Learning a Static Analyzer from Data

by Pavol Bielik, Veselin Raychev, and Martin Vechev

Daniel Perez

The University of Tokyo

January 29, 2018

Static analyzers

Writing a static analyzer is hard

Many corner cases → many handcrafted rules

FlowJS type checking core is ~$12,000$ lines of ML

JavaScript points-to sample

```
global.length = 4;
var dat = [5, 3, 9, 1];
function isBig(value) {
  return value >= this.length;
}
dat.filter(isBig);
dat.filter(isBig, 42);
dat.filter(isBig, dat);
```

Sample learned analyzer

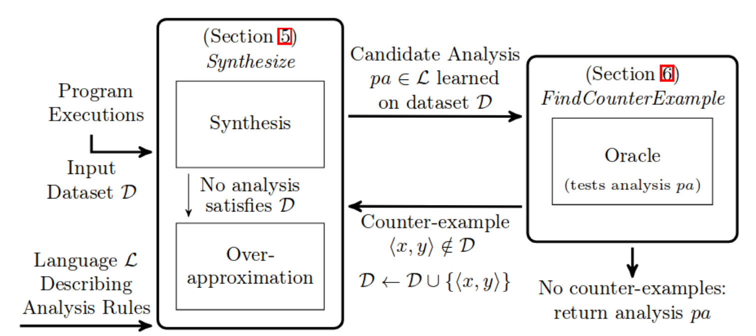
We would like to automatically learn such rules while avoiding overfitting the training data

```
// points to global
dat.filter(isBig);
// points to boxed 42
dat.filter(isBig, 42);
// points to dat object
dat.filter(isBig, dat);
```

```
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 // 2nd arg is a primitive value
    points–to new allocation site
```

Overview

System takes dataset and rules as input and outputs analysis



Model input

Dataset

System takes a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$ where

- x is an input program
- y is the analysis result

Example sample

Sample input x

```
var b = {}; // object s0
a = b;
```

Analysis result

$y = \{(a \to \{s_0\})\}$

Rules description language

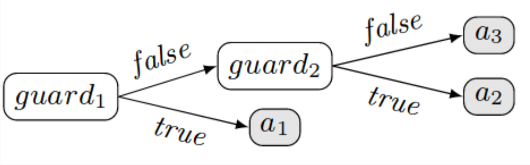
Language template is

$\langle Action \rangle$ ::= `action on AST`

$\langle Guard \rangle$ ::= `condition`

$\langle Prog \rangle$ ::= $\langle Action \rangle$
| `'if'` $\langle Guard \rangle$ `'then'` $\langle Prog \rangle$ `'else'` $\langle Prog \rangle$

which enables to model



Analyzer properties

We want the analyzer *pa* to be sound and precise

Sound

Analzyer *pa* is sound if

$$\forall p \in \mathcal{T}_{\mathcal{L}}, \alpha([\![p]\!]) \sqsubseteq pa(p)$$

but too hard to proof, instead

$$\forall i \in 1 \ldots N, y^i \sqsubseteq pa(x^i)$$

i.e. sound on the dataset

Precise

Given

$$r(x, y, pa) = \begin{cases} 1 & y \neq pa(x) \\ 0 & \text{otherwise} \end{cases}$$

the goal is to minimize

$$cost(\mathcal{D}, pa) = \sum_{(x,y) \in \mathcal{D}} r(x, y, pa)$$

Learning algorithm

Learning procedure based on ID3

procedure SYNTHETIZE($\mathcal{D}$)

Input: Dataset $\mathcal{D} = \left\{ \left(x^i, y^i\right) \right\}_{i=1}^{N}$

Output: Program $pa \in \mathcal{L}$

$a_{best} \leftarrow \arg\min_{a \in Actions} cost(\mathcal{D}, a)$

if $cost(\mathcal{D}, a) = 0$ **then**

return a_{best}

$g_{best} \leftarrow \arg\max_{g \in Guards} {}^{\top} IG^{a_{best}}(\mathcal{D}, g)$

if $g_{best} = \perp$ **then**

return Approximate($\mathcal{D}$)

$p_1 \leftarrow$ Synthetize($\{(x, y) \in \mathcal{D} | g_{best}(x)\}$)

$p_2 \leftarrow$ Synthetize($\{(x, y) \in \mathcal{D} | \neg g_{best}(x)\}$)

return (**if** g_{best} **then** p_1 **else** p_2)

Information gain

IG is information gain: difference of entropy

$$w_d^{a_{best}} = \langle r(x_i, y_i, a_{best}) \mid i \in 1 \ldots |d| \rangle$$

$$IG^{a_{best}}(\mathcal{D}, g) = H\left(\mathbf{w}_{\mathcal{D}}^{a_{best}}\right) - \frac{|\mathcal{D}^g|}{|\mathcal{D}|} H\left(\mathbf{w}_{\mathcal{D}^g}^{a_{best}}\right) - \frac{|\mathcal{D}^{\neg g}|}{|D|} H\left(\mathbf{w}_{\mathcal{D}^{\neg g}}^{a_{best}}\right)$$

Algorithm properties

- Greedy, locally optimal
- Sound on $\mathcal{D}$ iif. Approximate is sound

Oracle — counter-example generator

Goal

Find counter-example (x, y) st. $pa(x) \neq y$ in reasonable time

- Random search too slow
- Prioritize modifications affecting execution path of $pa(x)$

Modification types

- Semantic preserving (Equivalence Modulo Abstraction, EMA)
- Non-semantic preserving (Global jump)

Example

Sample input

```
var b = {};
a = b;
```

Overfitted analysis

```
if y is VarDecl:y preceding x
  then y
if there is VarDecl:x(y)
  then y
else  ⊥
```

Counter-example

```
var b = {};
var c = 1;
a = b;
```

Evaluation

Overview

- Learned 2 analyzers
  - Points-to analysis subset (`this` points-to)
  - Site-call allocation analysis
- Input programs from ECMAScript conformance suite (~15000 samples)

Program modifications

F_{ema}	F_{gj}
Adding dead code	Adding method arguments
Renaming variables	Adding method parameters
Renaming user functions	Changing constants
Side-Effect Free expressions	

Points-to analysis

Goal

Learn `this` points-to rules, a function f st.

$$\frac{\texttt{VarPointsTo}(v_2, h) \qquad v_2 = f(\texttt{this})}{\texttt{VarPointsTo}(\texttt{this}, h)}$$

Example

```
// points to global
dat.filter(isBig);
// points to boxed 42
dat.filter(isBig, 42);
// points to dat object
dat.filter(isBig, dat);
```

```
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 // 2nd arg is a primitive value
    points—to new allocation site
```

Points-to analysis rules description language

Generate actions with programs up to size 5 and branches programs up to size 6 (5 moves and 1 write)

$\langle \mathit{MoveCore} \rangle ::= \text{Up} \mid \text{Left} \mid \text{Right} \mid \text{DownFirst} \mid \text{DownLast} \mid \text{Top}$

$\langle \mathit{MoveJS} \rangle ::= \text{GoToGlobal} \mid \text{GoToUndef} \mid \text{GoToNull} \mid \text{GoToThis} \mid \text{UpUntilFunc}$

$\langle \mathit{Move} \rangle ::= \langle \mathit{MoveCore} \rangle \mid \langle \mathit{MoveJS} \rangle \mid \text{GoToCaller}$

$\langle \mathit{Write} \rangle ::= \text{WriteValue} \mid \text{WritePos} \mid \text{WriteType} \mid \text{HasLeft} \mid \text{HasRight} \mid \text{HasChild}$

$\langle \mathit{Action} \rangle ::= \epsilon \mid \langle \mathit{Move} \rangle\ \langle \mathit{Action} \rangle$

$\langle \mathit{Guard} \rangle ::= \epsilon \mid \langle \mathit{Move} \rangle\ \langle \mathit{Guard} \rangle \mid \langle \mathit{Write} \rangle\ \langle \mathit{Guard} \rangle$

$\langle \mathit{Context} \rangle ::= \epsilon \mid (N \cup \Sigma \cup \mathbb{N})\ \langle \mathit{Context} \rangle$

$\langle \mathit{Prog} \rangle ::= \epsilon \mid \langle \mathit{Action} \rangle \mid \text{`}\texttt{if}\text{'}\ \langle \mathit{Guard} \rangle\ \text{`}\texttt{=}\text{'}\ \langle \mathit{Context} \rangle\ \text{`}\texttt{then}\text{'}\ \langle \mathit{Prog} \rangle\ \text{`}\texttt{else}\text{'}\ \langle \mathit{Prog} \rangle$

Points-to analysis results

Function Name	Dataset Size	Counter-examples Found	Analysis Size*
`Function.prototype`			
`call`	26	372	97(18)
`apply`	6	182	54(10)
`Array.prototype`			
`map`	315	64	36(6)
`some`	229	82	36(6)
`forEach`	604	177	35(5)
`find`	53	73	36(6)

* Number of instructions in $\mathcal{L}_{pt}$ (Number of **`if`** branches)

Allocation analysis

Goal

Learn a allocation site analysis function f st.

$$\frac{f(l) = \texttt{true}}{\texttt{AllocSite}(l)}$$

Results

- 34721 input/output samples
- 135 branches generated
- 905 counter examples found
- learned tricky cases — e.g. `new Object(obj)`

Summary

- New approach to learn static analyzer from data
    - Algorithm to learn analyzer from dataset and inference rules
    - Oracle to quickly generate counter-examples, avoiding overfitting
- Learned tricky rules for JavaScript points-to and site-allocation analysis

References

- P. Bielik, V. Raychev, and M. T. Vechev, "Learning a static analyzer from data," *CoRR*, vol. abs/1611.01752, 2016. [Online]. Available: http://arxiv.org/abs/1611.01752
- J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986. [Online]. Available: http://dx.doi.org/10.1023/A:1022643204877