LEARNING FROM LARGE CODEBASES

Program synthesis with noise
Part 2: the case of bounded noise and implementation

Author: Veselin Raychev
Presented by Tatiana Makhalova

January 24, 2018

Program Synthesis with Noise (ALG)

Input: Dataset $\mathcal{D}$, initial (e.g. random) dataset $\emptyset \subset d_1 \subseteq \mathcal{D}$

Output: Program p

begin

  $progs \leftarrow \emptyset$

  $i \leftarrow 0$

  repeat

    $i \leftarrow i + 1$

    // *Dataset sampling* step

    if $i > 1$ **then**

      $d_i \leftarrow ds(progs, |d_{i-1}| + 1)$

    end

    // *Program generation* step

    $p_i \leftarrow gen(d_i)$

    if $found\_program(p_i)$ **then**

      return p_i

    end

    $progs \leftarrow progs \cup \{p_i\}$

  until $d_i = \mathcal{D}$;

  return "No such program exists"

end

Basic Info

Definition (HDS)

A **hard dataset sampler** is a function ds^H such that for $Q \subseteq P$, $d' = ds^H(Q, min\_size)$, it holds that $\forall p \in Q . r(\mathcal{D}, p) \leq r(d', p)$ and $|d'| \geq min\_size$.

Theorem (THM)

Let $Q = \{p_1, ..., p_{i-1}\}$ *be the set of programs generated up to iteration i of Algorithm ALG, where the dataset sampler ds satisfies Definition HDS. If* $\varepsilon \geq r(d_i, p_{best}) - r(\mathcal{D}, p_{best})$*, then* $p_i = gen(d_i) \notin Q'$*, where*

$$Q' = \{p \in Q \mid r(\mathcal{D}, p) > r(\mathcal{D}, p_{best}) + \varepsilon\} .$$

Noise Bound

Definition (NB)

We say that ε_k is a **noise bound** for samples of size k if for the program p_{best}: $\forall d \subseteq \mathcal{D}. |d| = k \implies \varepsilon_k \geq r(d, p_{best}) - r(\mathcal{D}, p_{best})$, where r is a risk function (e.g. error rate, the number of error etc.)

Remark: We can easily instantiate THM by setting $\varepsilon = \varepsilon_k$ (see the preconndition of THM "$\varepsilon \geq r(d_i, p_{best}) - r(\mathcal{D}, p_{best})$").

Program Synthesis with Noise (ALG)

Termination criterion

Input: Dataset $\mathcal{D}$, initial (e.g. random) dataset $\emptyset \subset d_1 \subseteq \mathcal{D}$

Output: Program p

```
begin
    progs ← ∅
    i ← 0
    repeat
        i ← i + 1
        // Dataset sampling step
        if i > 1 then
            d_i ← ds(progs, |d_{i−1}| + 1)
        end
        // Program generation step
        p_i ← gen(d_i)
        if found_program(p_i) then
            return p_i
        end
        progs ← progs ∪ {p_i}
    until d_i = D;
    return ”No such program exists”
end
```

Termination criteria

We limit the error rate for a satisfactory program

$$r(\mathcal{D}, p_{satisfactory}) \leq r(\mathcal{D}, p_{best}) + \varepsilon_{satisfactory}.$$

The stopping criteria has a form:

$$found\_program(p_i) \triangleq (p_i \in progs) \wedge \varepsilon_{|d_i|} \leq \varepsilon_{satisfactory}$$

By Definition NB $r(d, p_{best}) - r(D, p_{best}) \leq \varepsilon_{|d_i|}$ (that is the precondition of THM), thus $p_i = gen(d_i) \notin Q'$, where $Q' = \{p \in progs \mid r(D, p) > r(D, p_{best}) + \varepsilon_{|d_i|}\}$. Since all the members of Q satisfies THM (due to the noise bound), if $p_i \in progs$ it satisfies the following condition:

$$r(\mathcal{D}, p_i) \leq r(\mathcal{D}, p_{best}) + \varepsilon_i.$$

Termination criterion

Special case: bound on the number of errors

Let us assume that p_{best} makes at most K errors on $\mathcal{D}$. Consider following the cost function

$$r_K(d, p) = min(num\_errors(d, p), K + 1).$$

By Definition NB, $\varepsilon_k = 0$. By setting $\varepsilon_{satisfactory} = 0$ we get the following sopping criteria:

$$found\_program(p_i) \triangleq (p_i \in progs).$$

Thus, the Algorithm ALG terminates with p_{best}.

BitSyn: bitstream programs from noisy data

Scenarios to consider:

1. the dataset $\mathcal{D}$ is obtained dynamically and the noise is bounded (i.e., up to K errors);
2. the dataset $\mathcal{D}$ is present in advance and may contain an unknown number of errors.

Goal: synthesize a program having input/output examples (32-bit integers).

Key feature: a generated program may not satisfy all provided input/output examples.

Key components of BitSyn

Objective. Tackle with overfitting problem

Regularization with function $\Omega(p_a) : \mathbb{P} \to \mathbb{R}^+$ that outputs the number of the used instructions.

Regularized objective:

$$r_{reg}(d, p) = \text{error\_rate}(d, p) + \lambda \cdot \Omega(p),$$

where $\lambda \in \mathbb{R}$ is a regularization constant.

Example

Given data:

$d_1 = \{\{2 \rightarrow 3\}, \{5 \rightarrow 6\}, \{10 \rightarrow 11\}, \{15 \rightarrow 16\}, \{-2 \rightarrow 0\}\}$

True function: $p_a = \textbf{return}$ input $+ 1$, satisfies all the examples except for $\{-2 \rightarrow 0\}$.

$$\begin{array}{ccc} + & 2 & 0000\ 0010 \\ = & 1 & 0000\ 0001 \\ \hline & 3 & 0000\ 0011 \end{array} \qquad \begin{array}{ccc} + & 15 & 0000\ 1111 \\ = & 1 & 0000\ 0001 \\ \hline & 16 & 0001\ 0000 \end{array} \qquad \begin{array}{ccc} + & \text{-2} & 1111\ 1110 \\ = & 1 & 0000\ 0001 \\ \hline & \text{-1} & 1111\ 1111 \end{array}$$

Learned function: $p_a = \textbf{return}$ input $+ 1 +$ (input $>> 7$).

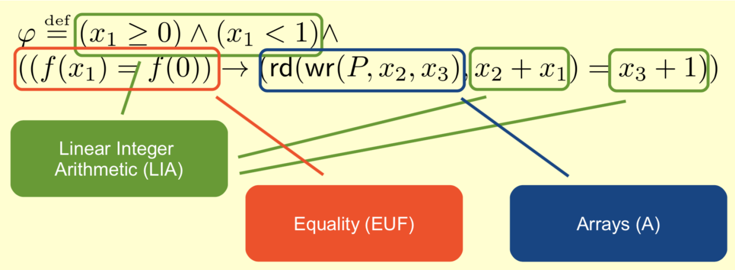
$$\begin{array}{ccc} + & 2 & 0000\ 0010 \\ + & 1 & 0000\ 0001 \\ = & 2 \gg 7 & 0000\ 0000 \\ \hline & 3 & 0000\ 0011 \end{array} \qquad \begin{array}{ccc} + & 15 & 0000\ 1111 \\ + & 1 & 0000\ 0001 \\ = & 15 \gg 7 & 0000\ 0000 \\ \hline & 16 & 0001\ 0000 \end{array} \qquad \begin{array}{ccc} + & \text{-2} & 1111\ 1110 \\ + & 1 & 0000\ 0001 \\ = & \text{-2} \gg 7 & 0000\ 0001 \\ \hline & 0 & 0000\ 0000 \end{array}$$

Key components of BitSyn

Z3 SMT Solver

Satisfiability Modulo Theories (SMT) problem is a decision problem for logical first order formulas with respect to combinations of background theories such as: arithmetic, bit-vectors, arrays, and uninterpreted functions.

Example [1]



[1] Alberto Griggio; the materials of the SAT/SMT summer school 2014

Key components of BitSyn

Z3 SMT Solver. Scheme

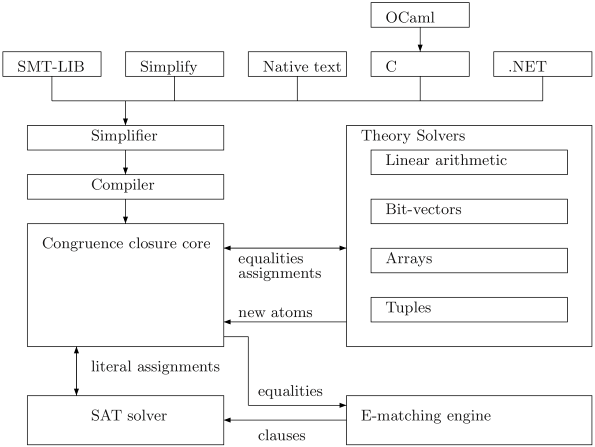


Figure: The scheme of Z3 SMT Solver, see more details on `http://research.microsoft.com/projects/z3`

Data Format

Input for SML solver:

- Encoded set of input/output examples $d = \{x_i\}_{i=1}^{n}$ [2]
- Number of allowed errors T, the solution must satisfy formula $T \geq \sum_{i=1}^{n} [\text{ if } \chi_i \text{ then } 0 \text{ else } 1]$

The program looks for the best scoring solution by iterating over the lengths of programs and T.

[2] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Ti- wari. Oracle-guided Component-based Program Synthesis. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1. ICSE 10. Cape Town, South Africa: ACM, 2010, pp. 215224. url: http://doi.acm.org/ 10.1145/1806799.1806833 (cit. on pp. 1, 101, 104, 108, 115, 116, 118, 119, 121).

Case 1: Examples in D are provided dynamically

Questions to answer

- How many errors can be processed to synthesize a correct solution?
- How many (more) examples does BitSyn need in order to compensate for the incorrect examples?

Case 1: Examples in D are provided dynamically

Results

Program	Number of instructions ↓	Number of errors (K): 0	1	2	3	4	5	6	7	8	9
		Number of input/output examples needed									
P1	2	4	4	10	7	9	11	14	16	17	22
P2	2	5	6	6	7	11	12	15	19	20	22
P3	3	4	4	9	10	8	13	15	16	17	21
P4	2	2	4	7	8	9	10	13	15	17	19
P5	2	3	3	9	9	10	10	14	16	20	22
P6	2	4	5	10	9	10	11	13	17	20	22
P7	3	5	5	7	9	11	12	15	19	20	22
P8	3	5	5	10	10	8	12	13	16	20	19
P9	3	3	timeout								

Program	Number of instructions ↓	Number of errors (K): 0	1	2	3	4	5	9
		Synthesis time (seconds)						
P1	2	1.11	1.17	1.98	1.51	1.80	7.33	102.76
P2	2	1.21	1.48	1.79	2.70	2.45	12.96	72.37
P3	3	1.75	1.81	4.42	8.63	9.20	40.62	156.09
P4	2	1.05	1.19	1.56	3.07	4.01	11.34	12.30
P5	2	1.08	1.10	1.84	3.45	9.38	11.64	139.75
P6	2	1.18	1.51	2.70	3.50	10.60	12.44	91.49
P7	3	1.80	2.20	2.77	5.15	12.65	21.62	117.16
P8	3	1.90	2.44	4.41	4.47	5.15	26.62	41.46
P9	3	2.58	timeout	timeout	timeout	timeout	timeout	timeout

Case 2: All examples in D are given in advance

Questions to answer

- Are the unsatisfied examples exactly the incorrect ones?

Remark: in the experiments the generated programs are not available before the process of anomaly detection starts.
The found unsatisfied examples is assumed to be outliers.

Case 2: All examples in D are given in advance

Results

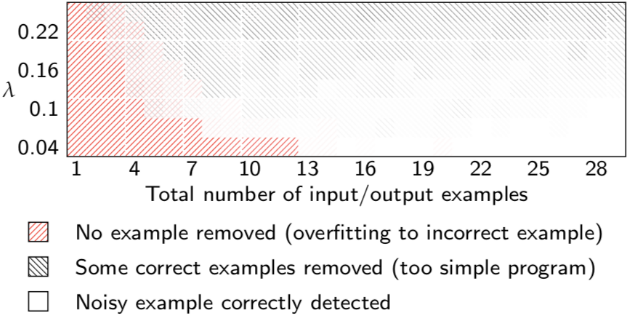


Figure: Ability of BitSyn to detect an incorrect example for programs (P1-P9) depending on total number of examples and regularization constant λ.

The results show that we need a dataset with **more than 10 examples** and a regularization constant **between 0.05 and 0.1**.

Related work

Boolean program synthesis

- synthesis from examples;
- partial programs;
  one part of a program is given imperatively and the other is given declaratively (e.g. conditions need to be achieved or maintained).
- synchronization.

All these approaches attempt to satisfy **all** provided examples and constraints.

Quantitative program synthesis

- Goal is to synthesize a program satisfying weaker specification and maximizing some quantitative objective

Summary

- A program synthesis approach that can deal with incorrect examples;
- Returns an optimal (or almost optimal) program and terminates early in case of the known bound on the cost function for the best program.
- Some suboptimal candidate programs are removed from the search space if the bound is unknown.

Thank you for your attention!