Combining the Smallest Subset of Programs from Enumerative Search with Decision Trees

Author: Filip Molnár F.M.T.Molnar@student.tudelft

Supervisor: Reuben Gardos Reid
Responsible Professor: Sebastijan Dumančić

1. INTRODUCTION

Program synthesis

- The ultimate goal of program synthesis is to write an algorithm that can automatically find programs that satisfy a user intent specified by some constraints
- applications[2]: FlashFill, code suggestion (Intellisense), data wrangling
  - "john.doe@mail.com" → "john doe"
- user intent and exponentially growing program space make this task challenging

Herb.jl

- many similar concepts are implemented in other projects
- researcher have to write their software from scratch
- Herb.jl provides an adaptable environment to explore program synthesis while allowing to test and develop new ideas

2. OBJECTIVE

- merge the smallest subset of programs with decision trees inspired by [1,3]
- decision tree as layer above any iterator

Sub Questions:

- **How does a decision tree algorithm improve the basic enumeration solver when run after partial programs are generated?**
- **What is the algorithm's performance on different enumerated programs and predicates?**
- **What is the quality of produced solutions?**

3. METHODOLOGY

1. **Predicates**: programs generated by context free grammar that evaluate to boolean values
   - starting symbol is required for given grammar (starting symbol is C)
1. **Enumerate and collect programs with breadth-first iterator:**
2. **Finding the smallest subset:**
   - Greedy approximation algorithm for set cover problem
   - programs covering the most number of examples are added
3. **Combining solutions with Decision trees**:
   - feature vectors → predicates evaluated on IO examples
   - labels → every IO example has program that satisfies it
   - if (PREDICATE) then PROG_1 else PROG_2
   - multi-label DT classification problem - many examples can be solved by multiple programs

```
S ::= T | if (C) then T else T
T ::= 0 | 1 | x | y | T + T
C ::= T ≤ T | C ∧ C | ¬ C
```

4. EXPERIMENTS AND RESULTS

Improvement analysis - SyGuS dataset

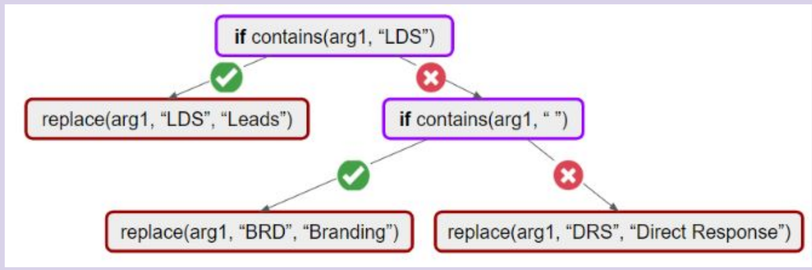
- the number of benchmarks solved and improvement of SubsetDT
- total number of problems SLIA (100), BV (317)

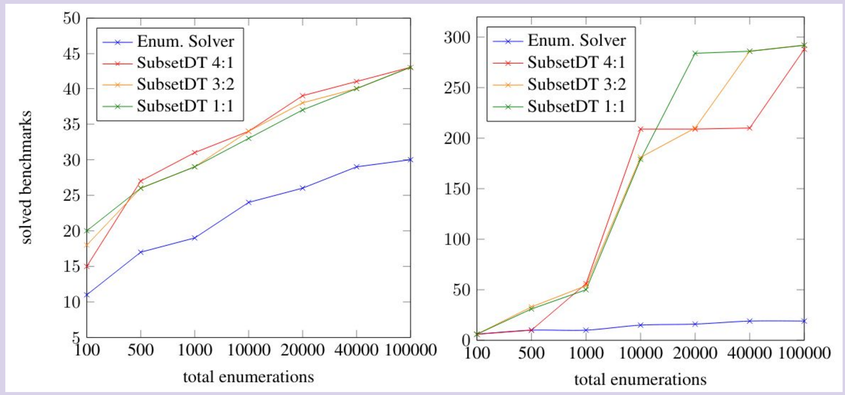
SLIA	ES	SubsetDT					NS
#Enum	0	100	500	1000	8000	24000	-
100	11	+9	+0	+0	+1	+0	0
500	17	+11	+1	+0	+2	+0	0
1000	19	+12	+1	+0	+2	+0	0
10000	24	+9	+1	+0	+3	+0	1
20000	26	+9	+2	+0	+3	+0	1
40000	29	+7	+3	+0	+3	+0	1
100000	30	+7	+3	+0	+3	+0	1

BV	ES	SubsetDT					NS
#Enum	0	100	500	1000	8000	24000	-
100	6	+0	+11	+0	+24	+22	2
500	10	+0	+40	+0	+60	+49	2
1000	10	+0	+50	+0	+71	+54	2
10000	15	+0	+69	+0	+126	+74	4
20000	16	+0	+71	+0	+123	+76	4
40000	19	+0	+73	+0	+121	+75	4
100000	19	+0	+73	+0	+121	+75	4

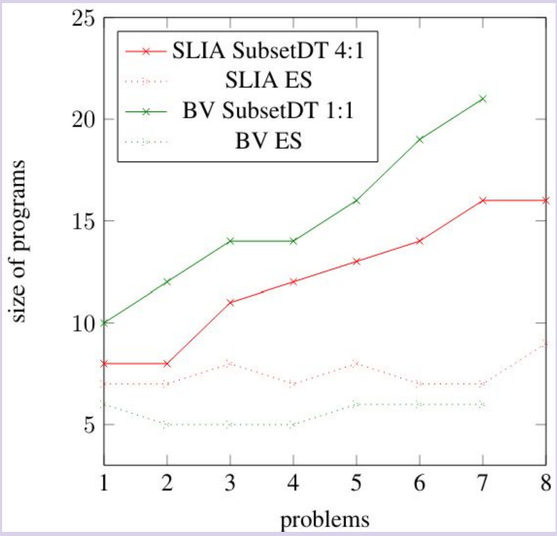
Final program consisting of 3 subprograms and 2 predicates

- overfitted on examples
- SubsetDT 1000 enums vs Enumerative Solver 500k+ enums





Performance comparison - Enumerative Solver vs SubsetDT on different **ratios** of enumerations when generating programs and predicates



Quality of solutions comparison

SLIA track - solutions are 1.5x larger compared to ES (enumerative solver)

BV track - SubsetDT programs are 3x larger than ES

Close program inspection showed overfitting

6. CONCLUSION

- SubsetDT cannot produce new solutions → highly dependent on iterator
- Standalone → can be used on top of any iterator
- SubsetDT provides a substantial improvement when used as a layer on top of the enumeration solver
- For now, it only works for problems containing a predicate subgrammar
  - rule for generating those expressions is required
- Future
  - explore other iterators for both program and predicate generation
  - create generalized predicate grammar

RELATED LITERATURE

[1] R. Alur, A. Radhakrishna, and A. Udupa, Scaling Enumerative Program Synthesis via Divide and Conquer

[2] S. Gulwani, O. Polozov, and R. Singh, Program Synthesis

[3] A. Cropper, Learning logic programs through divide, constrain, and conquer