Guiding Enumerative Program Synthesis with Large Language Models

Yixuan Li[1], Julian Parsert[2], and Elizabeth Polgreen[1]

[1] University of Edinburgh, Edinburgh, UK.
[2] University of Oxford, Oxford, UK.

Pre-trained Large Language Models (LLMs) are beginning to dominate the discourse around automatic code generation with natural language specifications [5]. In contrast, the best-performing synthesizers in the domain of formal synthesis, where the synthesizer must generate code that satisfies a precise logical specification, are still based on enumerative algorithms [11, 8, 3, 7]. In this work, we evaluate the abilities of LLMs to solve formal synthesis benchmarks by carefully crafting a library of prompts for the domain, and propose two methods for integrating the syntactic guidance from an LLM into an enumerative synthesis loop.

We hypothesize that, in the cases where the LLM returns only incorrect solutions, the correct solutions are most often in the vicinity of the incorrect solutions, and that, by searching in the neighborhood of the incorrect solutions, we may be able to guide an enumerative synthesize to find a solution faster. We present two methods for integrating syntactic guidance from LLMs into an enumerative CounterExample Guided Inductive Synthesis (CEGIS) [12] algorithm. CEGIS describes enumeration search algorithms where an enumerator enumerates solution candidates from a context-free grammar with the help of a set of counter-examples, and a verifier is queried to check enumerated solution candidates and produce counter-examples in case of failure. The first method, pCFG-synth, prompts an LLM for solutions to the benchmark and generates a probabilistic context-free grammar (pCFG) from these solutions. It then deploys an enumerative synthesizer biased by the weights on this pre-trained pCFG [9, 10]. The second method, iLLM-synth, integrates the prompting within the enumerative synthesizer. Instead of asking the LLM to provide a full solution, we ask it to provide helper functions to help "a student" complete a partially enumerated program. We use the responses to augment the set of production rules in the grammar and update the weights across the existing production rules in the pCFG during enumeration. We implement two variants of both approaches, using two different enumeration algorithms: a probabilistic top-down enumerator [1] and a weighted search based on the A^* algorithm [6, 8].

We evaluated our techniques on benchmarks from the Syntax-Guided Synthesis (SyGuS) [2] competition, and found that LLMs and enumerative solvers have distinct strengths and weaknesses when deployed alone. By combining the LLM and enumeration in pCFG-synth, we can combine these strengths and weaknesses and outperform cvc5 [4], the winning SyGuS competition tool, solving 73 more benchmarks. Integrating the LLM via iLLM-synth gives greater performance gains over a basic enumerator than using the pre-trained pCFG, and demonstrates that, by allowing the enumerative synthesizer to prompt the LLM with information obtained during the enumeration and allowing the LLM to provide syntactic feedback to the enumeration, we can achieve performance that equals and exceeds the state-of-the-art solvers, even with relatively simple enumerative algorithms. We argue that our results show that LLMs have the potential to make significant contributions in the domain of formal program synthesis but the way to achieve this is by combining these techniques with existing algorithms in the literature.

References

[1] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. *Syntax-guided synthesis*. IEEE, 2013.

[2] Rajeev Alur, Dana Fisman, Rishabh Singh, and Abhishek Udupa. Syntax guided synthesis competition. https://sygus-org.github.io, 2017. Accessed: 2024-01-16.

[3] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. Scaling enumerative program synthesis via divide and conquer. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 319–336, 2017.

[4] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS (1)*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[6] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968.

[7] Kangjing Huang, Xiaokang Qiu, Peiyuan Shen, and Yanjun Wang. Reconciling enumerative and deductive program synthesis. In *PLDI*, pages 1159–1174. ACM, 2020.

[8] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. In *PLDI*, pages 436–449. ACM, 2018.

[9] Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In *ICML*, pages 639–646. Citeseer, 2010.

[10] Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. A machine learning framework for programming by example. In *International Conference on Machine Learning*, pages 187–195. PMLR, 2013.

[11] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. cvc4sy: Smart and fast term enumeration for syntax-guided synthesis. In *CAV (2)*, volume 11562 of *Lecture Notes in Computer Science*, pages 74–83. Springer, 2019.

[12] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 404–415, 2006.