Automatic Program Synthesis of Long Programs with a Learned Garbage Collector

Amit Zohar[1], *Lior Wolf*[1,2]

[1]*Tel Aviv University* [2]*Facebook AI Research*

TEL AVIV UNIVERSITY

facebook Artificial Intelligence Research

Overview

Problem:

- Input: A handful of input/output examples for a program
- Goal: Generate a program that corresponds to all of the examples

Method:

- Step-Wise Approach: Predict the program statement by statement
- Garbage Collection: Predict which variables can be discarded at each step
- Dynamic Input: Use intermediate program states as input for the model
- Prediction Guided Search: Perform a tree search guided by the model's predictions to find a correct program

Achievements:

- Synthesis of programs more than twice as long as state-of-the-art
- Near perfect success rate for existing lengths

DSL

```
a ← [int]
b ← FILTER (%2==0) a
c ← MAP (/2) b
d ← SORT c
e ← LAST d
```

Sample 1:

Input: [13,54,138,209,36,83]

Output: 69

Sample 2:

Input: [-167,-11,-199,140,148,-124]

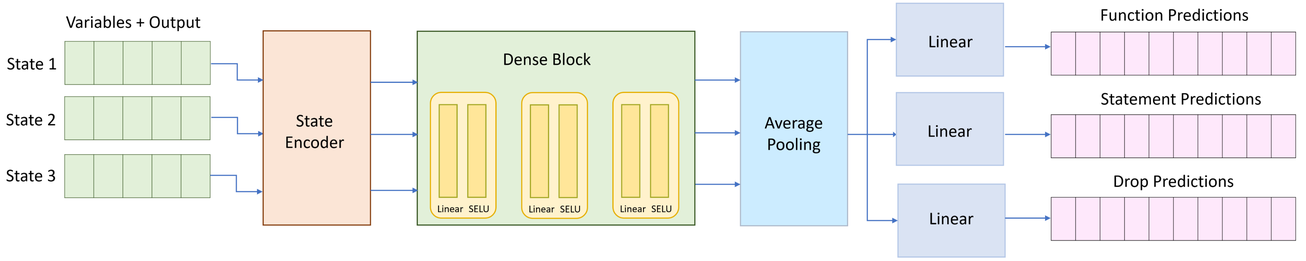
Output: 74

We use a Domain Specific Language (DSL) borrowed from *DeepCoder (Balog et al., 2017).*

- High-level programs – complex methods in a few lines of code
- A program is a sequence of function calls with their parameters
- Every statement creates a new variable – the result.
- The variables are either integers or arrays of integers.
- Contains high-level functions like ***MAP*** and low-level functions like ***HEAD, MIN***.
- The problem becomes increasingly harder with program length: for a program of length 8 – 752 possibilities for every statement.

amit.zhr@gmail.com
https://arxiv.org/abs/1809.04682
https://github.com/amitz25/PCCoder

Architecture



Our network's architecture

Program Environment

We wish to represent an intermediate state of a program during its execution:

Program State:
The sequence of all the variable values acquired thus far, starting with the program's input, and concatenated with the desired output

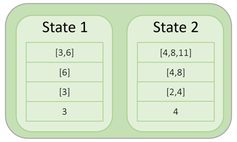
Program Environment:
The concatenation of all the program's states (one for each input/output example)

```
a ← [int]
b ← FILTER (%2==0) a
c ← MAP (/2) b
d ← SORT c
e ← LAST d
```

Sample 1: Input [3,6] Output 3

Sample 2: Input [4,8,11] Output 4

A program in the DSL with two input-output samples



The environment of the program after executing line 3

A depiction of the state encoding module of our network

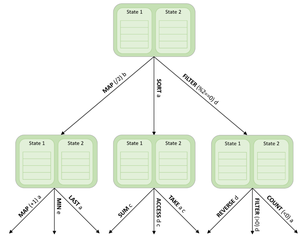
Program Environment

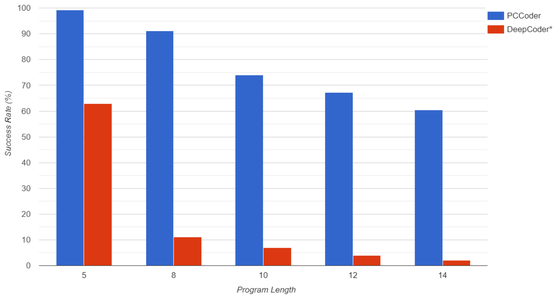
Training:
Optimize three tasks concurrently:

- Statement Prediction: The main task
- Variable Dropping: Allows to generate longer programs
- Function Prediction: Auxiliary task - provides hierarchical structure to statements

Inference:
Search for a correct program:

- Tree search – nodes are environments, edges are statements
- Use Complete Anytime Beam Search (CAB) – perform beam search in a loop, increasing the beam size and width at each iteration
- Maintain a program environment with the program predicted thus far
- Query the network at each step and order edges accordingly
- When the number of variables is exceeded – drop variables according to the network's predictions





Comparison of our method (PCCoder) with DeepCoder (*reimplementation) for various program lengths

```
v0 ← [list]
v1 ← [list]
v2 ← MAX v1
v3 ← MAP (+1) v0
v4 ← REVERSE v3
v5 ← ZIPWITH (+) v4 v4
v6 ← FILTER (>0) v5
v7 ← REVERSE v6
v8 ← MIN v7
v9 ← TAKE v8 v7
v10 ← SORT v9
v11 ← REVERSE v10
v12 ← SCANL1 (+) v11
v13 ← MAX v12
v14 ← TAKE v2 v12
v15 ← TAKE v13 v14
```

Input: [2, 1, -1, 2, 4, -1, -1, 3, 1, 4, 4, 4, -1, 2, 0, 5], [57, 133, 220, 231, 186, 82, 45, 14, 227, 227, 89, 109]
Output: [6, 10]

Input: [1, 0, 7, -1, 1, 3, 7, 2, 5, 7, -1, 1, 4, 1], [188, 237, 212, 202, 50, 19, 232]
Output: [4, 6]

Input: [17, 13, -1, 0, 8], [253, 51]
Output: [36, 64]

Input: [6, 5, 0, 6, 3, 4, 1, 7, 7, 7, 3, 8], [42, 59, 64, 29, 186, 102, 186, 141]
Output: [14, 26]

Input: [12, 3, 8, 8, 12, 6, 11, 2], [246, 113, 222, 18, 144, 250, 6, 63]
Output: [26, 52, 70, 88, 102, 110]

A real program synthesized by our method

Model	Total solved	Timeout needed to solve			
		1%	2%	4%	5%
PCCoder	83%	0.2s	0.3s	0.4s	0.6s
DeepCoder*	5%	44s	245s	311s	971s
RobustFill* (Attn A)	2%	8s	843s	-	-
RobustFill* (Attn B)	4%	7s	11s	517s	-

Comparison of our method (PCCoder) with other synthesis methods for program length 8 (*reimplementation)

Model	Total solved	Timeout needed to solve				
		20%	40%	60%	70%	80%
PCCoder	83%	3s	10s	84s	202s	635s
PCCoder (ten I/O test samples)	84%	0.6s	9s	70s	135s	530s
PCCoder (no function head)	70%	5s	66s	347s	953s	
PCCoder (no drop head)	77%	6s	11s	161s	741s	-
PCCoder (only statement head)	66%	7s	126s	660s	-	-
PCCoder (DFS)	67%	29s	310s	692s	-	-

Ablation analysis of our method (program length 8)