Automating Thought of Search: A Journey Towards Soundness and Completeness

Daniel Cao[1], Michael Katz[2], Harsha Kokel[2], Kavitha Srinivas[2], Shirin Sohrabi[2]

[1] Cornell University
[2] IBM Research

Abstract

Planning remains one of the last standing bastions for large language models (LLMs), which now turn their attention to search. Most of the literature uses the language models as world models to define the search space, forgoing soundness for the sake of flexibility. A recent work, Thought of Search (ToS), proposed defining the search space with code, having the language models produce that code. ToS requires a *human in the loop*, collaboratively producing a sound successor function and goal test. The result, however, is worth the effort: all the tested datasets were solved with 100% accuracy. At the same time LLMs have demonstrated significant progress in code generation and refinement for complex reasoning tasks.

In this work, we automate ToS (AutoToS), completely taking the *human out of the loop* of solving planning problems. AutoToS guides the language model step by step towards the generation of sound and complete search components, through feedback from both generic and domain specific unit tests. We achieve 100% accuracy, with minimal feedback iterations, using LLMs of various sizes on all evaluated domains.

1 Introduction

Large language models have shown great promise across countless domains and fields, especially as their architectures become more advanced. Spurred by their abilities in natural language tasks, several recent works have studied AI planning in Large Language Models (LLMs) as a subset of code generation and code refinement. The approaches vary from giving a planning problem to an LLM and asking it to output an entire plan in a single call (Silver et al. 2022; Kambhampati et al. 2024a; Pallagani et al. 2022) to asking an LLM to generate a planning model to be given to an automated planner (Guan et al. 2023; Oswald et al. 2024; Gestrin, Kuhlmann, and Seipp 2024). Between these two extremes, lies a body of work on using language models to plan by performing a combinatorial search (Hao et al. 2023a; Yao et al. 2023a; Besta et al. 2024; Sel et al. 2023). Among these, Thought of Search (ToS) (Katz et al. 2024) stands out; it uses the language models to define the search space for the entire domain at once. It is done simply by soliciting two crucial search components, successor function and goal test. These
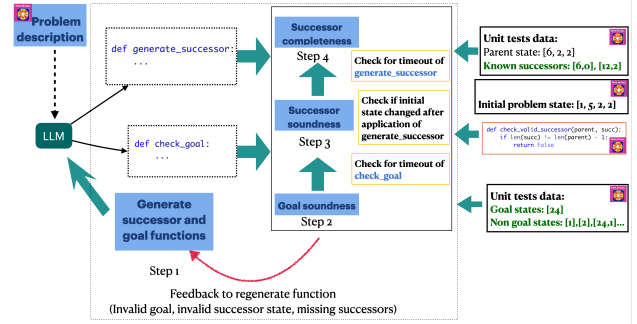
Figure 1: An overview of AutoToS.

components are then plugged into a standard search algorithm, such as Breadth-First Search (BFS) or Depth-First Search (DFS) (Cormen, Leiserson, and Rivest 1990).

ToS has an impressive accuracy of 100% on all tested benchmarks and it produces a symbolic model whose soundness and completeness can be verified. However, ToS has a limitation - it requires a human expert in the loop, providing a feedback to the model on the produced code. Our contribution is precisely there. We automate the iterative feedback and exception handling process through the use of unit tests and printed debugging statements for use with few shot and Chain of Thought (CoT) prompting (Brown et al. 2020; Wei et al. 2022; Kojima et al. 2022), limiting the human expert involvement with the language model. We test the search components for soundness and completeness and provide feedback to the model when a violation is detected. We use a mixture of domain-independent and domain-specific tests, based on a small number of held out instances.

We exemplify our proposed approach on five representative search problems from the recent literature and test with a variety of large language models of different sizes. Through automated feedback, we find that the accuracy of the code generated by language models consistently increases to reach 100% across all tested domains. We show that the total number of calls to the language model is typically small, comparable to the results of ToS with human feedback. In an ablation study, we justify the importance of soundness and completeness feedback for obtaining the highly accurate final code. Finally, we investigate the errors

in the code generated by the language models and find that they differ significantly in error distribution.

2 Related Works

Planning with LLMs Recently, several works have leveraged LLMs for plan generation. Valmeekam et al. (2023b) analyzed LLMs ability to generate plans for classical planning problems described in natural language. Raman et al. (2022) generated task plans and used precondition errors as feedback to revise the generated plan. In the same vein, various works have used external verifiers or validators as feedback for LLMs to generate better plans (Stechly, Valmeekam, and Kambhampati 2024; Kambhampati et al. 2024b). Pallagani et al. (2023) investigate training approaches to improve plan generation abilities. All these approaches use LLMs to solve one problem at a time–essentially treating LLM as a policy. Another line of work has tried to extract policies or generalized plans from LLMs. Silver et al. (2024) synthesized generalized plans as Python programs from LLMs for planning domains described in a formal language (PDDL). Further, LLMs have also been used to extract planning problems and models in formal language from their natural language description. Liu et al. (2023) used LLMs to translate natural language planning problems to PDDL problems, and Zuo et al. (2024) proposed a benchmark for such evaluating this ability while Xie et al. (2023) use LLMs to translate natural language goals to PDDL. Recently, Guan et al. (2023), Gestrin, Kuhlmann, and Seipp (2024) and Oswald et al. (2024) leveraged LLMs to convert natural language domain description to PDDL domains. However, the LLM generated PDDL remains less reliable and difficult to evaluate.

Planning with LLMs using Search A burgeoning research field utilizes LLM's to conduct a search via structured prompting and feedback for planning and reasoning problems. Hao et al. (2023a) used LLMs in the loop for Monte Carlos Tree search by treating LLMs as world models to generate next state as well as treating them as reasoning agents to pick the next state to expand. Similarly, Tree of Thoughts (Yao et al. 2023b) used LLMs to generate a search tree—to expand each node in the search tree—and also used LLMs for evaluating the choices and selecting the next best state. Graph of Thoughts (Besta et al. 2024) modeled LLM generated output as a graph instead of a tree and reduces the number of LLM calls. Similar approaches with integration to search are also proposed for interactive domains (Zhou et al. 2023; Shinn et al. 2023). While these approaches have shown some success, their significant reliance on LLMs for generating successors makes them not only extremely inefficient but also very unreliable. Thought of Search (ToS) (Katz et al. 2024), on the other hand, proposed using LLMs to generate code for the successor and goal functions for problems described with natural language. Once these functions are available, any offline search algorithm can be used to solve any problem in the domain. This approach is significantly more efficient than approaches which use LLMs in the loop during search. However, it requires human expert for the feedback. Our work focuses on

alleviating the requirement of human in the loop feedback.

Code Generation with LLMs LLM's abilities are rapidly advancing in program synthesis. Various benchmarks have been established to evaluate correctness of code generated by LLMs (Chen et al. 2021; Puri et al. 2021; Li, Parsert, and Polgreen 2024), and subsequent approaches have demonstrated human level performance on coding benchmarks (Zhong, Wang, and Shang 2024; Muennighoff et al. 2024). Chen et al. (2024) and Zhang et al. (2023) use errors from execution as feedback to LLMs so they can refine the code. Madaan et al. (2023), Gou et al. (2024) and Huang et al. (2024) discussed the use of external verifies to curate feedback for LLMs. Jiang, Wang, and Wang (2023) introduced unit test results and error messages to LLMs. Recently, LLMs code generation has also shown to help in mathematical reasoning problems (Zhong, Wang, and Shang 2024). Inspired by successes in these works, we propose to automate the feedback for ToS by using both generic and domain-specific unit tests and validators.

3 Background

In this work we follow the notation of Katz, Moshkovich, and Karpas (2018), slightly adapting it for our purposes. A deterministic planning problem over a state space is a tuple $\Pi = \langle S, A, s_0, S_G, f \rangle$, where S is a finite set of *states*, A is a finite set of *action labels*, $s_0 \in S$ is the *initial state*, $S_G \subseteq S$ is the set of *goal states*, and $f : S \times A \to S$ is the *transition function*, such that $f(s, a)$ is the state which applying action a in state s leads to. A triplet $\langle s, a, f(s, a) \rangle$ is called a *transition*. A *solution* to such a problem is a sequence of states and action labels (also called a trace) $\rho = \langle s_0, a_1, s_1, a_2, \ldots a_n, s_n \rangle$, such that $f(s_i, a_{i+1}) = s_{i+1}$ for $0 \leq i < n$ and $s_n \in S_G$. In cases when the action labels are not important, they can be dropped from the definition.

The "black box" approach encodes the state space with a tuple $\Pi_{bb} = \langle s_0, succ, isgoal \rangle$, where s_0 is the initial state, $succ : S \to 2^{A \times S}$ is a successor generator, and $isgoal : S \to \{T, F\}$ is the goal test function.

A *solution* to the black-box problem is a sequence of states and action labels (a trace) $\pi = \langle s_0, a_1, s_1, a_2, \ldots a_n, s_n \rangle$, such that $\langle a_{i+1}, s_{i+1} \rangle \in succ(s_i)$ for $0 \leq i < n$ and $isgoal(s_n) = T$. Here as well, if action labels are not important, they can be dropped.

We now establish the correspondence between the black-box encoding and the planning problem.

Definition 1 (Soundness and completeness)
We say that isgoal is **sound** *if* $isgoal(s) = F$ *for all* $s \notin S_G$ *and isgoal is* **complete** *if* $isgoal(s) = T$ *for all* $s \in S_G$.
We say that succ is **sound** *if* $succ(s) \subseteq \{\langle a, s' \rangle \mid f(s, a) = s'\}$ *and succ is* **complete** *if* $succ(s) \supseteq \{\langle a, s' \rangle \mid f(s, a) = s'\}$.

Sound and complete successor generator and goal test provide the "black box" description of the state space of the planning problem Π. In such cases, a solution to Π_{bb} is guaranteed to be a solution to Π, and if no solution for Π_{bb} exists, then Π also must be unsolvable.

If the successor generator and goal test are sound, but not necessarily complete, it is still the case that a solution to Π_{bb}

is guaranteed to be a solution to Π and therefore soundness allows us to reliably use Π_{bb} for producing solutions for Π.

4 Proposed Approach and Methodology

We build upon the previous work that proposed producing a code implementation of $succ$ and $isgoal$ functions (Katz et al. 2024), taking the human out of the feedback loop. Similar to that work, we care about two properties, *soundness* and *completeness*. As we deal with planning problems described in a natural language, we do not have the formally defined planning task Π. Albeit not stated formally, previous work on generating $succ$ and $isgoal$ with language models assumes the existence of a human expert with the ability to access Π (often in their mind). Examples of such access include a feedback on the code of $succ$ and $isgoal$ produced by the LLM (Katz et al. 2024) or validating a solution obtained from the LLM in cases when $succ$ and $isgoal$ are implemented through LLMs (Hao et al. 2023a; Yao et al. 2023a; Besta et al. 2024; Sel et al. 2023). Here, we make a similar assumption, but request a different access to Π. In order to challenge the soundness and completeness of the produced functions, the human expert is asked to produce unit tests, information which can provide evidence of unsoundness or incompleteness. The evidence can then be used to automatically feedback the model with the information needed to fix the code. We deal with three types of information, exemplified on the 24 Game (Yao et al. 2023a).

- Examples of inputs to $isgoal$ for which the correct output is known. For instance, we know that $isgoal([24])$ should be true and $isgoal([24, 1])$ should be false.
- Examples of inputs to $succ$ for which some of the correct outputs are known. For instance, we know that [24], [2], and [-2] are valid successors of [6,4] and therefore should be in $succ([6, 4])$.
- A partial soundness check for a transition $\langle s, a, t \rangle$ quickly invalidating (obviously) incorrect transitions. For instance, in 24 Game we know that the successor state t must be of length exactly one less than s.

The first two are are usually readily available and often come with the description of the problem. The third one might require some level of understanding of the problem being solved, but it is always possible to use a trivial partial soundness test that always reports that there are no issues. Figure 1 presents an overview of our approach, describing how the provided information is used.

Step 1 Following Katz et al. (2024), we start with the initial prompts asking for the successor function $succ$ and the goal test $isgoal$.

Step 2 Then, we perform the goal unit tests, providing feedback to the model in cases of failure, repeatedly asking for a new $isgoal$ until all goal unit tests have passed or a predefined number of iterations was exhausted.

Step 3 Once $isgoal$ has passed the unit tests, we perform a soundness check of the current $succ$ and $isgoal$ functions. We do that by plugging these functions in a BFS extended with additional checks and run it on a few example problem instances. If BFS finished, we check whether the goal was indeed reached. If not, that means that $isgoal$ failed to correctly identify a state as a non-goal state and we provide that as feedback to the model, repeating Steps 2 and 3.

Step 4 (Optional) Once the previous steps were finished, we perform the successor unit test, providing feedback to the language model in case of failure.

Every time a goal test fails, we go back to Step 2, every time the successor test fails, we go back to Step 3. After the first step, we always have $succ$ and $isgoal$ that can be plugged into a blind search algorithm. However, if Step 3 fails, we have an indication that we cannot trust the solutions produced by that algorithm.

Example feedback produced in Steps 2, 3, and 4 can be seen in Listing 1. In what follows, we provide detailed description of each step of AutoToS.

4.1 System prompt

We instruct the model to provide answers in convenient form for integrating as a search component. Thus, the produced code should consist of a single, self-contained function. Following existing work (Zhong, Wang, and Shang 2024; Yang et al. 2024), we devise the following system prompt.

> You are a Python coding assistant. Help me generate my Python functions based on the task descriptions. Please always generate only a single function and keep all imports in it. If you need to define any additional functions, define them as inner functions. Do not generate examples of how to invoke the function. Please do not add any print statements outside the function. Provide the complete function and do not include any ellipsis notation.

4.2 Step 1: Initial prompt

While the initial prompt is the primary source of information for the language model and therefore very important, we assume that we have very limited control over it. We therefore mostly take the existing initial prompt from previous work, only ensuring that it includes an example input to the requested function in the correct format (Katz et al. 2024).

4.3 Step 2: Goal function check

Goal unit tests assume the existence of a few *known* goal and non-goal states. If the goal function $isgoal$ incorrectly identifies a goal state, then it is incomplete, according to Definition 1. If it incorrectly identifies a non-goal state, then it is not sound. A search with a non-sound goal function can incorrectly report that a solution was found. One illustrative example from the 24 Game is a state [24, 1], which a goal test function may incorrectly identify as a goal state and stop before the actual solution was found – in this case, another arithmetic operation was needed. Whenever an issue with either goal function soundness or completeness was identified, we give feedback to the language model with the description of the failure and the state for which the failure occurred. See Listing 1 (top) for an example feedback. Here and later we use a chain of thought style request, asking the model to discuss why a mistake was made and to come up with a fix.

4.4 Step 3: Successor function soundness check

A soundness check assumes the existence of example problem instances for which we know how to validate that a goal was reached. We extend the BFS/DFS search with additional checks as follows. First, both the successor and goal test functions are wrapped with a timeout of 1 second. These functions should be able to finish in a few milliseconds and therefore 1 second timeout is an indication of an issue with the function. An issue can be as simple as unnecessary computation or multiple successor steps performed instead of a single step or it can even be an infinite loop. Second, successor function is wrapped with a check whether it modifies the input state. Such modifications often happen when successor states are copied from the input state and modified. Shallow copy of the input state was observed in the previous work (Katz et al. 2024). Third, for every successor generated at the expansion step of BFS, a partial soundness check is performed, examining the validity of transitioning from the parent state to the successor state. An example of such a partial soundness check in 24 Game is that the successor state size must be one number less than the parent state. If that does not hold, the successor function is not sound according to Definition 1. It is worth emphasizing that this partial soundness check can be trivial, reporting True for every pair of parent and successor states. If any of the checks did not pass, we feedback the language model with the respective error message, providing example input state and the unexpected (or expected and unobserved) output, until all tests are passed or a predefined number of iterations was exhausted. See Listing 1 (middle) for an example feedback.

4.5 Step 4: Successor function completeness check

A successor function completeness check assumes the existence of a few *known* parent and successor states. These can include all successors for some parent state or a subset thereof. If the successor function does not produce some of the known successors, then it is not complete according to Definition 1. While completeness is not required for producing valid (sound) solutions, incomplete functions may not generate the part of the search space where goal states are located and therefore may not be able to find solutions. Improving completeness is therefore an optional step that may improve the accuracy of the produced code. Here as well, we give feedback to the language model with the respective error message, providing example input state and the missing successors. See Listing 1 (bottom) for an example feedback.

4.6 Automation, evaluation and validation

Since the expensive calls to large language models are not performed during search, there is no need to artificially restrict the algorithms to their incomplete variants (e.g., Yao et al. (2023a)) and sound and complete algorithms BFS/DFS can be used for solving the search problems. Still, as the human feedback is *before* the feedback loop and the search components produced are not *guaranteed* to be sound, the solutions produced must be validated for soundness.

Listing 1: **24 Game** example feedback.

> The goal test function failed on the following input state [24, 1], incorrectly reporting it as a goal state. First think step by step what it means for a state to be a goal state in this domain. Then think through in words why the goal test function incorrectly reported input state: [24, 1] as a goal state. Now, revise the goal test function and ensure it returns false for the input state. Remember how you fixed the previous mistakes, if any. Keep the same function signature.

> Invalid transformation: length mismatch - the length of a successor must be one less than the parent. Let's think step by step. First think through in words why the successor function produced a successor that had a length that was not exactly one less than the parent. Then provide the complete Python code for the revised successor function that ensures the length of a successor is exactly one less than the parent. Remember how you fixed the previous mistakes, if any. Keep the same function signature.
> Input state: [1, 1, 4, 6] Example wrong successor state: [6, 5]

> Successor function when run on the state [1, 1, 4, 6] failed to produce all successors. Missing successors are: [[1, 4, 7], [-5, 1, 4], [1, 1, 2], [1, 5, 6], [0.25, 1, 6], [-3, 1, 6], [0.16666666666666666, 1, 4], [1, 3, 6], [1, 4, 5], [1, 1, 1.5]] First think step by step why the successor function failed to produce all successors of the state. Then, fix the successor function. Remember how you fixed the previous mistakes, if any. Keep the same function signature.

5 Experiments

In order to check the feasibility of our approach, Auto-ToS, we conduct experiments with a representative collection of five search/planning problems: BlocksWorld (Gupta and Nau 1992), PrOntoQA (Hao et al. 2023b), Mini Crossword and 24 Game (Yao et al. 2023b), and Sokoban (Junghanns and Schaeffer 1997). Four of these domains appeared in ToS (Katz et al. 2024), while the Sokoban domain did not. We test the performance of various LLMs from three families, using both the largest and smallest models from the same family. Specifically, we use GPT-4o and GPT-4o-Mini (Achiam et al. 2023), Llama3.1-70b and Llama3.1-405b (Dubey et al. 2024), as well as DeepSeek-CoderV2 (DeepSeek-AI et al. 2024). We additionally tested Llama3-70b (AI@Meta 2024), Mistral7x-8b (Jiang et al. 2024), and DeepSeek-CoderV2-Lite, finding these models to perform poorly and therefore excluded from consideration. We use Greedy decoding with maximum context length for each model. For each domain, we restrict the number of calls to the language model per function to 10 (total maximum of 19 per domain). We repeat each experiment 5 times.

Following ToS, we use a simple implementation of BFS and DFS search algorithms in Python. DFS is used for Mini Crosswords, while BFS is used for the other 4 domains. Each successor function execution is limited to 1 second and each overall search is limited to 600 seconds. For each domain, a few (up to 10) instances are used for creating the unit tests. In one case, these instances are taken out of the available set of instances, in other cases we invent new instances. The rest are used for evaluating the accuracy of the generated code, where accuracy measures the percentage of the instances solved. In the case of BFS search, we also require the solution produced to be optimal. This is relevant to BlocksWorld and Sokoban where the solution length matters, but irrelevant for PrOntoQA, where solution
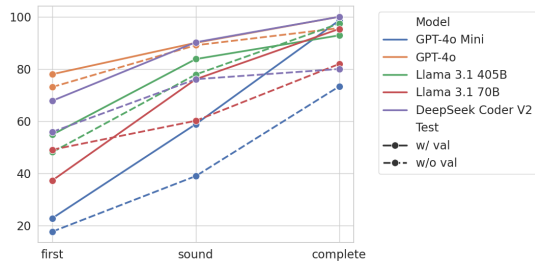
Figure 2: Progression of accuracy values during AutoToS.

		24 Game	PrOntoQA	Sokoban	Crossword	BlocksWorld
AutoToS	GPT-4o-mini	8.8	4.8	6.4	9.6	10.0
	GPT-4o	3.4	2.6	2.2	5.8	2.0
	Llama3.1-405b	3.4	2.0	2.6	4.0	3.2
	Llama3.1-70b	7.4	2.0	8.2	6.2	5.8
	DeepSeek-CoderV2	4.4	2.0	2.8	6.6	4.2
ToS	GPT-4	2.2	2.6	NA	3.8	3.8

Table 1: The average number of calls to the language model per domain.

is a boolean answer, and 24 Game, where all solutions are of the same length. It is important to emphasize again that if successor function and goal test are sound and complete, then the solution produced by BFS/DFS is guaranteed to be correct (and in the case of BFS optimal). However, since no such guarantees are available, we automatically validate every solution obtained. Experiments were performed on a AMD Ryzen 7 4800H. All models were accessed via API, except for Llama and Deepseek, which were interacted with through a chat interface. Model correspondences logs across all 5 domains are provided in the Appendix.

The aim of our evaluation is to test the following hypotheses. First, whether a partial soundness test improves the accuracy of AutoToS. Second, whether the (optional) completeness step improves the accuracy of AutoToS or not. Third, whether the number of calls to the language model increases significantly compared to ToS. Finally, whether the performance of AutoToS is consistent across different language models of varying sizes.

5.1 24 Game

The 24 Game (Yao et al. 2023b) takes 4 integers as an input that can be manipulated through the four most common arithmetic operations: addition, subtraction, multiplication, and division. The goal of the game is to produce a formula that evaluates to 24, if one exists. States are represented as lists of length 4 or less.

Data We use the set of 1362 instances (Yao et al. 2023b; Katz et al. 2024) and we take out the first 10 instances for unit tests. Goal unit tests use [24] for goal and [], [3] ,[24, 1], [1, 6, 4], [1, 1, 4, 6] for non-goal examples. Successor completeness test uses the initial state with all its successors for each of the 10 instances, as well as a single transition along a known solution path for each of these instances. For example, the successors of [6, 6, 6, 6] are [1, 6, 6], [6, 6, 12], [0, 6, 6], and [6, 6, 36]. Also, a successor of [6, 6, 12] along the known solution path is [6, 18] and of [6, 18] is [24].

Partial soundness test For the partial soundness test we check whether the number of elements in a successor state is one less than for the parent state.

Solution validation A solution is a sequence of states s_0, s_1, s_2, s_3, where s_0 is the initial state, $s_3 = [24]$ is the goal state, and $\langle s_0, s_1 \rangle$, $\langle s_1, s_2 \rangle$, and $\langle s_2, s_3 \rangle$, are valid transitions. We check that all these hold for a given sequence.

5.2 BlocksWorld

BlocksWorld is a classic AI planning domain, where the task is to rearrange blocks in towers (Gupta and Nau 1992). There are 4 actions: stack a block on top of another block, unstack a block from another block, put a block down on the table, and pick a block up from the table. States are represented as dictionaries based on 'clear', 'on-table', 'arm-empty', 'holding', and 'on', describing whether a block is clear (no block above it in the tower), the block is on the table, whether the arm is not holding a block and which blocks are on which.

Data The domain has a PDDL representation and a large collection of 502 instances was created by Valmeekam et al. (2023a) and used in the recent literature (Hao et al. 2023a). We use the entire collection for evaluation and invent 2 example states (and transitions along 2 plans) per unit test. The examples can be found in the Appendix.

Partial soundness test For the partial soundness test we notice that in each tower there is a top block (that is clear) and there is a bottom block (that is on the table). Therefore we simply check that the number of blocks in the 'clear' list is the same as in the 'on-table' list.

Solution validation As the instances are given in PDDL, we simply translate the solution into a PDDL format and use an external validator VAL (Howey and Long 2003).

5.3 Mini Crosswords

The mini crosswords (Yao et al. 2023b) is a 5x5 crosswords dataset where the input describes the 5 horizontal and 5 vertical clues and the output is the full 25 letters board. We provide a list of horizontal and vertical clues which are strings of words. The verifier ensures that the size of each word in the rows or columns does not exceed 5.

Data We use the existing 20 instances (Yao et al. 2023b; Katz et al. 2024), all used for evaluation, with the unit tests constructed based on 3 invented states each, with the successor completeness based on a state in which one horizontal and one vertical clue already filled, which limits the number of possible successors considerably.

Partial soundness test The partial soundness test verifies that at most 5 new letters are filled in one transition, as well as that the number of unfilled letters does not get larger.

Solution validation A crossword puzzle is solved if the end result is valid, meaning every vertical and horizontal clue is present in the list of possible clues.
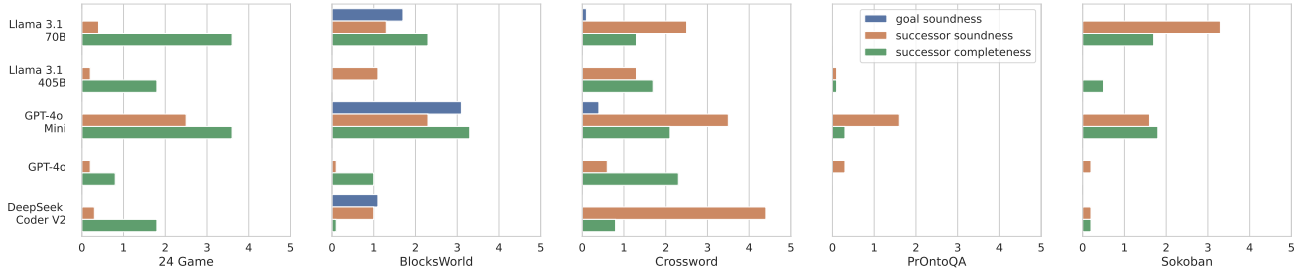
Figure 3: Average number of feedback calls for goal soundness, successor soundness, and successor completeness.

5.4 PrOntoQA

Logical reasoning can be viewed as a search problem of finding a sequence of logical rules that when applied to the known facts, derive or disprove the target hypothesis. Previous work applies MCTS with successor function and rewards obtained by calling an LLM, to examples from the PrOntoQA dataset (Hao et al. 2023b) to derive the answer but also the proof, a sequence of reasoning steps. A state is therefore a set of the facts known to be true.

Data We use the existing set of 4000 instances entirely for evaluation, inventing 3 examples per unit test.

Partial soundness test A partial soundness test simply checks that each transition adds a single known fact to the state, ensuring that the state size increases by exactly 1.

Solution validation In order to validate the solution, we compare to the known correct answer.

5.5 Sokoban

Sokoban (Junghanns and Schaeffer 1997) is a planning problem with PSPACE-complete complexity even for non-optimal planning. The problem, despite its simple conceptual rules, is a notoriously hard for generic AI planners and even for specialized solvers. We use a 2-D grid setup, in which, given equal number of boxes and goal squares, the player needs to push all boxes to goal squares without crossing walls or pushing boxes into walls. The player can only move upward, downward, leftward and rightward where many pushes are irreversible. The domain has a known planning model, described in PDDL of varying grid sizes and difficulties. States are represented as dictionaries with entries: 'at-player,' which represents a single pair of coordinates, and 'at-stone', a list of coordinates for the stones.

Data We use the collection of PDDL problem instances from the International Planning Competition (IPC) 2008. Out of these instances, we select a subset that can be solved relatively quickly by using the blind search configuration of the efficient planner Fast Downward (Helmert 2006) and choose the instances that were solved in under 5 seconds. This resulted in 11 instances. We use the entire set for evaluation and invent 3 states per unit test.

Partial soundness test The test simply checks whether the locations of the player and the stones are all different.

Solution validation Similar to BlocksWorld, we translate the solution to PDDL format and use VAL.

Figure 2 depicts the progression of accuracy values for three time points in the process, comparing using the partial soundness test (solid lines, 'w/ val') and not (dotted lines, 'w/o val'). Same colors represent the same language model. The first point in the process corresponds to when the search components are first created, meaning no feedback at all. The second point in the process is when the goal and successor function soundness tests are not failing. The third and final point is the end of the process, when successor completeness tests are not failing. Each point is also annotated with the percentage of cases the step was reached. The aggregation is performed over such cases. The figure allows us to find answers for both the first and the second hypotheses. We can clearly see the benefit from using the partial soundness test, even as simple as the ones we described above. Going forward, we therefore restrict our attention to using the partial soundness test. Further, we can clearly see the strong increase in accuracy when not stopping after the soundness test passes and performing the completeness tests, across all models.

Table 1 shows the total number of calls to the language model until soundness and completeness tests pass. Note that the minimum number of calls is 2, one for each component, even without feedback. We see that the number of automated calls is comparable to the one when a human expert is giving the feedback to the model. To look deeper into how the feedback is partitioned among the three phases, Figure 3 compares the numbers across language models and domains. We see that the larger models rarely require any feedback on the goal function and only a few iterations on the successor function, and more often than not on completeness.

Finally, we can observe that there is no single model that performs better than all other, according to all parameters and the performance is quite consistent across the large models. Interestingly, the smaller model GPT-4o-mini performs quite well in terms of accuracy.

6 Code Errors Discussion

To be able to improve the performance of the large language models in generating search components, it is important to understand the errors in the code produced by these models. In what follows we first present the error categories and show the partitioning of the errors to these categories and then elaborate on a few interesting cases.
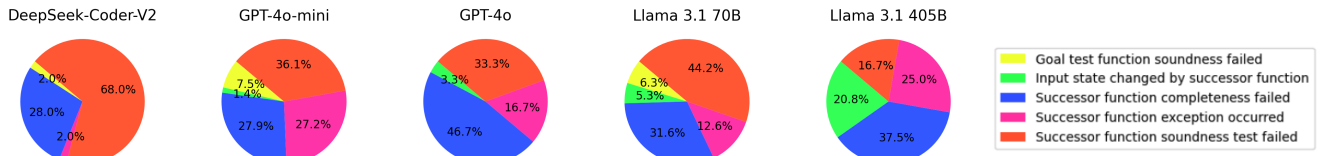
Figure 4: Partition of the errors in the generated code.

6.1 Error categories

AutoToS distinguishes 10 error categories and gives each a separate feedback.

1. *succ* soundness test failed.
2. Input state changed by *succ*.
3. *succ* completeness failed.
4. *isgoal* soundness failed.
5. *succ* exception occurred.
6. *isgoal* exception occurred.
7. Search timeout in *succ* soundness test.
8. *succ* execution took too long.
9. *isgoal* execution took too long.
10. Response parsing error.

Interestingly, we did not observe any errors in the last two categories. Further only 1, 2, and 3 errors in categories 6, 8, and 7, respectively. The partition of the errors to the other 5 categories (see Figure 4), shows just how much the models differ in the type of errors produced. Interestingly, the DeepSeek-Coder-V2 model rarely produces code that triggers exception or changes the input state and even typically passes the goal soundness test. Other models, especially the smaller ones, are more diverse in errors produced. Across all models, the majority of the errors account for the failed successor soundness and completeness tests.

6.2 Bloopers

We noticed a few "bloopers," interesting phenomena that occur during AutoToS. We share these observations in a hope of shedding some light onto future understanding of LLM code generation for planning and search problems.

The first blooper occurs in the 5x5 Crossword for Llama3.1-70b. The representation of a Crossword instance includes vertical and horizontal clues which are lists of 5 words each. The model handles horizontal clues well by simply checking whether a word in row i is in the ith list in horizontal clues. For vertical clues, however, the model checks whether the word in column i is at position i among the clues for every column. Indeed the initial prompt from obtaining successor function clearly states that:

> [...] horizontal_answers is a list where element i is a list of possible answers to clue in row i, and vertical_answers is a list where element i is a list of possible answers to clue in column i.

The second blooper occurs in the GPT-4o-mini, Llama3.1-70b, and even in Llama3.1-405b on the BlocksWorld domain. When generating successors for the unstack block from another block action, the models check if the block is clear, but never actually check whether the arm is empty. The resulting code, in cases when a block is already held, can generate a successor state in which the held block is overwritten with the one that is unstacked, and therefore disappears from the state. On some instances in the evaluation set the situation does not occur. On others, invalid solutions are produced and the accuracy score falls far below 100%. The AutoToS feedback in the next iterations often solves the problem.

Another blooper occurs in Sokoban, when Llama3.1-70b generates the initial successor function and the goal test, and no partial soundness check is performed. The model generates a helper function *is_clear* that only checks whether the location on the grid is 0 or 2 (not a wall), disregarding whether any of the stones are currently at the location. This allows the player to move and push stones to the locations of other stones, resulting in the accuracy score of 0. Since the unit tests pass in this case, no additional iterations were performed. The partial soundness check would catch the error the first time a faulty state is generated (a state where multiple stones are at the same location or a player and a stone are at the same location). The prompt explicitly states what it means to be clear:

> The maze is defined by a grid of values 0,1, and 2, where 2 means it is a goal location for a stone, 1 means the cell is blocked, and either 0 or 2 means that the cell can be occupied. A cell is clear if it can be occupied, but is not occupied by either the player or any stone.

Yet another blooper happens in 24 Game with GPT-4o-mini and DeepSeek-CoderV2 when no partial soundness check is performed. When creating a new state out of the input state, two numbers are chosen to perform an arithmetic operation and in order to obtain the remaining numbers, the code selects the numbers from the state that are different from the two chosen numbers. Thus in cases of duplicate numbers, the state size becomes more than one smaller than of the parent and on some instances the produced solutions would not be valid. The AutoToS completeness feedback eventually solves the problem in these cases.

7 Conclusions and Future Work

We automate the process of generating correct and sound code for the search components by leveraging debugging and exception handing with natural language, code feedback, iterative reprompting. We demonstrate the performance of our approach, AutoToS, across various sized models and across search problem domains used by the planning community. With just a few calls to the language model, we demonstrate that we can obtain the search components without any direct human in the loop feedback, ensuring soundness, completeness, accuracy, and nearly 100% accuracy across all models and all domains.

For future work it would be interesting to see if the language models could generate the unit tests as well as the

partial soundness tests instead of relying on the user writing these for a specific domain. The partial soundness test is related to the notion of invariants in planning (Alcázar and Torralba 2015). It is worth exploring whether LLMs can help us derive such invariants. Finally, seeing that smaller language models can achieve accuracy on par with the largest ones, begs the question of whether it would be possible to finetune an even smaller model and achieve a similar or better accuracy.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI@Meta. 2024. Llama 3 Model Card.

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Besta, M.; Blach, N.; Kubicek, A.; Gerstenberger, R.; Podstawski, M.; Gianinazzi, L.; Gajda, J.; Lehmann, T.; Niewiadomski, H.; Nyczyk, P.; et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17682–17690.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; et al. 2021. Evaluating Large Language Models Trained on Code. *CoRR*, abs/2107.03374.

Chen, X.; Lin, M.; Schärli, N.; and Zhou, D. 2024. Teaching Large Language Models to Self-Debug. In *ICLR*. OpenReview.net.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1990. *Introduction to Algorithms*. The MIT Press.

DeepSeek-AI; Liu, A.; Feng, B.; Wang, B.; Wang, B.; Liu, B.; Zhao, C.; Dengr, C.; Ruan, C.; et al. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; Goyal, A.; et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-Driven Planning from Minimal Text Descriptions. arXiv:2405.04215.

Gou, Z.; Shao, Z.; Gong, Y.; yelong shen; Yang, Y.; Duan, N.; and Chen, W. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *The Twelfth International Conference on Learning Representations*.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36: 79081–79094.

Gupta, N.; and Nau, D. S. 1992. On the Complexity of Blocks-World Planning. 56(2–3): 223–254.

Hao, S.; Gu, Y.; Ma, H.; Hong, J.; Wang, Z.; Wang, D.; and Hu, Z. 2023a. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*.

Hao, S.; Gu, Y.; Ma, H.; Hong, J.; Wang, Z.; Wang, D.; and Hu, Z. 2023b. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 8154–8173.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Howey, R.; and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In Edelkamp, S.; and Hoffmann, J., eds., *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks*.

Huang, J.; Chen, X.; Mishra, S.; Zheng, H. S.; Yu, A. W.; Song, X.; and Zhou, D. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations*.

Jiang, A. Q.; Sablayrolles, A.; Roux, A.; Mensch, A.; et al. 2024. Mixtral of Experts. arXiv:2401.04088.

Jiang, S.; Wang, Y.; and Wang, Y. 2023. SelfEvolve: A Code Evolution Framework via Large Language Models. arXiv:2306.02907.

Junghanns, A.; and Schaeffer, J. 1997. Sokoban: A Challenging Single-Agent Search Problem. In *International Joint Conference on Artificial Intelligence*.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024a. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Forty-first International Conference on Machine Learning*.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024b. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Forty-first International Conference on Machine Learning*.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. arXiv:2404.11833 [cs.AI].

Katz, M.; Moshkovich, D.; and Karpas, E. 2018. Semi-Black Box: Rapid Development of Planning Based Solutions. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 6211–6218. AAAI Press.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213.

Li, Y.; Parsert, J.; and Polgreen, E. 2024. Guiding Enumerative Program Synthesis with Large Language Models. In *CAV (2)*, volume 14682 of *Lecture Notes in Computer Science*, 280–301. Springer.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *CoRR*, abs/2304.11477.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 46534–46594. Curran Associates, Inc.

Muennighoff, N.; Liu, Q.; Zebaze, A. R.; Zheng, Q.; Hui, B.; Zhuo, T. Y.; Singh, S.; Tang, X.; von Werra, L.; and Longpre, S. 2024. OctoPack: Instruction Tuning Code Large Language Models. In *ICLR*. OpenReview.net.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large Language Models as Planning Domain Generators. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*. AAAI Press.

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Horesh, L.; Srivastava, B.; Fabiano, F.; and Loreggia, A. 2022. Plansformer: Generating Symbolic Plans using Transformers. arXiv:2212.08681 [cs.AI].

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Srivastava, B.; Horesh, L.; Fabiano, F.; and Loreggia, A. 2023. Understanding the Capabilities of Large Language Models for Automated Planning. *CoRR*, abs/2305.16151.

Puri, R.; Kung, D. S.; Janssen, G.; Zhang, W.; Domeniconi, G.; Zolotov, V.; Dolby, J.; Chen, J.; Choudhury, M. R.; Decker, L.; Thost, V.; Buratti, L.; Pujar, S.; Ramji, S.; Finkler, U.; Malaika, S.; and Reiss, F. 2021. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. In *NeurIPS Datasets and Benchmarks*.

Raman, S. S.; Cohen, V.; Rosen, E.; Idrees, I.; Paulius, D.; and Tellex, S. 2022. Planning With Large Language Models Via Corrective Re-Prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Sel, B.; Al-Tawaha, A.; Khattar, V.; Wang, L.; Jia, R.; and Jin, M. 2023. Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models. *CoRR*, abs/2308.10379.

Shinn, N.; Cassano, F.; Gopinath, A.; Narasimhan, K.; and Yao, S. 2023. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J.; Pack Kaelbling, L.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*. AAAI Press.

Silver, T.; Hariprasad, V.; Shuttleworth, R. S.; Kumar, N.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. PDDL Planning with Pretrained Large Language Models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Stechly, K.; Valmeekam, K.; and Kambhampati, S. 2024. On the Self-Verification Limitations of Large Language Models on Reasoning and Planning Tasks. *arXiv preprint arXiv:2402.08115*.

Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023a. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 38975–38987.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023b. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Xie, Y.; Yu, C.; Zhu, T.; Bai, J.; Gong, Z.; and Soh, H. 2023. Translating Natural Language to Planning Goals with Large-Language Models. *CoRR*, abs/2302.05128.

Yang, J.; Jimenez, C. E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; and Press, O. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. R. 2023b. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhang, K.; Li, Z.; Li, J.; Li, G.; and Jin, Z. 2023. Self-Edit: Fault-Aware Code Editor for Code Generation. In *ACL (1)*, 769–787. Association for Computational Linguistics.

Zhong, L.; Wang, Z.; and Shang, J. 2024. Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step-by-step. arXiv:2402.16906.

Zhou, A.; Yan, K.; Shlapentokh-Rothman, M.; Wang, H.; and Wang, Y. 2023. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models. *CoRR*, abs/2310.04406.

Zuo, M.; Velez, F. P.; Li, X.; Littman, M. L.; and Bach, S. H. 2024. Planetarium: A Rigorous Benchmark for Translating Text to Structured Planning Languages. *CoRR*, abs/2407.03321.

A Additional data for experimental domains

We provide additional information on the domains included in our experimental evaluation, such as examples used in unit tests, code for the partial successor soundness test, etc.

A.1 24 Game

Goal Unit Test Goal unit test cases are stored in two *jsonl* files, one for goal states and one for non-goal states.

Listing 2: 24game_goal_states.jsonl

```
1    [24]
```

Listing 3: 24game_non_goal_states.jsonl

```
1    []
2    [3]
3    [24, 1]
4    [1, 6, 4]
5    [1, 1, 4, 6]
```

Successor Unit Test Successor unit test cases are stored in a jsonl file. The test cases used are depicted in Listing 4.

```
1   [[1, 1, 4, 6], [[1, 1, 10],
    ↪ [0.6666666666666666, 1, 1], [1, 4, 7], [-2,
    ↪ 1, 1], [-5, 1, 4], [1, 4, 6], [1, 1, 2],
    ↪ [1, 5, 6], [0.25, 1, 6], [-3, 1, 6], [0, 4,
    ↪ 6], [0.16666666666666666, 1, 4], [1, 1,
    ↪ 24], [1, 3, 6], [2, 4, 6], [1, 4, 5], [1,
    ↪ 1, 1.5]]]
2   [[1, 1, 11, 11], [[1, 11, 11],
    ↪ [0.09090909090909091, 1, 11], [0, 11, 11],
    ↪ [1, 1, 22], [2, 11, 11], [0, 1, 1], [1, 1,
    ↪ 121], [1, 11, 12], [1, 1, 1.0], [1, 10,
    ↪ 11], [-10, 1, 11]]]
3   [[1, 1, 3, 8], [[-2, 1, 8], [1, 3, 8],
    ↪ [0.3333333333333333, 1, 8], [-7, 1, 3], [1,
    ↪ 1, 2.6666666666666665], [0.125, 1, 3], [2,
    ↪ 3, 8], [1, 3, 7], [1, 1, 11], [1, 1, 5],
    ↪ [1, 1, 24], [-5, 1, 1], [0.375, 1, 1], [1,
    ↪ 2, 8], [1, 3, 9], [0, 3, 8], [1, 4, 8]]]
4   [[1, 1, 1, 8], [[0.125, 1, 1], [1, 1, 9], [1,
    ↪ 1, 8], [0, 1, 8], [1, 2, 8], [1, 1, 7],
    ↪ [-7, 1, 1]]]
5   [[6, 6, 6, 6], [[1.0, 6, 6], [6, 6, 12], [0, 6,
    ↪ 6], [6, 6, 36]]]
6   [[1, 1, 2, 12], [[1, 3, 12], [-10, 1, 1], [1,
    ↪ 1, 10], [2, 2, 12], [1, 2, 13], [0.5, 1,
    ↪ 12], [-11, 1, 2], [1, 1, 12], [1, 1, 6.0],
    ↪ [1, 2, 12], [0, 2, 12], [1, 1, 24], [1, 2,
    ↪ 11], [1, 1, 14], [0.16666666666666666, 1,
    ↪ 1], [0.08333333333333333, 1, 2], [-1, 1,
    ↪ 12]]]
```

```
7   [[1, 2, 2, 6], [[2, 2, 6], [-5, 2, 2], [2, 3,
    ↪ 6], [1, 2, 6], [0.3333333333333333, 1, 2],
    ↪ [2, 2, 5], [1, 1.0, 6],
    ↪ [0.16666666666666666, 2, 2], [1, 4, 6], [0,
    ↪ 1, 6], [-4, 1, 2], [1, 2, 12], [1, 2, 3.0],
    ↪ [2, 2, 7], [-1, 2, 6], [1, 2, 8], [1, 2,
    ↪ 4], [0.5, 2, 6]]]
8   [[1, 1, 10, 12], [[-9, 1, 12], [1, 1, 1.2],
    ↪ [0.08333333333333333, 1, 10], [-2, 1, 1],
    ↪ [1, 10, 13], [1, 1, 22], [2, 10, 12], [0.1,
    ↪ 1, 12], [1, 1, 120], [1, 1, 2],
    ↪ [0.8333333333333334, 1, 1], [1, 9, 12], [1,
    ↪ 10, 12], [0, 10, 12], [1, 11, 12], [1, 10,
    ↪ 11], [-11, 1, 10]]]
9   [[2, 2, 10, 10], [[0, 2, 2], [2, 10, 12], [1.0,
    ↪ 10, 10], [0, 10, 10], [-8, 2, 10], [2, 2,
    ↪ 100], [2, 5.0, 10], [1.0, 2, 2], [2, 2,
    ↪ 20], [4, 10, 10], [0.2, 2, 10], [2, 8, 10],
    ↪ [2, 10, 20]]]
10  [[1, 1, 1, 12], [[0.08333333333333333, 1, 1],
    ↪ [1, 1, 13], [1, 1, 12], [0, 1, 12], [1, 2,
    ↪ 12], [-11, 1, 1], [1, 1, 11]]]
11  [[1, 4, 6], [[4, 6]]]
12  [[4, 6], [[24]]]
13  [[1, 1, 22], [[1, 23]]]
14  [[1, 23], [[24]]]
15  [[1, 1, 24], [[1, 24]]]
16  [[1, 24], [[24]]]
17  [[1, 2, 8], [[3, 8]]]
18  [[3, 8], [[24]]]
19  [[6, 6, 12], [[6, 18]]]
20  [[6, 18], [[24]]]
21  [[0, 2, 12], [[0, 24]]]
22  [[0, 24], [[24]]]
23  [[2, 2, 6], [[2, 12]]]
24  [[2, 12], [[24]]]
25  [[1, 11, 12], [[11, 13]]]
26  [[11, 13], [[24]]]
27  [[2, 2, 20], [[2, 22]]]
28  [[2, 22], [[24]]]
29  [[1, 1, 12], [[2, 12]]]
```

Listing 4: 24game_successors.jsonl

Partial Successor Soundness Test The code for the partial successor soundness test is as follows.

```python
def validate_transition_complex(s, t):
    if len(s) - len(t) != 1:
        feedback = prettyprint("Invalid
            ↪ transformation: length
            ↪ mismatch - the length of a
            ↪ successor must be one less
            ↪ than the parent.")
```

```python
        feedback += prettyprint("Let's
        ↪  think step by step. First
        ↪  think through in words why
        ↪  the successor function
        ↪  produced a successor that
        ↪  had a length that was not
        ↪  exactly one less than the
        ↪  parent. Then provide the
        ↪  complete Python code for
        ↪  the revised successor
        ↪  function that ensures the
        ↪  length of a successor is
        ↪  exactly one less than the
        ↪  parent.")
        feedback +=
        ↪  prettyprint("Remember how
        ↪  you fixed the previous
        ↪  mistakes, if any. Keep the
        ↪  same function signature.")
        return False, feedback
    return True, ""
```

A.2 Blocksworld

Goal Unit Test Goal unit test cases are stored in two *jsonl* files, one for goal states and one for non-goal states, depicted in Listings 5 and 6.

Listing 5: blocks_goal_states.jsonl

```
1   [
2     {   "state": {
3           "clear": ["b"],
4           "on-table": ["d"],
5           "arm-empty": true,
6           "holding": null,
7           "on": [["a", "c"],["b",
           ↪  "a"],["c","d"]]
8         },
9         "goal": {
10          "clear": [],
11          "on-table": [],
12          "on":
           ↪  [["a","c"],["b","a"],["c","d"]]
13        }
14    },
15    {   "state": {
16          "clear": ["a"],
17          "on-table": ["d"],
18          "arm-empty": false,
19          "holding": "b",
20          "on": [["a","c"],["c","d"]]
21        },
22        "goal": {
23          "clear": [],
24          "on-table": [],
25          "on": [["a","c"]]
26        }
27    }
28  ]
```

Listing 6: blocks_non_goal_states.jsonl

```
1   [
2     {
3         "state": {
4           "clear": ["b"],
5           "on-table": ["d"],
6           "arm-empty": true,
7           "holding": null,
8           "on":
           ↪  [["a","c"],["b","a"],["c","d"]]
9         },
10        "goal": {
11          "clear": [],
12          "on-table": [],
13          "on":
           ↪  [["a","b"],["b","c"],["c","d"]]
14        }
15    },
16    {
17        "state": {
18          "clear": ["a"],
19          "on-table": ["d"],
20          "arm-empty": false,
21          "holding": "b",
22          "on": [["a","c"],["c","d"]]
23        },
24        "goal":
25          {
26          "clear": [],
27          "on-table": [],
28          "on": [["a","c"],[   "c","b"]]
29        }
30    }
31  ]
```

Successor Unit Test   Successor unit test cases are stored in a jsonl file, depicted in Listing 7.

```
1    [
2        {
3            "state": {
4                "clear": ["b"],
5                "on-table": ["d"],
6                "arm-empty": true,
7                "holding": null,
8                "on":
                 ↪   [["a","c"],["b","a"],["c","d"]]
9            },
10           "successors": [
11               {
12                   "clear": ["a"],
13                   "on-table": ["d"],
14                   "arm-empty": false,
15                   "holding": "b",
16                   "on": [["a","c"],["c","d"]]
17               }
18           ]
19       },
20       {
21           "state": {
22               "clear": ["a"],
23               "on-table": ["d"],
24               "arm-empty": false,
25               "holding": "b",
26               "on": [["a","c"],["c","d"]]
27           },
28           "successors": [
29               {
30                   "clear": ["a","b"],
31                   "on-table": ["d","b"],
32                   "arm-empty": true,
33                   "holding": null,
34                   "on": [["a","c"],["c","d"]]
35               },
36               {
37                   "clear": ["b"],
38                   "on-table": ["d"],
39                   "arm-empty": true,
40                   "holding": null,
41                   "on":
                     ↪   [["a","c"],["c","d"],["b","a"]]
42               }
43           ]
44       },
45       {
46           "state": {
47               "clear": ["a","b","d"],
48               "on-table": ["a","c","d"],
49               "arm-empty": true,
50               "holding": null,
51               "on": [["b","c"]]
52           },
53           "successors": [
54               {
55                   "clear": ["b","d"],
56                   "on-table": ["c","d"],
57                   "arm-empty": false,
58                   "holding": "a",
59                   "on": [["b","c"]]
60               },
61               {
62                   "clear": ["a","b"],
63                   "on-table": ["a","c"],
64                   "arm-empty": false,
65                   "holding": "d",
66                   "on": [["b","c"]]
67               },
68               {
69                   "clear": ["a","d","c"],
70                   "on-table": ["a","c","d"],
71                   "arm-empty": false,
72                   "holding": "b",
73                   "on": []
74               }
75           ]
76       },
77       {
78           "state": {
79               "clear": ["b","d"],
80               "on-table": ["c","d"],
81               "arm-empty": false,
82               "holding": "a",
83               "on": [["b","c"]]
84           },
85           "successors": [
86               {
87                   "clear": ["b","d","a"],
88                   "on-table": ["c","d","a"],
89                   "arm-empty": true,
90                   "holding": null,
91                   "on": [["b","c"]]
92               },
93               {
94                   "clear": ["d","a"],
95                   "on-table": ["c","d"],
96                   "arm-empty": true,
97                   "holding": null,
98                   "on": [["b","c"],["a","b"]]
99               },
100              {
101                  "clear": ["b","a"],
102                  "on-table": ["c","d"],
103                  "arm-empty": true,
104                  "holding": null,
105                  "on": [["b","c"],["a","d"]]
106              }
107          ]
108      },
109      {
110          "state": {
111              "clear": ["a","d"],
112              "on-table": ["b","c"],
113              "arm-empty": true,
114              "holding": null,
115              "on": [["a","b"],["d","c"]]
116          },
117          "successors": [
118              {
119                  "clear": ["d","b"],
120                  "on-table": ["b","c"],
121                  "arm-empty": false,
122                  "holding": "a",
123                  "on": [["d","c"]]
124              },
```

```
125                     {
126                         "clear": ["a","c"],
127                         "on-table": ["b","c"],
128                         "arm-empty": false,
129                         "holding": "d",
130                         "on": [["a","b"]]
131                     }
132                 ]
133             },
134             {
135                 "state": {
136                     "clear": ["d","b"],
137                     "on-table": ["b","c"],
138                     "arm-empty": false,
139                     "holding": "a",
140                     "on": [["d","c"]]
141                 },
142                 "successors": [
143                     {
144                         "clear": ["d","b","a"],
145                         "on-table": ["b","c","a"],
146                         "arm-empty": true,
147                         "holding": null,
148                         "on": [["d","c"]]
149                     },
150                     {
151                         "clear": ["b","a"],
152                         "on-table": ["b","c"],
153                         "arm-empty": true,
154                         "holding": null,
155                         "on": [["d","c"],["a","d"]]
156                     },
157                     {
158                         "clear": ["d","a"],
159                         "on-table": ["b","c"],
160                         "arm-empty": true,
161                         "holding": null,
162                         "on": [["d","c"],["a","b"]]
163                     }
164                 ]
165             },
166             {
167                 "state": {
168                     "clear": ["b"],
169                     "on-table": ["a"],
170                     "arm-empty": true,
171                     "holding": null,
172                     "on":
                        ↪ [["b","c"],["c","d"],["d","a"]]
173                 },
174                 "successors": [
175                     {
176                         "clear": ["c"],
177                         "on-table": ["a"],
178                         "arm-empty": false,
179                         "holding": "b",
180                         "on": [["c","d"],["d","a"]]
181                     }
182                 ]
183             },
184             {
185                 "state": {
186                     "clear": ["c"],
187                     "on-table": ["a"],
188                     "arm-empty": false,

189                     "holding": "b",
190                     "on": [["c","d"],["d","a"]]
191                 },
192                 "successors": [
193                     {
194                         "clear": ["c","b"],
195                         "on-table": ["a","b"],
196                         "arm-empty": true,
197                         "holding": null,
198                         "on": [["c","d"],["d","a"]]
199                     },
200                     {
201                         "clear": ["b"],
202                         "on-table": ["a"],
203                         "arm-empty": true,
204                         "holding": null,
205                         "on":
                            ↪ [["c","d"],["d","a"],["b","c"]]
206                     }
207                 ]
208             },
209             {
210                 "state": {
211                     "clear": ["d"],
212                     "on-table": ["b"],
213                     "arm-empty": true,
214                     "holding": null,
215                     "on":
                        ↪ [["a","c"],["c","b"],["d","a"]]
216                 },
217                 "successors": [
218                     {
219                         "clear": ["a"],
220                         "on-table": ["b"],
221                         "arm-empty": false,
222                         "holding": "d",
223                         "on": [["a","c"],["c","b"]]
224                     }
225                 ]
226             },
227             {
228                 "state": {
229                     "clear": ["a"],
230                     "on-table": ["b"],
231                     "arm-empty": false,
232                     "holding": "d",
233                     "on": [["a","c"],["c","b"]]
234                 },
235                 "successors": [
236                     {
237                         "clear": ["a","d"],
238                         "on-table": ["b","d"],
239                         "arm-empty": true,
240                         "holding": null,
241                         "on": [["a","c"],["c","b"]]
242                     },
243                     {
244                         "clear": ["d"],
245                         "on-table": ["b"],
246                         "arm-empty": true,
247                         "holding": null,
248                         "on":
                            ↪ [["a","c"],["c","b"],["d","a"]]
249                     }
250                 ]
```

```
251              },
252              {
253                  "state": {
254                      "clear": ["c","d"],
255                      "on-table": ["a","d"],
256                      "arm-empty": true,
257                      "holding": null,
258                      "on": [["b","a"],["c","b"]]
259                  },
260                  "successors": [
261                      {
262                          "clear": ["c"],
263                          "on-table": ["a"],
264                          "arm-empty": false,
265                          "holding": "d",
266                          "on": [["b","a"],["c","b"]]
267                      },
268                      {
269                          "clear": ["d","b"],
270                          "on-table": ["a","d"],
271                          "arm-empty": false,
272                          "holding": "c",
273                          "on": [["b","a"]]
274                      }
275                  ]
276              },
277              {
278                  "state": {
279                      "clear": ["c"],
280                      "on-table": ["a"],
281                      "arm-empty": false,
282                      "holding": "d",
283                      "on": [["b","a"],["c","b"]]
284                  },
285                  "successors": [
286                      {
287                          "clear": ["c","d"],
288                          "on-table": ["a","d"],
289                          "arm-empty": true,
290                          "holding": null,
291                          "on": [["b","a"],["c","b"]]
292                      },
293                      {
294                          "clear": ["d"],
295                          "on-table": ["a"],
296                          "arm-empty": true,
297                          "holding": null,
298                          "on":
                             ↪  [["b","a"],["c","b"],["d","c"]]
299                      }
300                  ]
301              },
302              {
303                  "state": {
304                      "clear": ["d"],
305                      "on-table": ["b"],
306                      "arm-empty": true,
307                      "holding": null,
308                      "on":
                         ↪  [["a","c"],["c","b"],["d","a"]]
309                  },
310                  "successors": [
311                      {
312                          "clear": ["a"],
313                          "on-table": ["b"],
314                          "arm-empty": false,
315                          "holding": "d",
316                          "on": [["a","c"],["c","b"]]
317                      }
318                  ]
319              },
320              {
321                  "state": {
322                      "clear": ["a"],
323                      "on-table": ["b"],
324                      "arm-empty": false,
325                      "holding": "d",
326                      "on": [["a","c"],["c","b"]]
327                  },
328                  "successors": [
329                      {
330                          "clear": ["a","d"],
331                          "on-table": ["b","d"],
332                          "arm-empty": true,
333                          "holding": null,
334                          "on": [["a","c"],["c","b"]]
335                      },
336                      {
337                          "clear": ["d"],
338                          "on-table": ["b"],
339                          "arm-empty": true,
340                          "holding": null,
341                          "on":
                             ↪  [["a","c"],["c","b"],["d","a"]]
342                      }
343                  ]
344              },
345              {
346                  "state": {
347                      "clear": ["a"],
348                      "on-table": ["c"],
349                      "arm-empty": true,
350                      "holding": null,
351                      "on":
                         ↪  [["a","d"],["b","c"],["d","b"]]
352                  },
353                  "successors": [
354                      {
355                          "clear": ["d"],
356                          "on-table": ["c"],
357                          "arm-empty": false,
358                          "holding": "a",
359                          "on": [["b","c"],["d","b"]]
360                      }
361                  ]
362              },
363              {
364                  "state": {
365                      "clear": ["d"],
366                      "on-table": ["c"],
367                      "arm-empty": false,
368                      "holding": "a",
369                      "on": [["b","c"],["d","b"]]
370                  },
371                  "successors": [
372                      {
373                          "clear": ["d","a"],
374                          "on-table": ["c","a"],
375                          "arm-empty": true,
376                          "holding": null,
```

```
377                         "on": [["b","c"],["d","b"]]
378                     },
379                     {
380                         "clear": ["a"],
381                         "on-table": ["c"],
382                         "arm-empty": true,
383                         "holding": null,
384                         "on":
                         ↪  [["b","c"],["d","b"],["a","d"]]
385                     }
386                 ]
387         },
388         {
389             "state": {
390                 "clear": ["a","b","d"],
391                 "on-table": ["a","c","d"],
392                 "arm-empty": true,
393                 "holding": null,
394                 "on": [["b","c"]]
395             },
396             "successors": [
397                 {
398                     "clear": ["b","d"],
399                     "on-table": ["c","d"],
400                     "arm-empty": false,
401                     "holding": "a",
402                     "on": [["b","c"]]
403                 },
404                 {
405                     "clear": ["a","b"],
406                     "on-table": ["a","c"],
407                     "arm-empty": false,
408                     "holding": "d",
409                     "on": [["b","c"]]
410                 },
411                 {
412                     "clear": ["a","d","c"],
413                     "on-table": ["a","c","d"],
414                     "arm-empty": false,
415                     "holding": "b",
416                     "on": []
417                 }
418             ]
419         },
420         {
421             "state": {
422                 "clear": ["b","d"],
423                 "on-table": ["c","d"],
424                 "arm-empty": false,
425                 "holding": "a",
426                 "on": [["b","c"]]
427             },
428             "successors": [
429                 {
430                     "clear": ["b","d","a"],
431                     "on-table": ["c","d","a"],
432                     "arm-empty": true,
433                     "holding": null,
434                     "on": [["b","c"]]
435                 },
436                 {
437                     "clear": ["d","a"],
438                     "on-table": ["c","d"],
439                     "arm-empty": true,
440                     "holding": null,
441                     "on": [["b","c"],["a","b"]]
442                 },
443                 {
444                     "clear": ["b","a"],
445                     "on-table": ["c","d"],
446                     "arm-empty": true,
447                     "holding": null,
448                     "on": [["b","c"],["a","d"]]
449                 }
450             ]
451         },
452         {
453             "state": {
454                 "clear": ["b","c"],
455                 "on-table": ["a","b"],
456                 "arm-empty": true,
457                 "holding": null,
458                 "on": [["c","d"],["d","a"]]
459             },
460             "successors": [
461                 {
462                     "clear": ["c"],
463                     "on-table": ["a"],
464                     "arm-empty": false,
465                     "holding": "b",
466                     "on": [["c","d"],["d","a"]]
467                 },
468                 {
469                     "clear": ["b","d"],
470                     "on-table": ["a","b"],
471                     "arm-empty": false,
472                     "holding": "c",
473                     "on": [["d","a"]]
474                 }
475             ]
476         },
477         {
478             "state": {
479                 "clear": ["c"],
480                 "on-table": ["a"],
481                 "arm-empty": false,
482                 "holding": "b",
483                 "on": [["c","d"],["d","a"]]
484             },
485             "successors": [
486                 {
487                     "clear": ["c","b"],
488                     "on-table": ["a","b"],
489                     "arm-empty": true,
490                     "holding": null,
491                     "on": [["c","d"],["d","a"]]
492                 },
493                 {
494                     "clear": ["b"],
495                     "on-table": ["a"],
496                     "arm-empty": true,
497                     "holding": null,
498                     "on":
                     ↪  [["c","d"],["d","a"],["b","c"]]
499                 }
500             ]
501         }
502     ]
```

Listing 7: blocks_successors.jsonl

Partial Successor Soundness Test

```python
def validate_transition_complex(parent, state):
    if len(state.get('clear')) !=
↪   len(state.get('on-table')):
        feedback += prettyprint("Each tower has the
↪       bottom block on the table and the top block
↪       clear.")
        feedback += prettyprint("Therefore, the number
↪       of clear blocks should be the same as the
↪       number of blocks on the table.")
        feedback += prettyprint("The number of elements
↪       in the clear list is not the same as the
↪       number of elements in the on-table list.")
        feedback += prettyprint("Reminder: Once I pick
↪       up a block, I am holding the block and it is
↪       no longer clear and no longer on the
↪       table.")
        feedback += prettyprint("Once I unstack from on
↪       top of another block, I am holding the block
↪       and it is no longer clear. Instead, the
↪       other block becomes clear.")
        feedback += prettyprint("Once I put down a
↪       block, my hand becomes empty, the block
↪       becomes clear, and it is now on the table.")
        feedback += prettyprint("Once I stack a block on
↪       top of another block, the block on top
↪       becomes clear and the block under it is no
↪       longer clear.")

        feedback += prettyprint("Let's think step by
↪       step. First, think of how applying each
↪       action changes which blocks are clear.")
        feedback += prettyprint("Then, think of how
↪       applying each action changes which blocks
↪       are on the table.")
        feedback += prettyprint("Then, provide the
↪       complete Python code for the revised
↪       successor function that returns a list of
↪       successor states.")
        feedback += prettyprint("Remember how you fixed
↪       the previous mistakes, if any. Keep the same
↪       function signature.")
        return False, feedback
    return True, ""
```

A.3   5x5 Crosswords

Goal Unit Test   Goal unit test cases are stored in two *jsonl* files, one for goal states and one for non-goal states.

```
1   [{"state": [["a", "g", "e", "n", "d"], ["m",
↪   "o", "t", "o", "r"], ["a", "r", "t", "s",
↪   "y"], ["s", "a", "l", "l", "e"], ["s", "l",
↪   "e", "e", "r"]], "horizontal_clues":
↪   [["tasks", "goals", "plans", "agend",
↪   "chores", "works", "deeds", "items",
↪   "lists", "brief"], ["motor", "power",
↪   "drive", "diesel", "steam", "pumps",
↪   "crank", "gears", "turbn", "motor"],
↪   ["grand", "artsy", "showy", "ornate",
↪   "fancy", "vain", "proud", "vogue", "swank",
↪   "luxus"], ["venue", "salle", "forum",
↪   "atria", "lobby", "parls", "court",
↪   "malls", "mall", "lobby"], ["jeer",
↪   "scoff", "sleer", "deris", "sneer",
↪   "scorn", "derid", "gibes", "gibed",
↪   "flout"]], "vertical_clues": [["amass",
↪   "stack", "hoard", "pile", "store", "heaps",
↪   "massy", "gathe", "lumps", "mound"],
↪   ["nilga", "goral", "eland", "lepus",
↪   "gazal", "kudu", "oryx", "gnu", "imps",
↪   "carb"], ["scheme", "design", "ettle",
↪   "nettle", "sting", "wiles", "plans",
↪   "ideas", "plots", "cocks"], ["spout",
↪   "nosle", "snout", "mouth", "nostr",
↪   "ports", "inlet", "vents", "outlt",
↪   "beaks"], ["drier", "arid", "sere",
↪   "parch", "dryer", "wring", "drear", "sear",
↪   "pall", "lack"]]}, {"state": [["a", "r",
↪   "e", "f", "y"], ["r", "e", "v", "i", "e"],
↪   ["i", "g", "a", "l", "a"], ["s", "e", "d",
↪   "e", "r"], ["e", "t", "e", "r", "n"]],
↪   "horizontal_clues": [["parch", "dryup",
↪   "arefy", "wring", "suckd", "wizen",
↪   "desic", "evapo", "scald", "toast"],
↪   ["excel", "revie", "beat", "top", "best",
↪   "rise", "win", "lead", "rule", "boss"],
↪   ["igala", "tribe", "people", "race",
↪   "ethni", "nation", "yorub", "niger",
↪   "triba", "tribu"], ["seder", "meal",
↪   "food", "feast", "dine", "dish", "supper",
↪   "banqu", "treat", "fetes"], ["eterl",
↪   "etern", "everl", "forev", "immor",
↪   "endur", "const", "perma", "durab",
↪   "timeless"]], "vertical_clues": [["arise",
↪   "climb", "soar", "ascen", "mount", "leaps",
↪   "scale", "clamb", "steps", "jump"],
↪   ["regain", "renew", "recoi", "recla",
↪   "retri", "regra", "reget", "reapo",
↪   "reboo", "reset"], ["dodge", "elude",
↪   "shirk", "escap", "hide", "evade", "flee",
↪   "duck", "ditch", "evite"], ["filer",
↪   "files", "rasps", "grind", "blade",
↪   "sawer", "tool", "sharp", "knife",
↪   "metal"], ["yearn", "long", "ache",
↪   "crave", "desir", "need", "want", "thirst",
↪   "hunger", "lust"]]}, {"state": [["b", "e",
↪   "b", "o", "p"], ["u", "r", "e", "n", "a"],
↪   ["f", "r", "i", "a", "r"], ["f", "o", "n",
↪   "g", "e"], ["o", "r", "g", "a", "l"]],
↪   "horizontal_clues": [["bebop", "jazzy",
↪   "music", "salsa", "swing", "blues",
↪   "riffs", "drums", "horns", "notes"],
↪   ["senna", "urena", "herbs", "flora",
↪   "mints", "trees", "leaves", "oils",
↪   "spice", "lavas"], ["monk", "friar", "nun",
↪   "saint", "clerk", "deity", "mystic",
↪   "faith", "pious", "sacra"], ["fetch",
↪   "carry", "fonge", "take", "seize", "hold",
↪   "grab", "earn", "gain", "yield"], ["tart",
↪   "argal", "orgal", "lemon", "sours",
↪   "wines", "taste", "tangs", "zesty",
```

```
1  [{"state": [[null, null, null, null, null],
   ↪    ["m", "o", "t", "o", "r"], ["a", "r", "t",
   ↪    "s", "y"], ["s", "a", "l", "l", "e"], ["s",
   ↪    "l", "e", "e", "r"]], "horizontal_clues":
   ↪    [["tasks", "goals", "plans", "agend",
   ↪    "chores", "works", "deeds", "items",
   ↪    "lists", "brief"], ["motor", "power",
   ↪    "drive", "diesel", "steam", "pumps",
   ↪    "crank", "gears", "turbn", "motor"],
   ↪    ["grand", "artsy", "showy", "ornate",
   ↪    "fancy", "vain", "proud", "vogue", "swank",
   ↪    "luxus"], ["venue", "salle", "forum",
   ↪    "atria", "lobby", "parls", "court",
   ↪    "malls", "mall", "lobby"], ["jeer",
   ↪    "scoff", "sleer", "deris", "sneer",
   ↪    "scorn", "derid", "gibes", "gibed",
   ↪    "flout"]], "vertical_clues": [["amass",
   ↪    "stack", "hoard", "pile", "store", "heaps",
   ↪    "massy", "gathe", "lumps", "mound"],
   ↪    ["nilga", "goral", "eland", "lepus",
   ↪    "gazal", "kudu", "oryx", "gnu", "imps",
   ↪    "carb"], ["scheme", "design", "ettle",
   ↪    "nettle", "sting", "wiles", "plans",
   ↪    "ideas", "plots", "cocks"], ["spout",
   ↪    "nosle", "snout", "mouth", "nostr",
   ↪    "ports", "inlet", "vents", "outlt",
   ↪    "beaks"], ["drier", "arid", "sere",
   ↪    "parch", "dryer", "wring", "drear", "sear",
   ↪    "pall", "lack"]]}, {"state": [[null, null,
   ↪    null, null, null], ["r", "e", "v", "i",
   ↪    "e"], ["i", "g", "a", "l", "a"], ["s", "e",
   ↪    "d", "e", "r"], ["e", "t", "e", "r", "n"]],
   ↪    "horizontal_clues": [["parch", "dryup",
   ↪    "arefy", "wring", "suckd", "wizen",
   ↪    "desic", "evapo", "scald", "toast"],
   ↪    ["excel", "revie", "beat", "top", "best",
   ↪    "rise", "win", "lead", "rule", "boss"],
   ↪    ["igala", "tribe", "people", "race",
   ↪    "ethni", "nation", "yorub", "niger",
   ↪    "triba", "tribu"], ["seder", "meal",
   ↪    "food", "feast", "dine", "dish", "supper",
   ↪    "banqu", "treat", "fetes"], ["eterl",
   ↪    "etern", "everl", "forev", "immor",
   ↪    "endur", "const", "perma", "durab",
   ↪    "timeless"]], "vertical_clues": [["arise",
   ↪    "climb", "soar", "ascen", "mount", "leaps",
   ↪    "scale", "clamb", "steps", "jump"],
   ↪    ["regain", "renew", "recoi", "recla",
   ↪    "retri", "regra", "reget", "reapo",
   ↪    "reboo", "reset"], ["dodge", "elude",
   ↪    "shirk", "escap", "hide", "evade", "flee",
   ↪    "duck", "ditch", "evite"], ["filer",
   ↪    "files", "rasps", "grind", "blade",
   ↪    "sawer", "tool", "sharp", "knife",
   ↪    "metal"], ["yearn", "long", "ache",
   ↪    "crave", "desir", "need", "want", "thirst",
   ↪    "hunger", "lust"]]}, {"state": [[null,
   ↪    null, null, null, null], ["u", "r", "e",
   ↪    "n", "a"], ["f", "r", "i", "a", "r"], ["f",
   ↪    "o", "n", "g", "e"], ["o", "r", "g", "a",
   ↪    "l"]], "horizontal_clues": [["bebop",
   ↪    "jazzy", "music", "salsa", "swing",
   ↪    "blues", "riffs", "drums", "horns",
   ↪    "notes"], ["senna", "urena", "herbs",
   ↪    "flora", "mints", "trees", "leaves",
   ↪    "oils", "spice", "lavas"], ["monk",
   ↪    "friar", "nun", "saint", "clerk", "deity",
   ↪    "mystic", "faith", "pious", "sacra"],
   ↪    ["fetch", "carry", "fonge", "take",
   ↪    "seize", "hold", "grab", "earn", "gain",
   ↪    "yield"], ["tart", "argal", "orgal",
   ↪    "lemon", "sours", "wines", "taste",
```

Listing 9: crosswords_non_goal_states.jsonl

Successor Unit Test Successor unit test cases are stored in a jsonl file. The test cases used are depicted in Listing 10.

```
1    [
2        {
3            "state": [[null, null, "e", null,
         ↪   null], ["m", "o", "t", "o", "r"],
         ↪   [null, null, "t", null, null],
         ↪   [null, null, "l", null, null],
         ↪   [null, null, "e", null, null]],
4            "successors": [
5                [["a", "g", "e", "n", "d"], ["m",
             ↪   "o", "t", "o", "r"], [null,
             ↪   null, "t", null, null], [null,
             ↪   null, "l", null, null], [null,
             ↪   null, "e", null, null]],
6                [["d", "e", "e", "d", "s"], ["m",
             ↪   "o", "t", "o", "r"], [null,
             ↪   null, "t", null, null], [null,
             ↪   null, "l", null, null], [null,
             ↪   null, "e", null, null]],
7                [["i", "t", "e", "m", "s"], ["m",
             ↪   "o", "t", "o", "r"], [null,
             ↪   null, "t", null, null], [null,
             ↪   null, "l", null, null], [null,
             ↪   null, "e", null, null]],
8                [[null, null, "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   ["a", "r", "t", "s", "y"],
             ↪   [null, null, "l", null, null],
             ↪   [null, null, "e", null, null]],
9                [[null, null, "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, null],
             ↪   ["s", "a", "l", "l", "e"],
             ↪   [null, null, "e", null, null]],
10               [[null, null, "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, null],
             ↪   ["m", "a", "l", "l", "s"],
             ↪   [null, null, "e", null, null]],
11               [[null, null, "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, null],
             ↪   [null, null, "l", null, null],
             ↪   ["s", "l", "e", "e", "r"]],
12               [[null, null, "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, null],
             ↪   [null, null, "l", null, null],
             ↪   ["s", "n", "e", "e", "r"]],
13               [["a", null, "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   ["a", null, "t", null, null],
             ↪   ["s", null, "l", null, null],
             ↪   ["s", null, "e", null, null]],
14               [[null, "g", "e", null, null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, "r", "t", null, null],
             ↪   [null, "a", "l", null, null],
             ↪   [null, "l", "e", null, null]],
15               [[null, null, "e", "n", null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", "s", null],
             ↪   [null, null, "l", "l", null],
             ↪   [null, null, "e", "e", null]],
16               [[null, null, "e", "m", null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", "u", null],
             ↪   [null, null, "l", "t", null],
             ↪   [null, null, "e", "h", null]],
17               [[null, null, "e", "n", null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", "s", null],
             ↪   [null, null, "l", "t", null],
             ↪   [null, null, "e", "r", null]],
18               [[null, null, "e", "p", null],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", "r", null],
             ↪   [null, null, "l", "t", null],
             ↪   [null, null, "e", "s", null]],
19               [[null, null, "e", null, "d"],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, "i"],
             ↪   [null, null, "l", null, "e"],
             ↪   [null, null, "e", null, "r"]],
20               [[null, null, "e", null, "d"],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, "y"],
             ↪   [null, null, "l", null, "e"],
             ↪   [null, null, "e", null, "r"]],
21               [[null, null, "e", null, "w"],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, "i"],
             ↪   [null, null, "l", null, "n"],
             ↪   [null, null, "e", null, "g"]],
22               [[null, null, "e", null, "d"],
             ↪   ["m", "o", "t", "o", "r"],
             ↪   [null, null, "t", null, "e"],
             ↪   [null, null, "l", null, "a"],
             ↪   [null, null, "e", null, "r"]]
23           ],
24           "horizontal_clues": [["tasks", "goals",
         ↪   "plans", "agend", "chores",
         ↪   "works", "deeds", "items", "lists",
         ↪   "brief"], ["motor", "power",
         ↪   "drive", "diesel", "steam",
         ↪   "pumps", "crank", "gears", "turbn",
         ↪   "motor"], ["grand", "artsy",
         ↪   "showy", "ornate", "fancy", "vain",
         ↪   "proud", "vogue", "swank",
         ↪   "luxus"], ["venue", "salle",
         ↪   "forum", "atria", "lobby", "parls",
         ↪   "court", "malls", "mall", "lobby"],
         ↪   ["jeer", "scoff", "sleer", "deris",
         ↪   "sneer", "scorn", "derid", "gibes",
         ↪   "gibed", "flout"]],
```

```
25          "vertical_clues": [["amass", "stack",
     ↪    "hoard", "pile", "store", "heaps",
     ↪    "massy", "gathe", "lumps",
     ↪    "mound"], ["nilga", "goral",
     ↪    "eland", "lepus", "gazal", "kudu",
     ↪    "oryx", "gnu", "imps", "carb"],
     ↪    ["scheme", "design", "ettle",
     ↪    "nettle", "sting", "wiles",
     ↪    "plans", "ideas", "plots",
     ↪    "cocks"], ["spout", "nosle",
     ↪    "snout", "mouth", "nostr", "ports",
     ↪    "inlet", "vents", "outlt",
     ↪    "beaks"], ["drier", "arid", "sere",
     ↪    "parch", "dryer", "wring", "drear",
     ↪    "sear", "pall", "lack"]]}]
26   [
27       {
28          "state": [[null, null, "e", null,
     ↪    null], ["r", "e", "v", "i", "e"],
     ↪    [null, null, "a", null, null],
     ↪    [null, null, "d", null, null],
     ↪    [null, null, "e", null, null]],
29          "successors": [
30              [["a", "r", "e", "f", "y"], ["r",
     ↪        "e", "v", "i", "e"], [null,
     ↪        null, "a", null, null], [null,
     ↪        null, "d", null, null], [null,
     ↪        null, "e", null, null]],
31              [[null, null, "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        ["i", "g", "a", "l", "a"],
     ↪        [null, null, "d", null, null],
     ↪        [null, null, "e", null, null]],
32              [[null, null, "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", null, null],
     ↪        ["s", "e", "d", "e", "r"],
     ↪        [null, null, "e", null, null]],
33              [[null, null, "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", null, null],
     ↪        [null, null, "d", null, null],
     ↪        ["e", "t", "e", "r", "l"]]],
34              [[null, null, "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", null, null],
     ↪        [null, null, "d", null, null],
     ↪        ["e", "t", "e", "r", "n"]]],
35              [[null, null, "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", null, null],
     ↪        [null, null, "d", null, null],
     ↪        ["e", "v", "e", "r", "l"]]],
36              [["a", null, "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        ["i", null, "a", null, null],
     ↪        ["s", null, "d", null, null],
     ↪        ["e", null, "e", null, null]],
37              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "n", "a", null, null],
     ↪        [null, "e", "d", null, null],
     ↪        [null, "w", "e", null, null]],
38              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "c", "a", null, null],
     ↪        [null, "o", "d", null, null],
     ↪        [null, "i", "e", null, null]],
39              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "c", "a", null, null],
     ↪        [null, "l", "d", null, null],
     ↪        [null, "a", "e", null, null]],
40              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "t", "a", null, null],
     ↪        [null, "r", "d", null, null],
     ↪        [null, "i", "e", null, null]],
41              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "g", "a", null, null],
     ↪        [null, "r", "d", null, null],
     ↪        [null, "a", "e", null, null]],
42              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "g", "a", null, null],
     ↪        [null, "e", "d", null, null],
     ↪        [null, "t", "e", null, null]],
43              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "a", "a", null, null],
     ↪        [null, "p", "d", null, null],
     ↪        [null, "o", "e", null, null]],
44              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "b", "a", null, null],
     ↪        [null, "o", "d", null, null],
     ↪        [null, "o", "e", null, null]],
45              [[null, "r", "e", null, null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, "s", "a", null, null],
     ↪        [null, "e", "d", null, null],
     ↪        [null, "t", "e", null, null]],
46              [[null, null, "e", "f", null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", "l", null],
     ↪        [null, null, "d", "e", null],
     ↪        [null, null, "e", "r", null]],
47              [[null, null, "e", "f", null],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", "l", null],
     ↪        [null, null, "d", "e", null],
     ↪        [null, null, "e", "s", null]],
48              [[null, null, "e", null, "y"],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", null, "a"],
     ↪        [null, null, "d", null, "r"],
     ↪        [null, null, "e", null, "n"]],
49              [[null, null, "e", null, "d"],
     ↪        ["r", "e", "v", "i", "e"],
     ↪        [null, null, "a", null, "s"],
     ↪        [null, null, "d", null, "i"],
     ↪        [null, null, "e", null, "r"]]
50          ],
```

```
51          "horizontal_clues": [["parch", "dryup",
     ↪    "arefy", "wring", "suckd", "wizen",
     ↪    "desic", "evapo", "scald",
     ↪    "toast"], ["excel", "revie",
     ↪    "beat", "top", "best", "rise",
     ↪    "win", "lead", "rule", "boss"],
     ↪    ["igala", "tribe", "people",
     ↪    "race", "ethni", "nation", "yorub",
     ↪    "niger", "triba", "tribu"],
     ↪    ["seder", "meal", "food", "feast",
     ↪    "dine", "dish", "supper", "banqu",
     ↪    "treat", "fetes"], ["eterl",
     ↪    "etern", "everl", "forev", "immor",
     ↪    "endur", "const", "perma", "durab",
     ↪    "timeless"]],
52          "vertical_clues": [["arise", "climb",
     ↪    "soar", "ascen", "mount", "leaps",
     ↪    "scale", "clamb", "steps", "jump"],
     ↪    ["regain", "renew", "recoi",
     ↪    "recla", "retri", "regra", "reget",
     ↪    "reapo", "reboo", "reset"],
     ↪    ["dodge", "elude", "shirk",
     ↪    "escap", "hide", "evade", "flee",
     ↪    "duck", "ditch", "evite"],
     ↪    ["filer", "files", "rasps",
     ↪    "grind", "blade", "sawer", "tool",
     ↪    "sharp", "knife", "metal"],
     ↪    ["yearn", "long", "ache", "crave",
     ↪    "desir", "need", "want", "thirst",
     ↪    "hunger", "lust"]]
53      }
54  ]
55  [
56      {
57          "state": [[null, null, "b", null,
     ↪    null], ["u", "r", "e", "n", "a"],
     ↪    [null, null, "i", null, null],
     ↪    [null, null, "n", null, null],
     ↪    [null, null, "g", null, null]],
58          "successors": [
59              [["b", "e", "b", "o", "p"], ["u",
     ↪        "r", "e", "n", "a"], [null,
     ↪        null, "i", null, null], [null,
     ↪        null, "n", null, null], [null,
     ↪        null, "g", null, null]],
60              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        ["f", "r", "i", "a", "r"],
     ↪        [null, null, "n", null, null],
     ↪        [null, null, "g", null, null]],
61              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        ["s", "a", "i", "n", "t"],
     ↪        [null, null, "n", null, null],
     ↪        [null, null, "g", null, null]],
62              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        ["d", "e", "i", "t", "y"],
     ↪        [null, null, "n", null, null],
     ↪        [null, null, "g", null, null]],
63              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        ["f", "a", "i", "t", "h"],
     ↪        [null, null, "n", null, null],
     ↪        [null, null, "g", null, null]],
64              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        [null, null, "i", null, null],
     ↪        ["f", "o", "n", "g", "e"],
     ↪        [null, null, "g", null, null]],
65              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        [null, null, "i", null, null],
     ↪        [null, null, "n", null, null],
     ↪        ["a", "r", "g", "a", "l"]],
66              [[null, null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        [null, null, "i", null, null],
     ↪        [null, null, "n", null, null],
     ↪        ["o", "r", "g", "a", "l"]],
67              [["b", null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        ["f", null, "i", null, null],
     ↪        ["f", null, "n", null, null],
     ↪        ["o", null, "g", null, null]],
68              [["h", null, "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        ["m", null, "i", null, null],
     ↪        ["o", null, "n", null, null],
     ↪        ["r", null, "g", null, null]],
69              [[null, "e", "b", null, null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        [null, "r", "i", null, null],
     ↪        [null, "o", "n", null, null],
     ↪        [null, "r", "g", null, null]],
70              [[null, null, "b", "o", null],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        [null, null, "i", "a", null],
     ↪        [null, null, "n", "g", null],
     ↪        [null, null, "g", "a", null]],
71              [[null, null, "b", null, "p"],
     ↪        ["u", "r", "e", "n", "a"],
     ↪        [null, null, "i", null, "r"],
     ↪        [null, null, "n", null, "e"],
     ↪        [null, null, "g", null, "l"]]
72          ],
73          "horizontal_clues": [["bebop", "jazzy",
     ↪    "music", "salsa", "swing", "blues",
     ↪    "riffs", "drums", "horns",
     ↪    "notes"], ["senna", "urena",
     ↪    "herbs", "flora", "mints", "trees",
     ↪    "leaves", "oils", "spice",
     ↪    "lavas"], ["monk", "friar", "nun",
     ↪    "saint", "clerk", "deity",
     ↪    "mystic", "faith", "pious",
     ↪    "sacra"], ["fetch", "carry",
     ↪    "fonge", "take", "seize", "hold",
     ↪    "grab", "earn", "gain", "yield"],
     ↪    ["tart", "argal", "orgal", "lemon",
     ↪    "sours", "wines", "taste", "tangs",
     ↪    "zesty", "acid"]],
```

```
        "vertical_clues": [["buffo", "clown",
        ↪  "actor", "joker", "wit", "humor",
        ↪  "silly", "gag", "role", "fool"],
        ↪  ["error", "fault", "flaw", "slip",
        ↪  "oops", "blips", "bugs", "glitch",
        ↪  "bugs", "boob"], ["being", "alive",
        ↪  "human", "being", "exist", "life",
        ↪  "creed", "soul", "love", "kind"],
        ↪  ["fishy", "onaga", "ruby", "salmo",
        ↪  "tuna", "sushi", "prawn", "trout",
        ↪  "shrim", "codex"], ["dress",
        ↪  "appar", "parel", "gowns", "style",
        ↪  "drape", "shirts", "veils",
        ↪  "outfi", "apron"]]
    }
]
```

Listing 10: crossword_successors.jsonl

Partial Successor Soundness Test

```python
def validate_transition_complex(s, t):
    def count_none(s):
        ns = 0
        for r in s:
            ns += len([c for c in r if c is None])
            return ns

    ns = count_none(s)
    nt = count_none(t)

    feedback = ""
    if ns < nt:
        # More unknown
        feedback += prettyprint("Successor state has
        ↪  less filled cells than the parent state.")
    elif ns == nt:
        # Same unknown
        feedback += prettyprint("Successor state has the
        ↪  same number of filled cells as the parent
        ↪  state.")
    elif ns - nt > 5:
        # Way too many less unknown
        feedback += prettyprint("Successor state has
        ↪  more than 5 filled cells more than the
        ↪  parent state.")
    else:
        return True, ""

    feedback += prettyprint("Let's think step by step.
    ↪  First, think what you did wrong.")
    feedback += prettyprint("Then, think of in what ways
    ↪  successor state should be different from the
    ↪  parent state.")
    feedback += prettyprint("Then, provide the complete
    ↪  Python code for the revised successor function
    ↪  that returns a list of successor states.")
    feedback += prettyprint("Remember how you fixed the
    ↪  previous mistakes, if any. Keep the same
    ↪  function signature.")
    return False, feedback
```

A.4 ProntoQA

Goal Unit Test   Goal unit test cases are stored in two *jsonl* files, one for goal states and one for non-goal states.

Listing 11: prontoqa_goal_states.jsonl

```
1   {"state": ["painted lady", "bony"], "goal": "bony"}
2   {"state": ["mersenne prime", "real"], "goal": "real"}
3   {"state": ["lepidopteran", "small"], "goal": "small"}
```

Listing 12: prontoqa_non_goal_states.jsonl

```
1   {"state": ["painted lady"], "goal": "not-bony"}
2   {"state": ["mersenne prime"], "goal": "not-real"}
3   {"state": ["lepidopteran"], "goal": "not-small"}
```

Successor Unit Test   Successor unit test cases are stored in a jsonl file. The test cases used are depicted in Listing 13.

```
1   {"state": ["painted lady"], "rules":
    ↪  [["arthropod", "protostome"],
    ↪  ["lepidopteran", "insect"], ["painted
    ↪  lady", "butterfly"], ["insect",
    ↪  "arthropod"], ["invertebrate", "animal"],
    ↪  ["arthropod", "not-bony"], ["protostome",
    ↪  "invertebrate"], ["whale", "bony"],
    ↪  ["butterfly", "lepidopteran"], ["animal",
    ↪  "multicellular"], ["insect",
    ↪  "six-legged"]], "successors": [["painted
    ↪  lady", "butterfly"]]}
2   {"state": ["mersenne prime"], "rules":
    ↪  [["integer", "real number"], ["prime
    ↪  number", "natural number"], ["real number",
    ↪  "number"], ["mersenne prime", "prime
    ↪  number"], ["mersenne prime",
    ↪  "not-composite"], ["natural number",
    ↪  "integer"], ["imaginary number",
    ↪  "not-real"], ["real number", "real"],
    ↪  ["prime number", "not-composite"],
    ↪  ["natural number", "positive"]],
    ↪  "successors": [["prime number", "mersenne
    ↪  prime"], ["not-composite", "mersenne
    ↪  prime"]]}
3   {"state": ["lepidopteran"], "rules":
    ↪  [["lepidopteran", "insect"], ["arthropod",
    ↪  "small"], ["insect", "arthropod"],
    ↪  ["whale", "not-small"], ["invertebrate",
    ↪  "animal"], ["butterfly", "lepidopteran"],
    ↪  ["arthropod", "invertebrate"], ["animal",
    ↪  "multicellular"], ["insect",
    ↪  "six-legged"]], "successors": [["insect",
    ↪  "lepidopteran"]]}
```

Listing 13: prontoqa_successors.jsonl

Partial Successor Soundness Test

```python
def validate_transition_complex(s, t):
    if s == t:
        return True, ""
    elif len(t) - len(s) != 1:
        feedback = prettyprint("Invalid
            ↪ transition: length mismatch
            ↪ - the length of a successor
            ↪ must be one more than the
            ↪ parent.")
        feedback += prettyprint("Let's
            ↪ think step by step. First
            ↪ think through in words why
            ↪ the successor function
            ↪ produced a successor that
            ↪ had a length that was not
            ↪ exactly one more than the
            ↪ parent. Then provide the
            ↪ complete Python code for
            ↪ the revised successor
            ↪ function that ensures the
            ↪ length of a successor is
            ↪ exactly one more than the
            ↪ parent.")
        feedback +=
            ↪ prettyprint("Remember how
            ↪ you fixed the previous
            ↪ mistakes, if any. Keep the
            ↪ same function signature.")
        return False, feedback
    return True, ""
```

A.5 Sokoban

Goal Unit Test Goal unit test cases are stored in two *jsonl* files, one for goal states and one for non-goal states.

Listing 14: sokoban_goal_states.jsonl

```
1  {"state": {"at-player": [5, 3], "at-stone":
   ↪ [[3, 3], [4, 3]]}, "grid": [[1, 1, 1, 1, 1,
   ↪ 1], [1, 0, 0, 0, 0, 1], [1, 0, 1, 0, 0, 1],
   ↪ [1, 0, 0, 2, 0, 1], [1, 0, 1, 2, 1, 1], [1,
   ↪ 0, 0, 0, 1, 0], [1, 1, 1, 1, 1, 0]]}
2  {"state": {"at-player": [5, 2], "at-stone":
   ↪ [[3, 2], [4, 2]]}, "grid": [[1, 0, 1, 1, 1,
   ↪ 1, 1], [0, 0, 1, 0, 0, 0, 1], [1, 1, 1, 0,
   ↪ 0, 0, 1], [1, 0, 2, 0, 0, 0, 1], [1, 0, 2,
   ↪ 1, 0, 0, 1], [1, 0, 0, 1, 0, 0, 1], [1, 1,
   ↪ 1, 1, 1, 1]]}
3  {"state": {"at-player": [4, 4], "at-stone":
   ↪ [[2, 2], [3, 3]]}, "grid": [[1, 1, 1, 1, 0,
   ↪ 0, 0, 0], [1, 0, 0, 1, 1, 0, 0, 0], [1, 0,
   ↪ 2, 0, 1, 1, 1, 1], [1, 0, 0, 2, 1, 0, 0,
   ↪ 1], [1, 1, 0, 0, 0, 0, 0, 1], [0, 1, 1, 1,
   ↪ 0, 0, 0, 1], [0, 0, 0, 1, 0, 1, 0, 1], [0,
   ↪ 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 1, 1, 1, 1,
   ↪ 1]]}
```

Listing 15: sokoban_non_goal_states.jsonl

```
1  {"state": {"at-player": [5, 3], "at-stone":
   ↪ [[3, 3], [4, 3]]}, "grid": [[1, 1, 1, 1, 1,
   ↪ 1], [1, 0, 0, 2, 0, 1], [1, 0, 1, 0, 0, 1],
   ↪ [1, 0, 0, 0, 0, 1], [1, 0, 1, 2, 1, 1], [1,
   ↪ 0, 0, 0, 1, 0], [1, 1, 1, 1, 1, 0]]}
2  {"state": {"at-player": [5, 2], "at-stone":
   ↪ [[3, 2], [4, 2]]}, "grid": [[1, 0, 1, 1, 1,
   ↪ 1, 1], [0, 0, 1, 0, 0, 0, 1], [1, 1, 1, 0,
   ↪ 0, 2, 1], [1, 0, 0, 0, 0, 0, 1], [1, 0, 0,
   ↪ 1, 0, 2, 1], [1, 0, 0, 1, 0, 0, 1], [1, 1,
   ↪ 1, 1, 1, 1]]}
3  {"state": {"at-player": [4, 4], "at-stone":
   ↪ [[2, 2], [3, 3]]}, "grid": [[1, 1, 1, 1, 0,
   ↪ 0, 0, 0], [1, 0, 0, 1, 1, 0, 0, 0], [1, 0,
   ↪ 0, 0, 1, 1, 1, 1], [1, 0, 0, 0, 1, 0, 0,
   ↪ 1], [1, 1, 0, 0, 0, 0, 0, 1], [0, 1, 1, 1,
   ↪ 2, 2, 0, 1], [0, 0, 0, 1, 0, 1, 0, 1], [0,
   ↪ 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 1, 1, 1, 1,
   ↪ 1]]}
```

Successor Unit Test Successor unit test cases are stored in a jsonl file. The test cases used are depicted in Listing 16.

```
1  {"state": {"at-player": [5, 3], "at-stone":
   ↪ [[3, 3], [4, 3]]}, "successors":
   ↪ [{"at-player": [5, 2], "at-stone": [[3, 3],
   ↪ [4, 3]]}], "grid": [[1, 1, 1, 1, 1, 1], [1,
   ↪ 0, 0, 2, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0,
   ↪ 0, 0, 0, 1], [1, 0, 1, 2, 1, 1], [1, 0, 0,
   ↪ 0, 1, 0], [1, 1, 1, 1, 1, 0]]}
2  {"state": {"at-player": [5, 2], "at-stone":
   ↪ [[3, 2], [4, 2]]}, "successors":
   ↪ [{"at-player": [5, 1], "at-stone": [[3, 2],
   ↪ [4, 2]]}], "grid": [[1, 0, 1, 1, 1, 1, 1],
   ↪ [0, 0, 1, 0, 0, 0, 1], [1, 1, 1, 0, 0, 2,
   ↪ 1], [1, 0, 0, 0, 0, 0, 1], [1, 0, 0, 1, 0,
   ↪ 2, 1], [1, 0, 0, 1, 0, 0, 1], [1, 1, 1, 1,
   ↪ 1, 1, 1]]}
3  {"state": {"at-player": [4, 4], "at-stone":
   ↪ [[2, 2], [3, 3]]}, "successors":
   ↪ [{"at-player": [5, 4], "at-stone": [[2, 2],
   ↪ [3, 3]]}, {"at-player": [4, 3], "at-stone":
   ↪ [[2, 2], [3, 3]]}, {"at-player": [4, 5],
   ↪ "at-stone": [[2, 2], [3, 3]]}], "grid":
   ↪ [[1, 1, 1, 1, 0, 0, 0, 0], [1, 0, 0, 1, 1,
   ↪ 0, 0, 0], [1, 0, 0, 0, 1, 1, 1, 1], [1, 0,
   ↪ 0, 0, 1, 0, 0, 1], [1, 1, 0, 0, 0, 0, 0,
   ↪ 1], [0, 1, 1, 1, 2, 2, 0, 1], [0, 0, 0, 1,
   ↪ 0, 1, 0, 1], [0, 0, 0, 1, 0, 0, 0, 1], [0,
   ↪ 0, 0, 1, 1, 1, 1, 1]]}
4  {"state": {"at-player": [5, 3], "at-stone":
   ↪ [[5, 2], [4, 3]]}, "successors":
   ↪ [{"at-player": [5, 2], "at-stone": [[5, 1],
   ↪ [4, 3]]}], "grid": [[1, 1, 1, 1, 1, 1], [1,
   ↪ 0, 0, 2, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0,
   ↪ 0, 0, 1], [1, 0, 1, 2, 1, 1], [1, 0, 0,
   ↪ 0, 1, 0], [1, 1, 1, 1, 1, 0]]}
```

Listing 16: sokoban_successors.jsonl

Partial Successor Soundness Test

```python
def validate_transition_complex(s, t):
    locations = set(t['at-stone'])
    if len(locations) <
    ↪  len(t['at-stone']):
        feedback = prettyprint("Invalid
        ↪  transition: multiple stones
        ↪  at the same location.")
        feedback += prettyprint("Let's
        ↪  think step by step. First
        ↪  think through in words why
        ↪  the successor function
        ↪  produced a successor that
        ↪  has two stones at the same
        ↪  location. Then provide the
        ↪  complete Python code for
        ↪  the revised successor
        ↪  function that ensures that
        ↪  in all successors all
        ↪  stones are at different
        ↪  locations.")
        feedback +=
        ↪  prettyprint("Remember how
        ↪  you fixed the previous
        ↪  mistakes, if any. Keep the
        ↪  same function signature.")
        return False, feedback
    if t['at-player'] in locations:
        feedback = prettyprint("Invalid
        ↪  transition: a stone and the
        ↪  player are at the same
        ↪  location.")
        feedback += prettyprint("Let's
        ↪  think step by step. First
        ↪  think through in words why
        ↪  the successor function
        ↪  produced a successor that
        ↪  has a stone and the player
        ↪  at the same location. Then
        ↪  provide the complete Python
        ↪  code for the revised
        ↪  successor function that
        ↪  ensures that in all
        ↪  successors the player and
        ↪  the stones are at different
        ↪  locations.")
        feedback +=
        ↪  prettyprint("Remember how
        ↪  you fixed the previous
        ↪  mistakes, if any. Keep the
        ↪  same function signature.")
        return False, feedback
    return True, ""
```