Extending Program Synthesis Grammar with Subprograms Learned from Solutions to Simpler Problems

Program synthesis is the process of automatically producing a program which can satisfy a high level specification [3]. It is a hard problem which can take a long time using brute force algorithms.

One of the ways to optimize this is to limit the search space, so the synthesizer can reach a solution faster if one exists. One common way of doing this is to reduce the allowed degree of complexity in the produced program, which can be done by reducing the allowed nesting height of the statements.

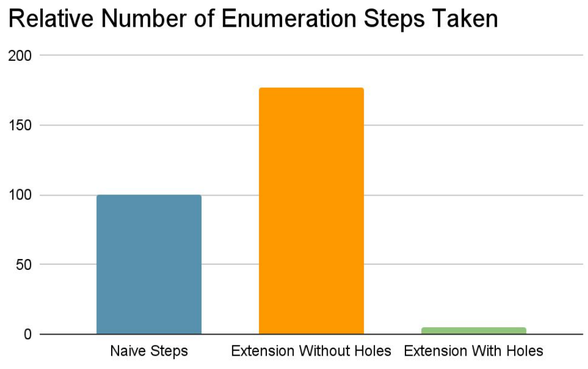
This creates the problem of not being able to solve overly complex specifications, because they would require a more complex program than the above limitation imposes. To allow a solution to such problems while keeping the limitation, one can provide the algorithm with more abstractions [2]. This research explores ways of doing this automatically, via extending the program from common parts of the solutions to simpler problems, before tackling the complex problem.

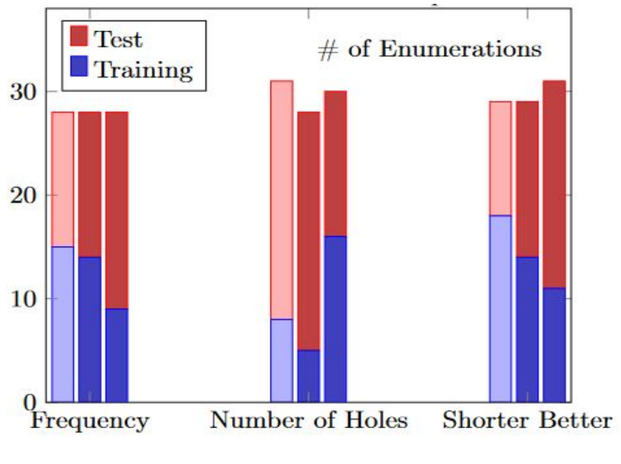
Research Questions

- How to pick useful components out of a program?
- How to evaluate the equivalence and similarity of different program snippets?
- How to introduce useful program snippets into a program synthesis program for further, possibly more performant synthesis?

Experiments and Results

- Effect on number of solved problems with limited number of enumeration steps once the grammar is extended.
- Effect of allowing and not allowing replaceable parts in the extensions.
- Effect of heuristics used for choosing grammar extensions. Choosing shorter subprograms, more often encountered programs, or programs with more substitutable parts, termed holes.
- Performance comparison to other subproblem solution unification methods proposed by other members of the research project team (anti-unification, splitting grammars, etc.). The proposed method in this research was not the best because of excessive enumeration with not very useful grammar extensions for some classes of problems.





Proposed Algorithm Starting Grammar	ListOrNumber -> 1 | 2 | 3 … 13 ListOrNumber -> x ListOrNumber -> times ListOrNumber ListOrNumber ListOrNumber -> head ListOrNumber ListOrNumber -> tail ListOrNumber ListOrNumber -> cons ListOrNumber ListOrNumber | nil Give different sets of examples, termed tasks breadth-first-search, max search depth = 4	
Examples	[2, 4, 10] -> 12 [3, 5, 8] -> 15 [2, 3, 4] -> 9	[2, 4, 10] -> 8 [3, 5, 8] -> 10 [2, 3, 4] -> 6
Program	(times 3 (head (tail x))) [depth 4]	(times 2 (head (tail x))) [depth 4]
Common Pattern- Grammar Extension	ListOrNumber -> second-item = (head (tail x))	
More Examples	[2, 4, 10] -> [2, 4] [4, 2, 8] -> [4, 2] [8, 5, 13] -> [8, 5]	
Program	Extended grammar: (cons (head x) (cons second-item nil)) [depth 3] Original grammar: (cons (head x) (cons (head (tail x)) nil)) [depth 5]	

Future Work

- Use of equivalence graphs for subprograms, as proposed by Detlefs, et al [1], in order to improve the conciseness of produced programs.
- The algorithm can be made into a program iterators as the ones found in Herb.jl in order to implement a system that refines itself as more simpler problems are solved.

Source Code

github.com/boraini/HerbAutomaticAbstraction

Contact the Author

Mert Bora İnevi (5540488)
Delft University of Technology
CSE3000 Research Project Group 28
m.b.inevi@student.tudelft.nl
https://boraini.com

References

[1] David Detlefs, Greg Nelson, and James Saxe. Simplify: A theorem prover for program checking. *Journal of the ACM,* 52, 09 2003.

[2] Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Cary, L., Solar-Lezama, A., & Tenenbaum, J. B. (2021). Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation,* 835–850.

[3] Gulwani, S., Polozov, O., & Singh, R. (2017). Program synthesis. *Foundations and Trends in Programming Languages,* 4 (1-2), 1–119.

[4] *Bootstrap Icons.* https://icons.getbootstrap.com/