Learning to analyze programs at scale

Martin Vechev
ETH Zurich and DeepCode.ai

http://plml.ethz.ch
Big Code: last 5 years @ ETH
more: http://plml.ethz.ch

Joint work with:
- Pavol Bielik
- Veselin Raychev
- Andreas Krause
- Christine Zeller
- Svetoslav Karaivanov
- Pascal Roos
- Benjamin Bichsel
- Timon Gehr
- Petar Tsankov
- Mateo Panzacchi

Publications
- Robust Relational Layout Synthesis from Examples, ACM OOPSLA’18
- Inferring Crypto API Rules from Code Changes, ACM PLDI’18
- Program Synthesis for Character Level Language Modeling, ICLR’17
- Learning a Static Analyzer from Data, CAV’17
- Statistical Deobfuscation of Android Applications, ACM CCS’16
- Probabilistic Mode for Code with Decision Trees, ACM OOPSLA’16
- PHOG: Probabilistic Mode for Code, ACM ICML’16
- Learning Programs from Noisy Data, ACM POPL’16
- Predicting Program Properties from “Big Code”, ACM POPL’15, CACM’18
- Code Completion with Statistical Language Models, ACM PLDI’14
- Machine Translation for Programming Languages, ACM Onward’14

Statistical Systems for Code
- apk-deguard.com
- jsnice.org
- nice2predict.org
What are we working on now in Big Code?
Problem I: Automating Design
[00PSLA'18]

Custom Deep Learning models

Probabilistic constraints learned over existing layouts

Synthesis Algorithms [interpretability]

Provable robustness metrics [to work across many devices]

http://www.inferui.com/
Problem II: Probabilistic Models of Code

Program

```
elem.notify({
    position: 'top',
    autoHide: false,
    ?
});
```

AST

```
P(elem | S (element | S (element | S (element | S (Identifier elem, Property notify, Property position, Property autoHide | MemberExpression | ObjectExpression | CallExpression, String 'top', Boolean false)))))
```

Probabilistic Model

- $S$ is a learned semantic analyzer
- Can predict entire expressions at once, precisely
- Generalizes PHOG

We tried various attention-based networks to capture context but generally deep learning performs worse.

See B. Mularzyk’s ETH Thesis

We plan to release the PHOG code [ICML’16, OOPSLA’16]
Problem III: Translation of languages

Sounds easy, but very hard problem: 0 margin for error, different than NLP (where similar intent fine)

Learns translation rules in a DSL (fully interpretable)

Easy to get data, learning hard to get to scale, with 0 mistakes!

Deep Learning does not work well (various ICLR papers). See Pavle Djordevich’s ETH M.Sc. thesis
Problem IV: Statistical Deobfuscation
[JavaScript (http://jsnice.org), Android (http://apk-deguard.com), Binaries (Debin)]

Lots of JS code uploaded to be analyzed
Major redesign 2 months ago

New baseline training in Nice2Predict.org

Major challenge: fast inference with precision. Various works sacrifice one, making tool less usable in practice
Analyzer again learns right few features to use. Fast to extract, yet precise.
Problem is harder than just name prediction (many names removed, not only one, so harder to condition on)
Applicable to graphical models generally
Problem V: Learning from commits

Learn from thousands of commits

Predict bugs elsewhere
Make explainable predictions

Semantic analysis over large corpus of code and learning using **interpretable** model

Key for displaying **actionable** suggestions to users

Enable learning of new static analysis checks [PLDI’18]
Problem VI: Learn to Analyze Programs

Learn analyzer from code, changes, etc.

Synthesis + machine learning

Neural-based analyzers not too effective
Few lessons

Learning over syntax does not work well, need deeper representations...but not hardcoded...so, learn (interpretable) analyzer to generate the representation/features for the particular task

Interpretability important for checking hard specifications

Most Big Code systems require some combination of synthesis, analysis, learning

Opportunity to push the concepts beyond Big Code [e.g., PHOG to NLP]

Need systems and datasets that are easy to use and extend
Repeating pattern: semantic learning

Dataset of syntactic programs $\mathcal{D}$

Language parser

Dataset of syntactic ASTs $\mathcal{D}_{ast}$

Synthesize Static Analyzer (interpretable)

DSL

Dataset of semantic facts/features $\mathcal{D}_{sem}$

Apply $S$

$F_1$

$x > 0 \land y < 0$

$F_2$

$a = \text{open}(); \text{read}(a)$

$F_n$

Learn Probabilistic Model $\text{Pr}$

Prediction $\text{Target}$ (names, statements, etc)

$\text{AST}_1 \rightarrow \text{open}$

$\text{AST}_2 \rightarrow \text{splitsString}$

$\text{AST}_n \rightarrow \text{rename}$

$\Pr(\text{Target} | F)$

Goodness of $\text{Pr}$
Lets see an example of learning in more detail
Goal: learn a static analyzer

Few interesting points:

We will see how interpretability helps Approximation space on labels, due to lattices

New data generated to cover corner cases

Decision Tree Learning + CEGIS
Writing Static Analyzer is Hard

**DOOP**
Framework for Java Pointer Analysis
17 contributors

**TAJS**
Static Type Checker for JavaScript

**Static Type Checker for JavaScript**
512 contributors

Writing static analyzer is
- hard
- frustrating
- time consuming
- brittle

Learn more
Example of unsoundness

false negative

from: https://flow.org/try/
Input Dataset
\( \mathcal{D} = \{(x^j, y^j)\}_{j=1} \)

Language \( \mathcal{L} \) for abstract transformers

Synthesis + Over-approximation

\( p a_{\text{best}} \in \mathcal{L} \)
How to obtain a suitable dataset?

Input Dataset
\[ \mathcal{D} = \{(x^j, y^j)\}_{j=1} \]

How to learn over large search spaces? How to prevent overfitting?

Language \( \mathcal{L} \) for abstract transformers

What is the language of transformers?

Synthesis + Over-approximation

\[ p\alpha_{\text{best}} \in \mathcal{L} \]
Example learned transformer

Array.prototype.filter ::=  
  if caller has one argument then  
    points-to global object  
  else if 2nd argument is Identifier then  
    if 2nd argument is undefined then  
      points-to global object  
    else  
      points-to 2nd argument  
  else if 2nd argument is this then  
    points-to 2nd argument  
  else if 2nd argument is null then  
    points-to global object  
  else //2nd argument is a primitive value point to boxed version of argument  
    points-to new allocation site
Let us show the learning on an example analysis (aka points-to analysis)
Dataset: Points-to Analysis

**Program**

```javascript
function collect(value, idx, obj) {
    if (value >= this.threshold) {
        ...
    }
    ...
}
```

**Abstract Syntax Tree (AST)**

- `l1` - IfStatement
- `l2` - BinaryExpression
- `l3` - Identifier:value
- `l4` - MemberExpression
- `l5` - ThisExpression
- `l6` - Property:threshold

**Objects read/written during execution**

- `01` - Value
- `02` - Value
- `03` - Value
Dataset: Points-to Analysis

Program

```javascript
function collect(value, idx, obj) {
    if (value >= this.threshold) {
        ...
    }
    ...
}
```

Abstract Syntax Tree (AST)

Objects read/written during execution

```
\[
\mathcal{D} = \{(x^j, y^j)\}_{j=1}
\]
```

\[
\langle (\text{AST}, l_5), o_2 \rangle
\]
Language Describing Transformers

\[ l \in \mathcal{L} := a \mid \text{if } g = c \text{ then } l \text{ else } l \]

\[ a \in \text{Actions} \quad \quad \quad g \in \text{Guards} \]

```javascript
function collect(val, idx, obj) {
  if (val >= this.threshold) { ... }
}
```

```javascript
var dat = [5, 3, 9];
dat.filter( collect, ctx );
```

g_1 \quad \text{method name that called collect is filter}

g_2 \quad \text{function has a 2nd argument}

\[ g_1 \quad g_2 \quad a_1 \quad a_2 \quad a_3 \]

can be represented as decision tree
Learning: Decision Trees + CEGIS

Input Dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^j$

Language $\mathcal{L}$ for abstract transformers

Synthesis + Over-approximation

candidate analysis $pa \in \mathcal{L}$

Oracle:
Test/Verify Analyzer

counter-example $(x, y) \notin \mathcal{D}$
$\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$

no counter-example
return analysis $pa$
Problem Formulation

\[ p_{a_{\text{best}}} = \arg \min_{p_a \in \mathcal{L}} \text{cost}(\mathcal{D}, p_a) \]
\[ \text{st. } \forall \langle x, y \rangle \in \mathcal{D} \cdot \alpha(y) \sqsubseteq p_a(x) \]

guarantees analysis soundness

Cost Function

\[ r(x, y, p_a) = \begin{cases} 1 & \text{if } (\alpha(y) \neq p_a(x)) \\ 0 & \text{else} \end{cases} \]
\[ \text{cost}(\mathcal{D}, p_a) = \sum_{\langle x, y \rangle \in \mathcal{D}} r(x, y, p_a) \]

prefer analysis with fewer errors

can capture lattice-based distance function to capture approximation
Learning Algorithm

\[ l \in \mathcal{L} := a \mid \text{if } g = c \text{ then } l \text{ else } l \]

Synthesise Programs in Parts
Learning Algorithm

\[ l \in \mathcal{L} := \alpha \mid \text{if } g = c \text{ then } l \text{ else } l \]

Synthesise Programs in Parts

Diagram:

- \( g_2 \) with true arrow to \( a_1 \)
Learning Algorithm

\[ l \in \mathcal{L} := a \mid \text{if } g = c \text{ then } l \text{ else } l \]

Synthesise Programs in Parts

```
g2
  true \rightarrow a_1
  false \rightarrow a_2
```
Learning Algorithm

\[ a_{\text{best}} = \arg \min_{a \in \text{Actions}} \text{cost}(\mathcal{D}, a) \]

- If \( \text{cost}(\mathcal{D}, a_{\text{best}}) > 0 \), refine analysis.
- If \( \text{cost}(\mathcal{D}, a_{\text{best}}) = 0 \), return \( a_{\text{best}} \).

\[ \mathcal{D} \]

\[ a_{\text{best}} \]
Learning Algorithm

\[ g_{\text{best}} = \text{arg max} \ \text{InfGain}(\mathcal{D}, g, a_{\text{best}}) \]

\[ g \in \text{Guards} \]

\[ \text{cost}(\mathcal{D}, a_{\text{best}}) > 0 \]

refine analysis

Find split that separates \( a_{\text{best}} \)

\[ g_{1} \rightarrow \]

\[ g^{*} \rightarrow \]

\[ g_{2} \rightarrow \]
Learning Algorithm

\[
g_{\text{best}} = \arg \max \ InfGain(\mathcal{D}, g, a_{\text{best}})
\]

\[
g \in \text{Guards}
\]

\[
\text{cost}(\mathcal{D}, a_{\text{best}}) > 0
\]

Refine analysis

Find split that separates

\[
a_{\text{best}}
\]
Learning Algorithm

\[ g_{\text{best}} = \arg \max \ InfGain(\mathcal{D}, g, a_{\text{best}}) \]

\[ g \in \text{Guards} \]

\[ \text{cost}(\mathcal{D}, a_{\text{best}}) > 0 \]

Refine analysis

Find split that separates

\[ a_{\text{best}} \]

\[ g^* \rightarrow \]

\[ g_1 \rightarrow \]

\[ g_2 \rightarrow \]

\[ \mathcal{D}_t \]

\[ \mathcal{D}_f \]

\[ \text{InfGain}(\mathcal{D}, g, a_{\text{best}}) = 0 \]

No split reduces entropy

\[ a_{\text{best}} = \text{approximate}(\mathcal{D}) \]
Learning: Decision Trees + CEGIS

Input Dataset
\( \mathcal{D} = \{ \langle x^i, y^i \rangle \}_{j=1} \)

Language \( \mathcal{L} \) for abstract transformers

Synthesis + Over-approximation

Oracle:
Test/Verify Analyzer

candidate analysis \( pa \in \mathcal{L} \)

counter-example \( \langle x, y \rangle \notin \mathcal{D} \)
\( \mathcal{D} \leftarrow \mathcal{D} \cup \{ \langle x, y \rangle \} \)

no counter-example return analysis \( pa \)

How to find complex counter-examples quickly?
How to efficiently explore hard to find corner cases?
Naïve Approach: Random Fuzzing

1. Pick a random training example $\langle x, y \rangle \in \mathcal{D}$
2. Mutate the input randomly $x \rightarrow x'$
3. Obtain the correct label Execute $x' \rightarrow \mathcal{D}'$
4. Check for correctness $\forall \langle x, y \rangle \in \mathcal{D}' . \alpha(y) \subseteq pa(x)$
5. Repeat from beginning
Naïve Approach: Random Fuzzing

1. Pick a random training example
2. Mutate the input randomly
3. Obtain the correct label
4. Check for correctness
5. Repeat from beginning

Exponential Number of Choices
Slow
When to stop?
The Oracle: Testing an Analyzer

Key Idea: Take advantage of candidate analysis $pa$
Interpretability helps!

How to sample from space of all programs?
The Oracle: Testing an Analyzer

- **Execution path coverage of $pa$**
- **Mutate only parts that affect $pa$**
- **Select relevant program mutations**

\[
\mathcal{T}
\]

```javascript
fnc collect(val, idx, obj) {
    if (val >= this.threshold)
    {
        ...  // Query
    }
}

var dat = [5, 3, 9];
dat.filter(collect, ctx);
```

- Modification via Equivalence Modulo Abstraction (EMA)
- Modification via Global Jumps
The Oracle: Testing an Analyzer

 Modifications via Equivalence Modulo Abstraction (EMA)

Semantic preserving mutations
- Adding dead code
- Renaming variables
- Renaming user defined functions
- Side-effect free expressions

labels $y$ can be reused
The Oracle: Testing an Analyzer

Modifications via Equivalence Modulo Abstraction (EMA)

Semantic preserving mutations

Non-semantic preserving mutation

Modifications via Global Jumps
Evaluation

ECMAScript (ECMA-262) Conformance Suite

15 675 Programs

Points-to Analysis

```javascript
function collect(val, idx, obj) {
    if (val >= this.threshold) { ... }
}
var dat = [5, 3, 9];
dat.filter(collect, ctx);
```

Allocation Site Analysis

```javascript
var obj = {a: 7};
var arr = [1, 2, 3, 4];
if (arr.slice(0, 2) == ... )
    var n = new Number(7);
var obj2 = new Object(obj);
try { ... } catch (err) { ... }
```
Method Instantiation for Points-to Analysis

Input Dataset
\[ D = \{(x^j, y^j)\}_{j=1} \]

\( y \leftarrow \text{concrete object id} \)

Language \( \mathcal{L} \) for abstract transformers

\( a \in \text{Actions} \quad ::= \quad \epsilon \mid \text{Move; } a \)

\( \text{Move}_{\text{core}} \quad ::= \quad \text{Up, Left, Right, DownFirst, DownLast, Top} \)

\( g \in \text{Guards} \quad ::= \quad \epsilon \mid \text{Move; } g \mid \text{Write; } g \)

\( \text{WriteOp} \quad ::= \quad \text{WriteValue, WriteType, WritePos, HasLeftSibling, HasRightSibling, HasChild} \)

Synthesis

\( (\mathcal{H}, \Xi), \alpha, \gamma \)

Oracle

\begin{align*}
\text{semantic preserving mutations} & \quad \text{non-semantic preserving mutations} \\
\text{Adding dead code} & \quad \text{Add method arguments} \\
\text{Renaming variables} & \quad \text{Add method parameters} \\
\text{Renaming user defined functions} & \quad \text{Change program constants} \\
\text{Side-effect free expressions} &
\end{align*}
Learned points-to analysis

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Dataset Size</th>
<th>Analysis Size</th>
<th>Counter-examples Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function.prototype</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>call()</td>
<td>26</td>
<td>97(18)</td>
<td>372</td>
</tr>
<tr>
<td>apply()</td>
<td>6</td>
<td>54(10)</td>
<td>182</td>
</tr>
<tr>
<td>Array.prototype</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>map()</td>
<td>315</td>
<td>36(6)</td>
<td>64</td>
</tr>
<tr>
<td>some()</td>
<td>229</td>
<td>36(6)</td>
<td>82</td>
</tr>
<tr>
<td>forEach()</td>
<td>604</td>
<td>35(5)</td>
<td>177</td>
</tr>
<tr>
<td>every()</td>
<td>338</td>
<td>36(6)</td>
<td>31</td>
</tr>
<tr>
<td>filter()</td>
<td>408</td>
<td>38(6)</td>
<td>76</td>
</tr>
<tr>
<td>find()</td>
<td>53</td>
<td>36(6)</td>
<td>73</td>
</tr>
<tr>
<td>findIndex()</td>
<td>51</td>
<td>28(7)</td>
<td>96</td>
</tr>
<tr>
<td>Array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>from()</td>
<td>32</td>
<td>57(7)</td>
<td>160</td>
</tr>
<tr>
<td>JSON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stringify()</td>
<td>18</td>
<td>9(2)</td>
<td>55</td>
</tr>
</tbody>
</table>

Average Learning Time 14 minutes (4 min synthesis, 10 min oracle)
Learned Allocation-Site Analysis

134 721 training dataset size

905 counter-examples found

99 refinement iterations

3 hours Synthesis time

7 hours time to find counter-examples

Overview of Learned Analysis

```java
if HasPrevNodeValue then T
elif WriteType == CallExpression then
  if Up WriteType == ExpressionStatement then
    T // return value not assigned
  else ...
elif WriteType == ArrayAccess then ...
elif WriteType == ObjectExp|ArrayExp|RegExp then
  NewAlloc // implicit constructors
elif WriteType == NewExpression then
  ... // explicit constructor
elif Up WriteType == AssignmentExpression
  if left hand side of the assignment then NoAlloc
  ...
```
Lessons

learn (interpretable) analyzer to generate the representation/features for the particular task

Interpretability important for checking hard specifications

Most Big Code systems require some combination of synthesis, analysis, learning

Opportunity to push the concepts beyond Big Code (e.g., PHOG to NLP)

Need systems and datasets that are easy to use and extend

Interesting Domain for Learning

Interpretable

Hard constraints

New data generated

Approximation of output label space

Algorithm: Decision Tree Learning + CEGIS

Input Dataset
$$D = \{(x', y')\}_{i=1}^n$$

Synthesis + Over-approximation

Language $$\mathcal{L}$$ for abstract transformers

Found Real Issues

rules missed by Facebook Flow