Predicting Code Behavior

Remy Bubulka
Coleman Gibson
Kieran Groble

Humans write imperfect software

“... the national cost estimate of an inadequate infrastructure for software testing is $59.5 billion. The potential cost reduction from feasible infrastructure improvements is $22.2 billion.” [1]

How to improve this?

- Static Analysis - [2] [3]
- Dynamic Analysis - Valgrind, JUnit
- Move less fast and break fewer things

Issues

- Determining code behavior is an undecidable problem
- Sound static analyses can give a large number of false positives, increasing noise for the developer
- Analysis can be time consuming, taking a day or more for some code bases [4]
- Doesn’t have a cool enough name
- Time is money and companies don’t like to wait for things

Potential Solution

- To reduce false positives and potentially cut down on running times, move to unsound analyses
- Allow a neural network to examine expressions within a program

Our Sandbox Language

```
<exp>         ::= <lambda-exp> | <app-exp> | <var-expr> | <lit-exp>

<lambda-exp>  ::= (λ (<formal>) <exp>)

<formal>      ::= x | y

<app-exp>     ::= (<lambda-exp> <exp>)

                | (<prim-proc> <exp> <exp>)

<prim-proc>   ::= + | - | * | /

<var-expr>    ::= x | y

<lit-exp>     ::= -2 | -1 | 0 | 1 | 2
```

Our Data

- Two methods of generation
    - Generate all possible programs up to a certain depth
    - Generate a set of random programs up to a certain depth
- Classify programs by evaluating each line to capture the true behavior
- Gives us a near infinite stream of possible programs

Our Data

```
[03:45:47] [~/Documents/ma-490/project/code-generation/data]
  ↬ wc -l output.txt
483522241 output.txt
[03:50:49] [~/Documents/ma-490/project/code-generation/data]
  ↬ ls
total 28G
drwxr-xr-x 2 kieran users 4.0K Oct 19 21:13 .
drwxr-xr-x 4 kieran users 4.0K Oct 19 21:13 ..
-rw-r--r-- 1 kieran users    0 Oct 19 21:12 .gitkeep
-rw-r--r-- 1 kieran users  28G Oct 20 03:45 output.txt
```

Our Data

```
code,output,error
-2,-2,0
-1,-1,0
0,0,0
1,1,0
2,2,0
((λ (x) x) (* (- 2 (+ -1 1)) (* 1 ((λ (x) x) -2)))),-4,0
(+ (* -2 (* (* 1 2) (+ -1 1))) ((λ (y) (+ (* -2 -1) 0)) (+ ((λ (y) y) 2) ((λ (x) 2) -1)))),2,0
((λ (y) (- (* (+ 0 -2) -1) ((λ (x) ((λ (x) 1) 0)) (- 0 -1)))) ((λ (y) ((λ (y) (- -1 -1)) ((λ (y) y) 1))) ((λ (y) ((λ (y) y) -2)) (/ -2 2)))),1,0
((λ (y) ((λ (y) (- ((λ (y) 2) -2) (/ 2 -1))) ((λ (y) (+ -2 1)) ((λ (y) y) 0)))) ((λ (x) ((λ (y) ((λ (x) x) -2)) ((λ (x) x) 1))) ((λ (y) y) (/ -2 0)))),0,1
((λ (y) (- (/ ((λ (y) 0) 1) ((λ (x) -1) 0)) (- ((λ (y) 2) -1) 0))) (/ 1 1)),-2,0
(+ 1 1),2,0
(- 2 (- (- ((λ (x) -1) -2) 1) ((λ (x) ((λ (y) -1) -2)) (- 2 2)))),3,0
(/ (+ 1 (* 0 -1)) (+ -1 (/ 0 (/ -2 0)))),0,1
((λ (y) ((λ (y) y) -1)) (+ (+ (* -1 0) ((λ (x) x) 1)) (* (- -1 2) -2))),-1,0
((λ (x) (/ 0 (* (/ 0 0) (+ -1 0)))) (+ (+ ((λ (y) y) 0) (+ -2 -2)) ((λ (x) ((λ (y) 1) 0)) ((λ (x) 2) -2)))),0,1
(* ((λ (y) (- (- 2 -2) (+ 2 -2))) -1) (+ 2 ((λ (y) 0) (* 0 -1)))),8,0
(- ((λ (y) (+ -2 (/ 1 2))) (+ (- 1 1) 0)) ((λ (x) (/ (- -1 1) (* 1 -2))) ((λ (y) y) ((λ (x) -2) -2)))),-2.5,0
(- 0 (- (* ((λ (y) 1) -1) ((λ (x) x) 2)) (- (/ -1 -2) -1))),-0.5,0
((λ (x) -1) ((λ (x) (/ ((λ (x) x) 0) ((λ (y) y) 0))) -1)),0,1
((λ (y) ((λ (x) (* 2 ((λ (x) 1) 0))) -1)) ((λ (y) ((λ (y) y) ((λ (y) 2) -1))) ((λ (y) ((λ (x) -2) 2)) (+ 0 2)))),2,0
(- -2 1),-3,0
((λ (y) 2) -1),2,0
(+ (- (- ((λ (y) y) -2) ((λ (y) y) 2)) ((λ (y) y) ((λ (x) x) 0))) ((λ (y) ((λ (x) (+ 1 2)) 1)) 2)),-1,0
((λ (x) ((λ (x) 1) (/ ((λ (x) x) -2) ((λ (x) x) -2)))) ((λ (x) 0) (- -1 (/ 0 2)))),1,0
((λ (y) (+ 1 -1)) ((λ (x) x) ((λ (x) x) 2))),0,0
(* ((λ (x) 2) (+ ((λ (y) y) 1) ((λ (x) x) -1))) ((λ (y) ((λ (y) y) (/ -2 0))) ((λ (y) (+ -1 0)) (* 2 0)))),0,1
(- 2 0),2,0
(- (* (+ -2 (- 1 -2)) (/ (- -1 -2) 2)) (* ((λ (x) ((λ (y) -2) 0)) ((λ (y) y) 2)) ((λ (y) ((λ (y) y) 0)) ((λ (x) x) 2)))),0.5,0
((λ (y) ((λ (x) ((λ (y) -2) (/ -2 -2))) (- 2 0))) (/ (* (* -2 0) ((λ (y) y) 0)) (/ ((λ (x) x) -2) -1))),-2,0
((λ (y) 2) 1),2,0
(/ ((λ (x) x) ((λ (x) (* 0 -2)) (* -2 1))) (/ -1 ((λ (y) (* 2 -2)) ((λ (x) -2) -2)))),0.0,0
(+ ((λ (y) y) ((λ (y) 0) ((λ (x) 2) 1))) (* -2 2)),-4,0
((λ (y) (/ ((λ (y) ((λ (x) -2) -1)) (* 2 -2)) (/ ((λ (y) y) -1) ((λ (y) y) -2)))) (/ (- (* -2 1) ((λ (x) -2) 0)) -2)),-4.0,0
((λ (y) ((λ (x) (/ (+ -1 2) ((λ (x) 1) -1))) ((λ (x) x) 1))) (+ -1 (* -2 ((λ (y) 2) 1)))),1,0
((λ (y) 0) -2),0,0
(* (+ ((λ (y) 1) ((λ (y) y) -1)) (/ -2 ((λ (y) 0) -1))) -2),0,1
(* ((λ (y) 0) 2) (/ (- (* 1 -1) ((λ (x) x) -1)) ((λ (y) y) 0))),0,1
((λ (y) y) 0),0,0
((λ (y) y) (- -2 ((λ (y) y) 0))),-2,0
((λ (x) ((λ (y) y) 1)) ((λ (y) 2) ((λ (x) ((λ (y) 0) 2)) (+ 0 0)))),1,0
(- -2 (/ ((λ (x) ((λ (x) -1) -2)) ((λ (x) x) 2)) (* (/ -2 1) -1))),-1.5,0
(- (- -1 ((λ (x) ((λ (x) 1) 1)) 2)) (- ((λ (y) y) 1) ((λ (x) ((λ (x) x) 2)) (+ -2 2)))),-1,0
(- (+ ((λ (x) ((λ (x) -1) 0)) ((λ (x) 1) 0)) (/ ((λ (x) x) 1) (* 2 -1))) ((λ (y) ((λ (y) (/ 1 1)) (/ 2 0))) (/ (/ -1 -2) (+ 2 -2)))),0,1
(* 0 (- (* (+ 0 -2) (* -1 -2)) ((λ (x) x) ((λ (x) -1) 1)))),0,0
((λ (y) (- (- 0 (/ 2 0)) ((λ (y) y) ((λ (y) y) 0)))) ((λ (y) y) ((λ (x) x) (/ -2 1)))),0,1
((λ (x) (/ 1 ((λ (y) y) -1))) ((λ (x) ((λ (y) ((λ (x) x) -2)) ((λ (x) x) -2))) (+ ((λ (x) x) 1) -2))),-1,0
((λ (x) (- 2 (* ((λ (x) x) 2) (* -1 0)))) 0),2,0
(/ ((λ (x) ((λ (y) x) (* -2 2))) ((λ (y) 2) 1)) 0),0,1
(* ((λ (x) (- 1 1)) ((λ (x) -1) -1)) (+ ((λ (x) (+ 2 1)) ((λ (x) x) 2)) 2)),0,0
(+ 2 1),3,0
(- ((λ (y) (+ (/ 2 -1) ((λ (y) y) 1))) 2) (+ ((λ (x) x) ((λ (y) y) 0)) (* 2 ((λ (x) x) -2)))),3,0
(+ ((λ (y) 1) ((λ (y) 1) (* 2 1))) (* (- (- 1 -1) ((λ (x) x) 0)) ((λ (y) y) (* -2 1)))),-3,0
(/ ((λ (y) y) (+ 0 ((λ (y) y) 1))) ((λ (y) -2) ((λ (y) ((λ (y) -2) 1)) -1))),-0.5,0
(+ -1 (+ 0 1)),0,0
```

Data Preprocessing

- Removed non-unique programs
- Balanced the data to not have a large bias towards working programs
- Tokenized each symbol (plus a “nothing” token)
- Program encoded as an $m \times n$ matrix
  - m: length of the longest program
  - n: number of tokens

Analyses

- Classification
  - Error or no error
- Regression
  - Predict raw output

LSTM - Long Short Term Memory

- An LSTM remembers a value for either long or short time periods
- A simplified recurrent neural network
- Doesn’t modify the value as it feeds it back to itself
- Considers the outputs of remembered values when evaluating new ones

Architecture & Results: Error Prediction

- 256 node LSTM layer
- 20% dropout layer
- 2 node output layer
- Softmax activation, accuracy metric

- Trained on random programs up to depth 3
- 99.19% accuracy on the test set

Results - Error Prediction

Code	Confidence of Error	Should Error
`(/ 1 1)`	0.0001%	No
`(/ 2 0)`	99.99%	Yes
`(/ 0 1)`	0.0001%	No
`(/ 1 (- 1 1))`	34.47%	Yes
`(+ 1 0)`	0.000099%	No
`(/ 1 ((λ (x) 1) 0))`	0.0001%	No
`(/ 1 ((λ (x) 0) 1))`	98.82%	Yes
`(+ 0 (* 0 (/ 0 1)))`	0.0001%	No
`((λ (x) (/ (- 0 0) (- 0 1))) 0)`	83.34%	No

Architecture & Results: Output Prediction

- 256 node LSTM layer
- 20% dropout layer
- 1 node output layer
- MSE, MAE metrics

- Trained on programs up to depth 2
- .0096 MSE, .027 MAE

Results - Output Prediction (Good Inputs)

Code	Predicted Output	Actual Output
(+ 2 1)	3.02568173	3
(+ 1 2)	3.05547428	3
((λ) (x) (+ x 1)) 1)	1.39392674	2
(/ 1 2)	0.44873938	0.5
(* 2 (* 2 2))	9.61662197	8
(* 2 2)	4.25841475	4
(- 1 2)	-0.90237552	-1
(* 1 (+ 1 1))	1.83598292	2
'(* 2 (+ 1 1))1	3.6835146	4
1	0.72898144	1
2	-0.44422317	2

Results - Output Prediction (Bad Inputs)

Code	Predicted Output
`(2 + 1)`	2.87946272
`2 + 1`	2.93325639
`1 + 2`	2.00386
`(2 * (1 + 1))`	2.85971975
`(+ x y)`	-0.4209832
`x`	-0.88294739
`y`	-0.89534634

Citations

[1] - The Economic Impacts of Inadequate Infrastructure for Software Testing

[2] - EXE: Automatically Generating Inputs of Death

[3] - Test Input Generation with Java PathFinder

[4] - A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World