Sketches That Save The Budget: Guiding Program Sythesis Efficiently

Una Jacimovic | Supervised by Sebastijan Dumancic
contact: U.Jacimovic@student.tudelft.nl

Introduction

- **Program synthesis** aims to automatically generate programs that satisfy a specification, such as a set of input–output examples.
- **Budgeted search** addresses the exponential search space by decomposing synthesis into multiple bounded attempts. These attempts often produce **partially successful programs** that satisfy some examples but not the full specification.
- Many approaches have been developed independently and lack systematic comparison
- **Anti-unification** provides a principled way to generalize programs by extracting shared structural patterns,

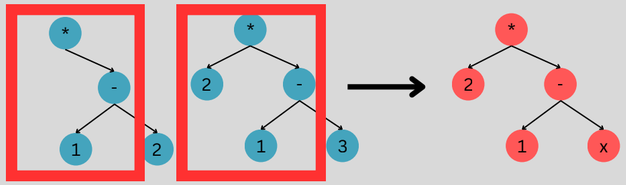

Figure 1. Anti-unified sketch between 2 programs

Research Question

How partially successful programs can reveal structural sketches that future attempts can reuse immediately, reducing the need to rediscover these patterns and improving overall efficiency.

- **Program Selection:** Which partially successful programs should be used as input for sketch search?
- **Program Grouping**: How should partially successfull programs be grouped to enable sketch extraction?
- **Rule Admission**: Under what criteria should extracted patterns be added to the grammar to improve search?

Methodology

The project is built on top of Herb.jl.

Budgeted Search Infratructure:

- Implemented a unified budgeted search framework in Herb.jl
- Enables **fixed-budget evaluation** across different iterator types
- Standardizes benchmarks, budgets, and stopping criteria

Budgeted Search Iteration

repeat bounded attempts:
partially successful programs ← Enumerator(fixed budget)
programs of interest ← Selector(programs)
updated enumerator ← Updater(programs of interest)

Figure 2. Budgeted Search Interface

Sketch Learning from Partial Programs

- Collect partial programs that satisfy ≥1 example
- Group programs to reduce heterogeneity:
  - No grouping (direct anti-unification)
  - Behavioral clustering (example agreement)
  - Root-based grouping (shared top-level rule)
- Extract sketches using anti-unification
- Reuse sketches by injecting them into the enumerator
- Give priority to sketch-generated programs

x	f(x)	program: 3 * x - 1	program: 2 * x + (x < 3)
1	3	2	2
2	5	5	5
3	7	8	6

program: 3 * x - 1	program: 2 * x + (x < 3)
0	1
1	1
1	0

→ **Similarity Score Computation**

Figure 3. Behavioral Grouping Example

Experimental Setup

- Benchmarks: HerbBenchmarks SyGuS PBE-SLIA Track 2019
- Search setting: Budgeted bottom-up synthesis
- Budget configuration:
- 100 attempts, each with 10,000 enumerations
- Sketch usage constraint:
- The number of sketch-derived programs explored per attempt is explicitly **bounded**
- Prevents a single attempt from focusing exclusively on sketches and ensures continued exploration
- Primary metric: Best solution quality achieved under a fixed budget
- Secondary metric: Enumeration step at which a given score is reached

Results

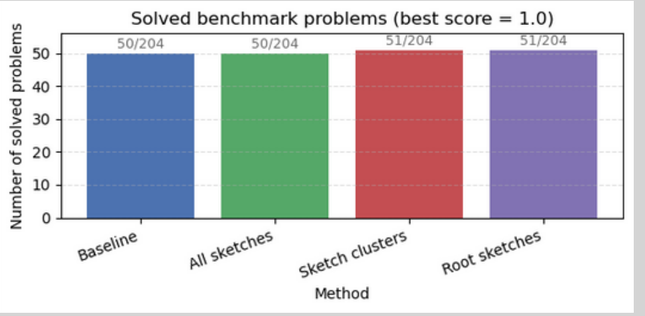

Figure 4. Benchmark Scores per grouping method

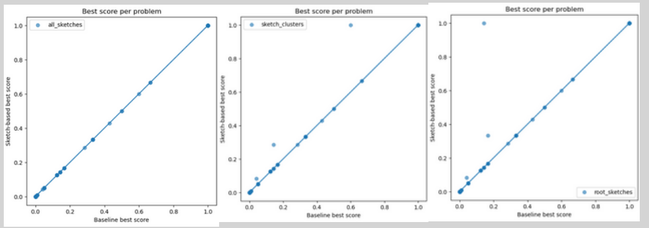

Figure 5. Best score per problem for sketch-based synthesis versus the baseline. From left to right: direct anti-unification (**all_sketches**), behavioral clustering (**sketch_clusters**), and root-based grouping (**root_sketches**).

Conclusions

- Learning from **partially successful** programs can improve budgeted program synthesis, but only when generalization is properly constrained.
- To prevent the search from over-committing to sketch-derived programs, this work **bounds** the number of **sketch** instantiations explored per attempt. As a result, sketch learning does not consistently reduce the number of enumeration steps required to reach a solution compared to the baseline.
- Sketch learning effectiveness depends on **balancing** guidance from sketches with continued exploration of alternative program structures.