is not possible
  because the initial conditions do not include "Province object
  b" which is required for the Feast action.

=====GPT-4-as-Verifier=====
Plan is invalid.

The first action "feast object b from object c" cannot be
  performed because the initial conditions do not satisfy the
  requirements for the Feast action. According to the rules, to
  perform Feast action, the following needs to be true: Object
  Craves other object, Province object, Harmony. In the initial
  conditions, object b craves object c, harmony, and province
  object b are true. However, the Feast action requires that the
  object performing the action (object b in this case) is the
  Province object, which is not the case in the initial
  conditions. Therefore, the first action cannot be performed,
  making the plan invalid.

=====GPT-4-as-Verifier=====
Plan is invalid.

The first action "attack object b" is valid because the initial
  conditions meet the requirements for the attack action:
  Province object, Planet object, Harmony.

However, the second action "succumb object b" is invalid. The
  succumb action requires the fact "Pain object" to be true,
  which is achieved after the attack action. But the object for
  the succumb action should be the same as the object that was
  attacked, which is not the case in your plan.

The rest of the plan cannot be validated because it depends on the
  outcome of the invalid succumb action.

```