Author: Radu Nicolae
Supervisor: Reuben Gardos Reid
Professor: Sebastijan Dumančić
Email: R.Nicolae@student.tudeflt.nl

Efficient Program Synthesis via Anti-Unification

How do we use anti-unification to combine programs to act as a starting point for synthesis?

1 Introduction

- Program synthesis is the automated process of creating a program based on user intent [1]
- Anti-unification is the problem of finding the most specific template (pattern) of two terms [2]
- Traditional synthesizers use domain-specific languages (DSLs) to guide the search process
- By enhancing the grammar that a DSL-based synthesizer utilizes, the search procedure could be streamlined
- To achieve this objective, we propose an approach, called Anti-unification Meta-iterator, that uses anti-unification to introduce common patterns from programs into the DSL

2 Methodology

- Starting with a search method that iterates through the state space to generate program, we collect outputs that partially satisfy the user intent
- Using anti-unification, common patterns between the programs collected are found and a list of candidate additions to the DSL is created
- The list of common patterns is converted in new rules and introduced into the DSL
- The process is repeated until a program that fully satisfies the user intent is found

3 Results

- We aim to evaluate the performance of this approach compared to the standalone search method it utilizes
- Out of a total of 100 problems, this approach solved 49 problems, while the search method solved 43 by itself
- Pruning the set of problems, we arrive at a set of 21 problems where at least one of the two methods found a solution and at least one common pattern was found through anti-unification

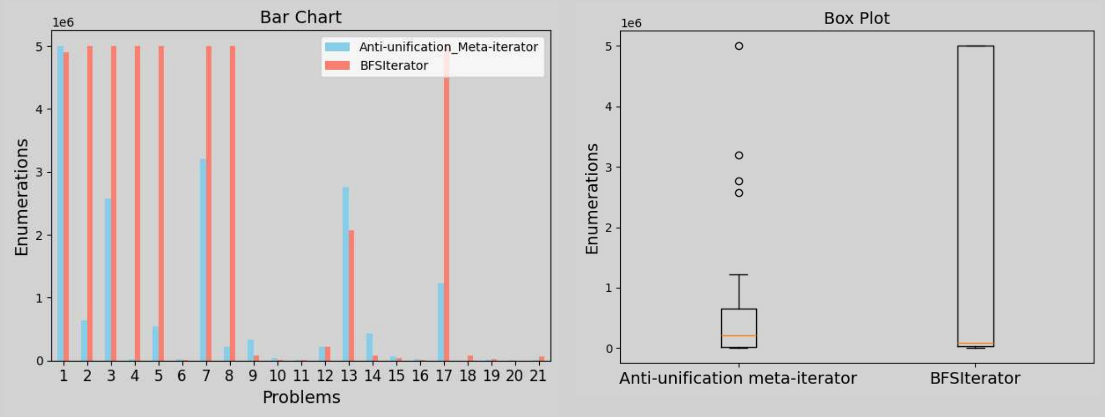


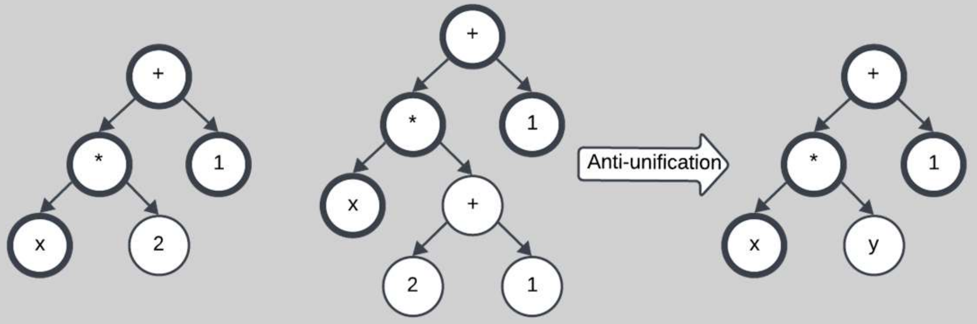
Figure 1: Plots illustrating the differences in enumerations between the Anti-unification Meta-iterator and the search method it utilizes, called BFSIterator.

Starting DSL

Number → 1 | 2
Number → x
Number → **Number** + **Number**
Number → **Number** * **Number**

Enhanched DSL

Number → 1 | 2
Number → x
Number → **Number** + **Number**
Number → **Number** * **Number**
Number → (x * **Number**) + 1



Program 1: (x * 2) + 1 **Program 2:** (x * (2 + 1)) + 1 **Common Pattern:** (x * y) + 1

Figure 2. Example iteration step in the process of enhancing the DSL of a synthesizer

4 Conclusion

- The anti-unification algorithm achieves better results by solving more problems
- However, a big limiting factor to the performance of the algorithm is additions to the DSL that do not help the synthesizer find a solution
- One such example is the first problem in Figure 2 where the common pattern found prevents the Anti-unification Meta-iterator from finding a solution
- Future work could be done to prevent such harmful additions from being added to the DSL

5 References

[1] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. Foundations and Trends in Programming Languages

[2] Peter E. Bulychev, Egor V. Kostylev, and Vladimir A. Zakharov. Anti-unification algorithms and their applications in program analysis