Automating Thought of Search: A Journey Towards Soundness and Completeness (Student Abstract)

Daniel Cao[1], **Michael Katz**[2], **Harsha Kokel**[2], **Kavitha Srinivas**[2], **Shirin Sohrabi**[2]
[1]Cornell University,
[2]IBM Research
dyc33@cornell.edu, {Michael.Katz1, Harsha.Kokel, Kavitha.Srinivas, ssohrab}@ibm.com

Abstract

Large language models (LLMs) now turn their attention to search. Recently, Thought of Search (ToS) proposed defining the search space with code, having an LLM produce that code. ToS requires a *human in the loop*, collaboratively producing a sound successor function and goal test, achieving impressive 100% accuracy on all the tested datasets. 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.

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. 2023; Yao et al. 2023; 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 24Game (Yao et al. 2023).

- Examples of inputs to $isgoal$ for which the correct output is known. E.g., we know that $isgoal([24])$ should be true and $isgoal([24, 1])$ should be false.

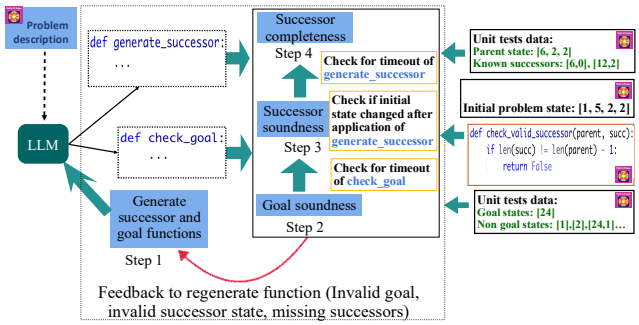

Figure 1: An overview of AutoToS.

- Examples of inputs to $succ$ for which some of the correct outputs are known. E.g., we know that [24], [2], and [-2] are valid successors of [6,4] and 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 24Game 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 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

Listing 1: **24Game** 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.

		24Game	PrOntoQA	Sokoban	XWord	Blocks
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: Average number of calls to the LLM per domain.

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.

Experiments

We conduct experiments with a representative collection of five search/planning problems: BlocksWorld (Gupta and Nau 1992), PrOntoQA (Hao et al. 2023), Mini XWord and 24Game (Yao et al. 2023), and Sokoban (Junghanns and Schaeffer 1997). We test the performance of 3 LLMs families, GPT-4o Llama3.1, and DeepSeek. We follow the experimental setup of Katz et al. (2024).

Our evaluation shows that (i) the partial soundness test and the completeness step improve the accuracy of AutoToS (Figure 2), (ii) the number of calls to LLM is comparable to ToS (Table 1), and (iii) there is no single model that performs better than all other. Interestingly, the smaller model GPT-4o-mini performs quite well in terms of accuracy.

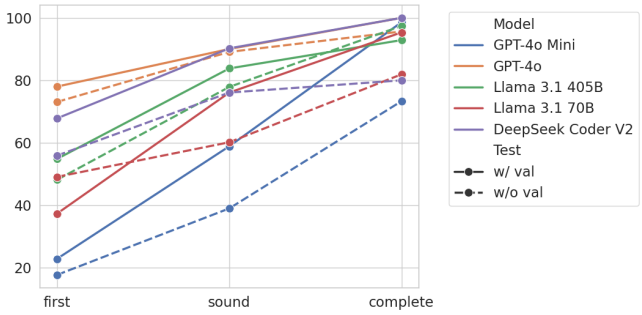

Figure 2: Progression of accuracy values during AutoToS.

Error Categories

We distinguish 10 error categories with 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.

No errors observed in the last two categories, only 1, 2, and 3 errors in categories 6, 8, and 7, respectively. DeepSeek rarely produces code that triggers exception or changes the input state, and typically passes the goal soundness test. Other models, especially the smaller ones, are more diverse in errors produced. Across all models, most errors account for the failed successor soundness and completeness tests.

Conclusions

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 domains. With just a few LLM calls, 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.

Acknowledgments

The work was performed while Daniel Cao was at IBM Research.

References

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 Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 17682–17690. AAAI Press.

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. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*.

Junghanns, A.; and Schaeffer, J. 1997. Sokoban: A Challenging Single-Agent Search Problem. In Pollack, M. E., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 1997)*. Morgan Kaufmann.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*.

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.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023. 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)*.