Learning a Static Analyzer from Data

Pavol Bielik Veselin Raychev Martin Vechev

Department of Computer Science
ETH Zurich

European Research Council
Established by the European Commission
Supporting top researchers from anywhere in the world

LLVM Compiler and Code Generation Socials
Zürich, February 8

Learning from “Big Code”

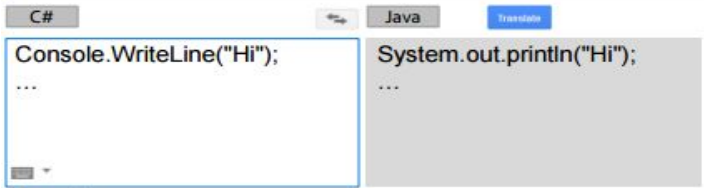
Write new code:
Code Completion

```
Camera camera = Camera.open();
camera.SetDisplayOrientation(90);
                ?
```

Understand code/security:
JavaScript Deobfuscation
Type Prediction

JS NICE

Port code:
Programming Language Translation



Debug code:
Statistical Bug Detection

```
...                    likely error
for x in range(a):
    print a[x]
```

All of these benefit from the “Big Code” and lead to applications not possible with previous techniques

Writing a Static Analyzer

DOOP

Framework for Java
Pointer Analysis

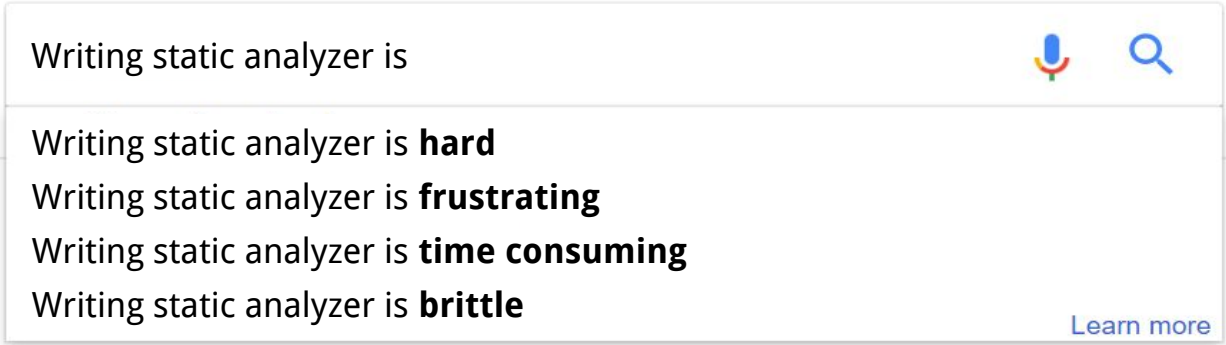
17 contributors

TAJS

Static Type Checker
for JavaScript

Static Type Checker
for JavaScript

~400 contributors



Example of Unsound Analysis

flow

```
1  /* @flow */
2
3  var data = [7, 2, 9];
4
5  function collect(val, idx, obj) {
6    if (val > this.threshold) {
7
8    }
9  }
10
11 data.threshold;
12 data.forEach(collect);
```

Missed Error

Error correctly reported

Errors | JSON | AST | v0.51.0

```
11: data.threshold;
          ^ property `threshold`. Property not found in
11: data.threshold;
    ^ Array
```

Learning Points-to Analysis: Intuition

$$\frac{\texttt{to = from} \qquad \texttt{VarPointsTo(from, h)}}{\texttt{VarPointsTo(to, h)}}$$

Learning Points-to Analysis: Intuition

$$\frac{\texttt{to = from} \qquad \texttt{VarPointsTo(from, h)}}{\texttt{VarPointsTo(to, h)}}$$

Variable

Allocation Site

$$\frac{\texttt{to = base.fld} \qquad \texttt{VarPointsTo(base, baseH)} \qquad \texttt{FldPointsTo(baseH, fld, h)}}{\texttt{VarPointsTo(to, h)}}$$

Learning Points-to Analysis: Practice

High Complexity **Domain Knowledge** **Lots of Corner Cases**

```
VCall(b, “Class.getMethod”, i, _)
ActualReturn(i, r)          VarPointsTo(b, hc)
ReifiedClass(t, hc)         ActualArg(i, 1, p)
VarPointsTo(p, c)           ConstantForMethod(c, s)
LookUp(t, s, _)             ReifiedMethod(s, hm)
-------------------------------------------------
VarPointsTo(r, hm)
```

Can we learn a static analyzer?

(aka its abstract transformers)

sound and interpretable

Can we learn a static analyzer?

(aka its abstract transformers)

This Work: Learn Static Analyzer from Data

Input Dataset

$\mathcal{D} = \{\langle x^j, y^j \rangle\}_{j=1}$

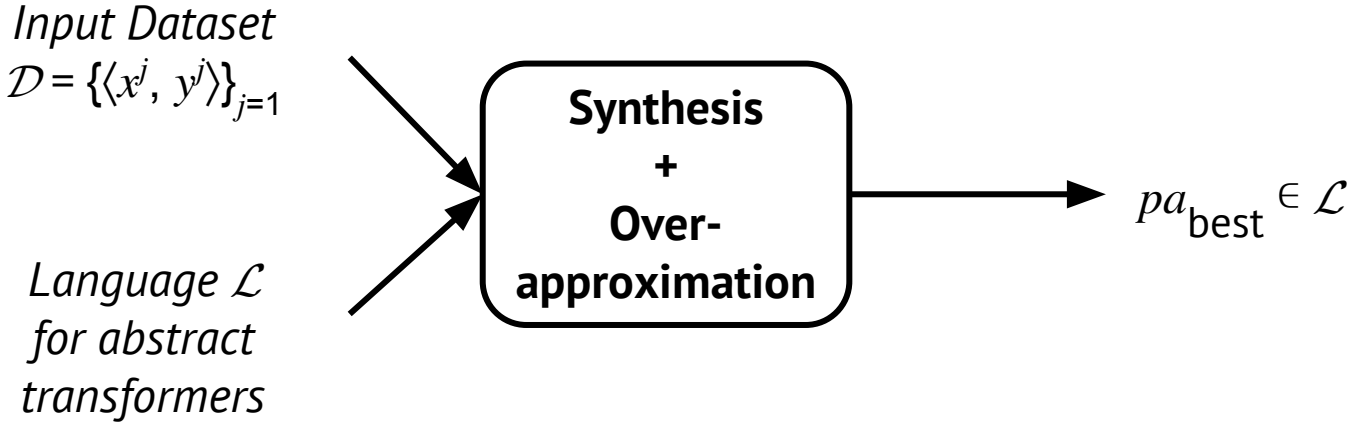
This Work: Learn Static Analyzer from Data

Input Dataset

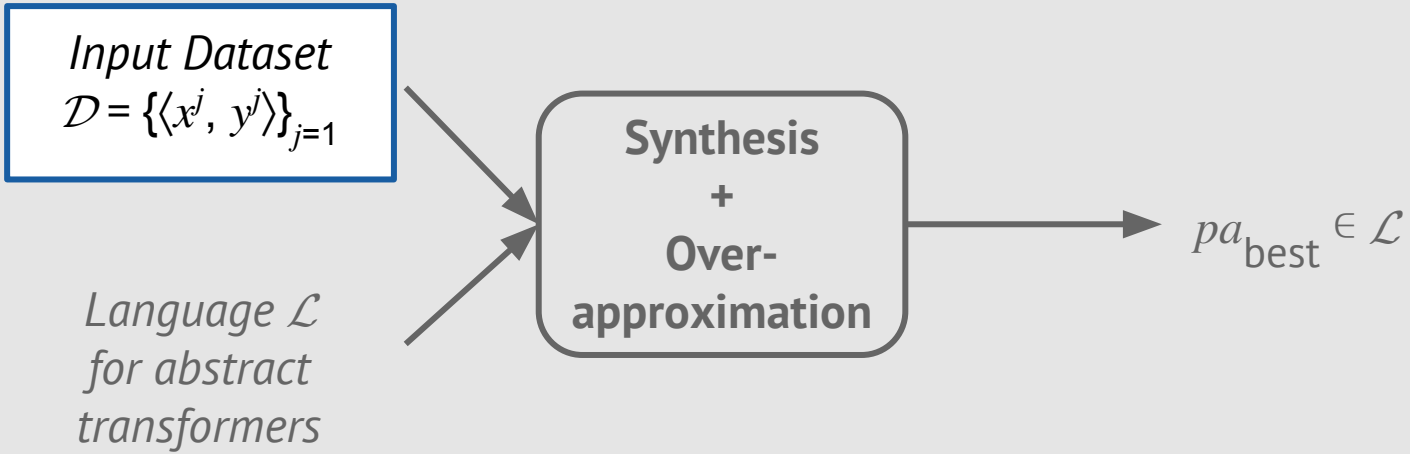
$\mathcal{D} = \{\langle x^j, y^j \rangle\}_{j=1}$

Language $\mathcal{L}$ for abstract transformers

This Work: Learn Static Analyzer from Data

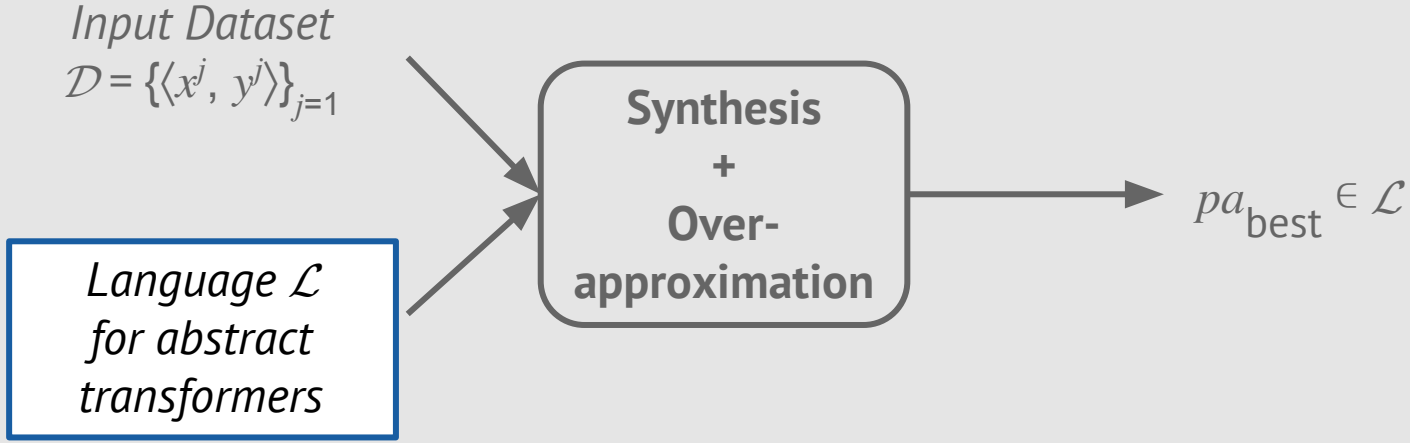


This Work: Learn Static Analyzer from Data



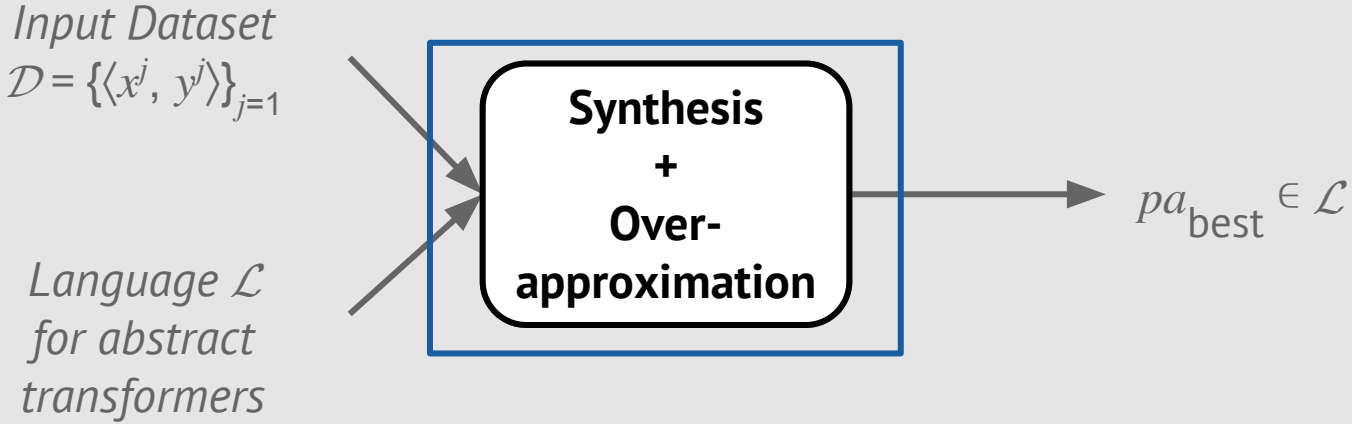
How to obtain suitable dataset?

This Work: Learn Static Analyzer from Data



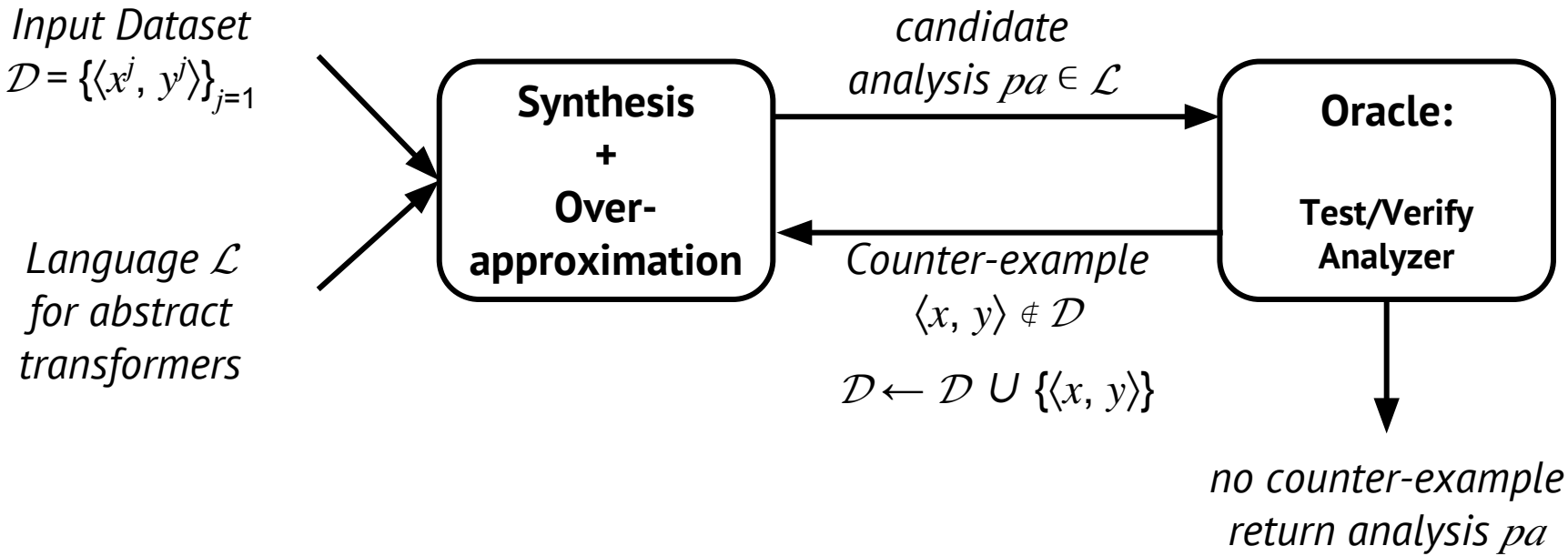
What is the language over which to learn?
How to allow generating new interesting transformers?

This Work: Learn Static Analyzer from Data

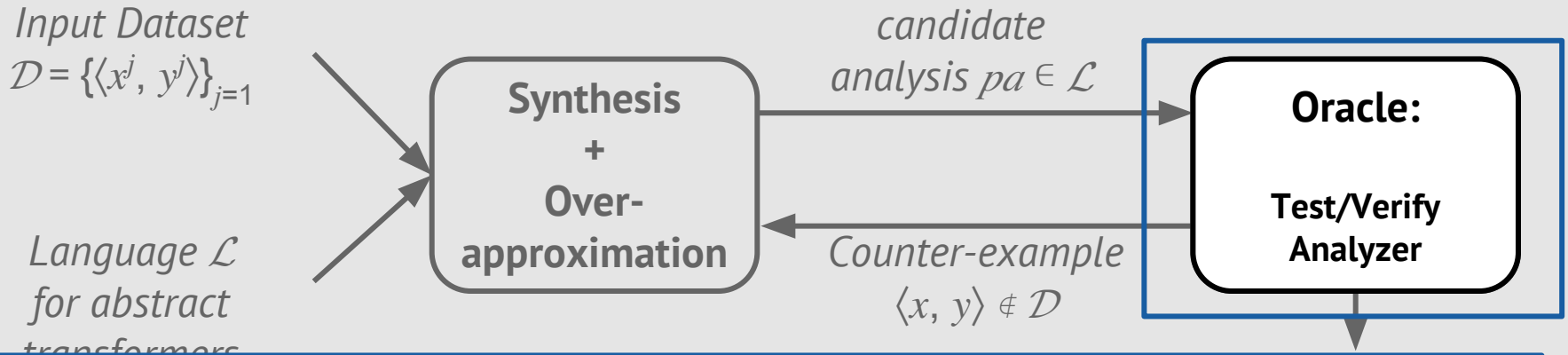


How to design scalable learning over large search spaces?
How to prevent overfitting?

Learning: Synthesis + CEGIS



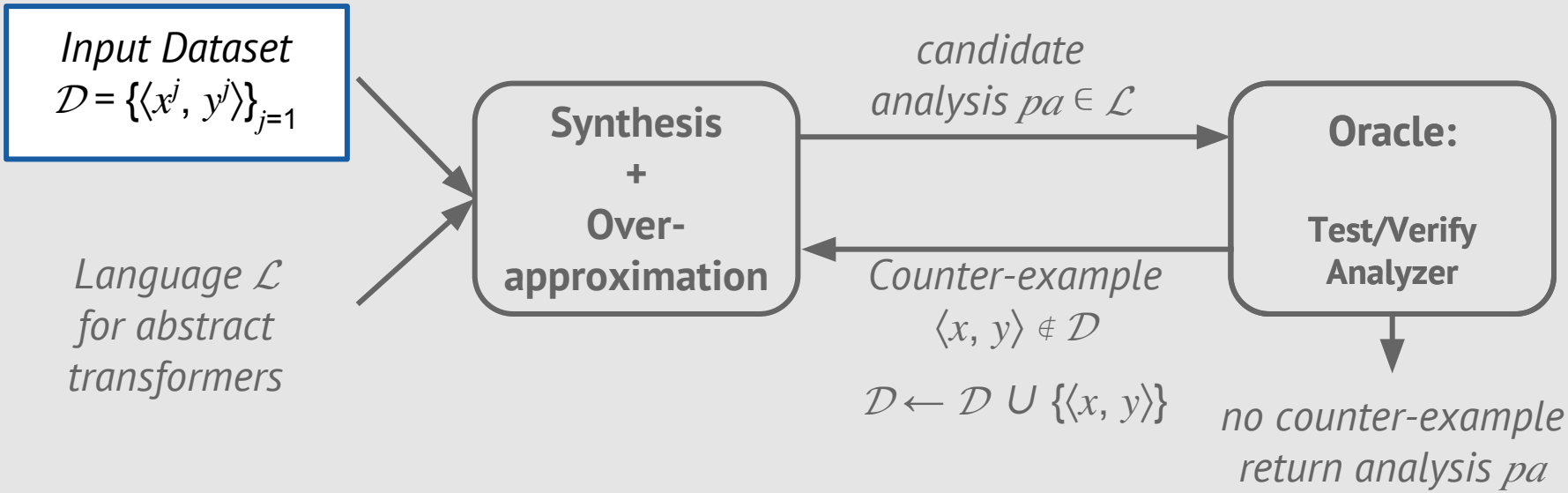
Learning: Decision Trees + CEGIS



How to find complex counter-examples quickly?
How to efficiently explore hard to find corner cases?

Let us show the learning on an example analysis (aka points-to analysis)

Dataset



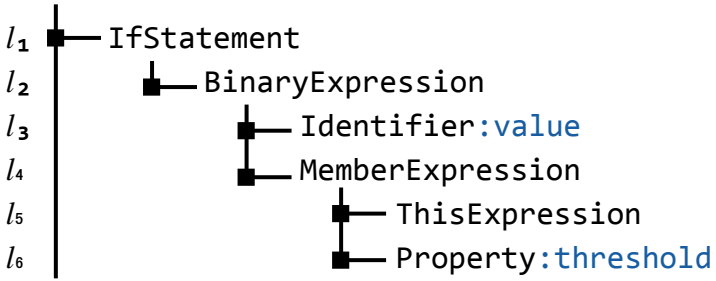
How to obtain suitable dataset?

Dataset: Points-to Analysis

Program

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

Abstract Syntax Tree (AST)



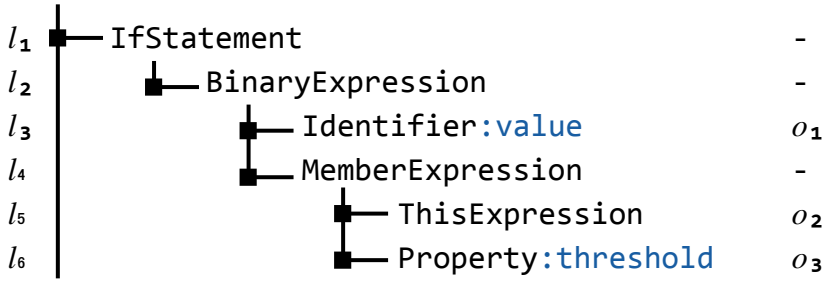
Dataset: Points-to Analysis

execute programs to obtain concrete behaviours

Program

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

Abstract Syntax Tree (AST)



execution reads/writes

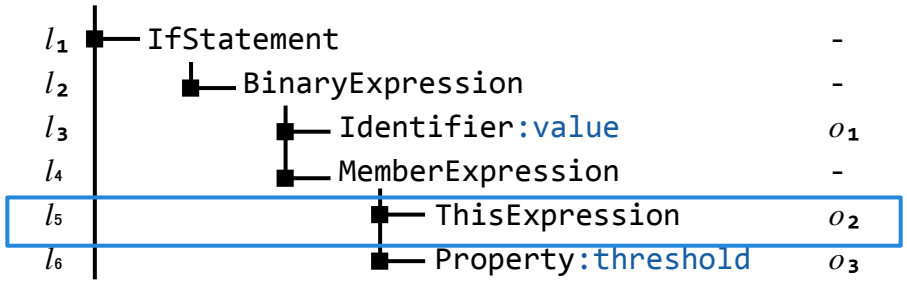
l_1	-
l_2	-
l_3	o_1
l_4	-
l_5	o_2
l_6	o_3

Dataset: Points-to Analysis

Program

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

Abstract Syntax Tree (AST)



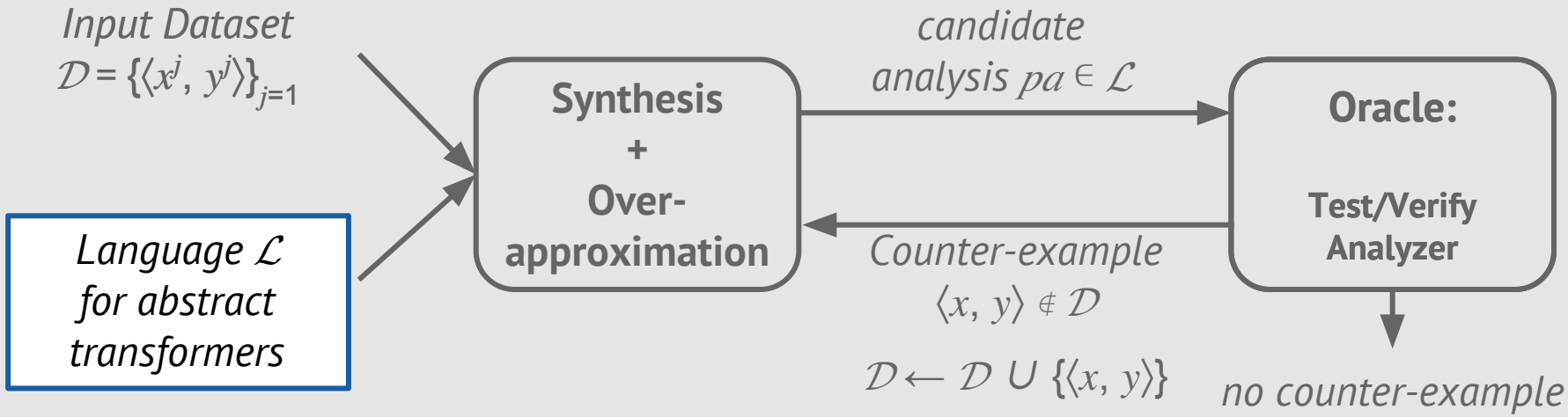
execution reads/writes

l_1 -
l_2 -
l_3 o_1
l_4 -
l_5 o_2
l_6 o_3

$$\langle (AST, l_5), o_2 \rangle$$

$$\mathcal{D} = \{\langle x^j, y^j \rangle\}_{j=1}$$

Language Describing Abstract Transformers



What is the language over which to learn?
How to allow generating new interesting transformers?

Language Describing Abstract Transformers

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

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

Points-to Query

method name
is Array.filter

has
2nd argument

Language Describing Abstract Transformers

$$l \in \mathcal{L} ::= a \mid \textbf{if } g \textbf{ then } l \textbf{ else } l$$

$a \in$ *Actions* $g \in$ *Guards*

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

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

Points-to Query a_1

method name is Array.filter g_1

has 2nd argument g_2

Language Describing Abstract Transformers

$$l \in \mathcal{L} ::= a \mid \textbf{if } g \textbf{ then } l \textbf{ else } l$$

$a \in$ *Actions* $\quad$ $g \in$ *Guards*

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

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

Points-to Query

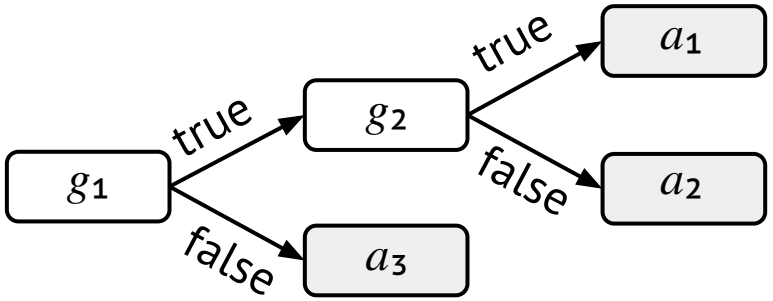
a_1

method name is Array.filter

g_1

has 2nd argument

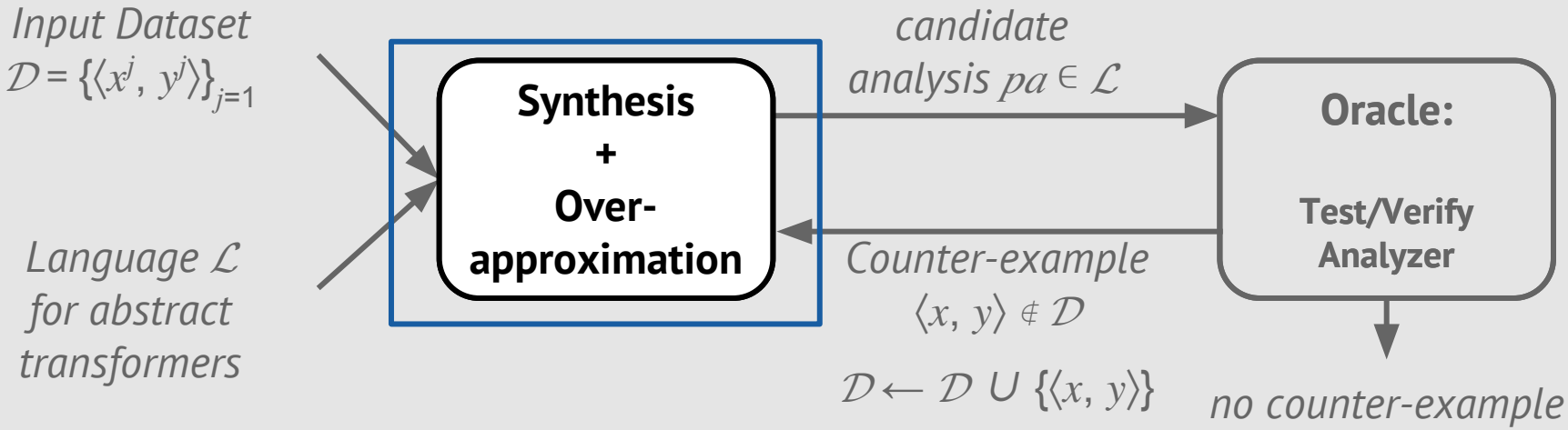
g_2



can be represented as decision tree

paths interpreted as abstract transformers

Learning: Decision Trees



How to design scalable learning over large search spaces?
How to prevent overfitting?

Learning: Problem Formulation

Problem Formulation

$$pa_{\text{best}} = \underset{pa \in \mathcal{L}}{\arg\min}\ cost(\mathcal{D}, pa)$$

$$\text{st. } \forall \langle x, y \rangle \in \mathcal{D} \,.\, \alpha(y) \sqsubseteq pa(x)$$

guarantees analysis soundness

Cost Function

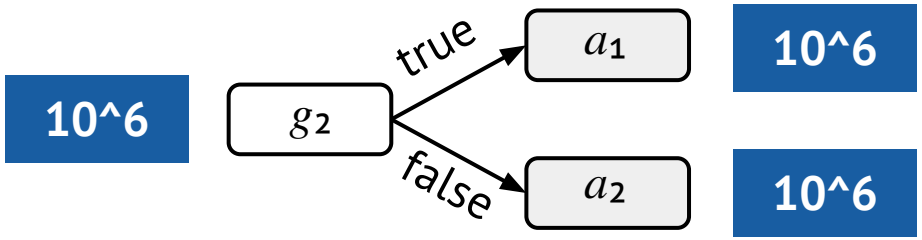
$$r(x, y, pa) = \textbf{if } (y \neq pa(x)) \textbf{ then } 1 \textbf{ else } 0$$

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

prefer analysis with fewer errors

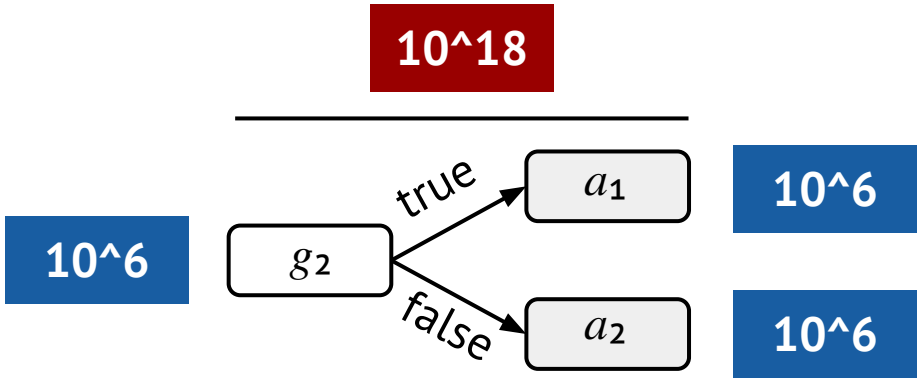
Learning Algorithm

$l \in \mathcal{L} := a \mid$ **if** g **then** l **else** l



Learning Algorithm

$l \in \mathcal{L} := a \mid$ **if** g **then** l **else** l

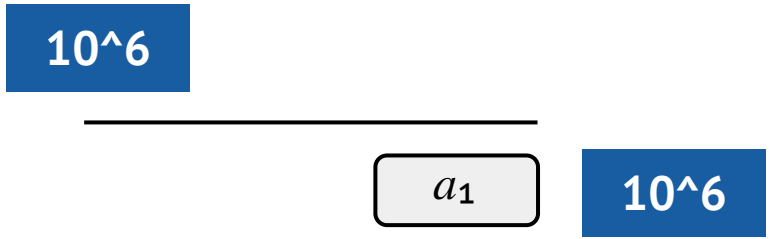


Untractable

Learning Algorithm

$l \in \mathcal{L} \coloneqq a \mid$ **if** g **then** l **else** l

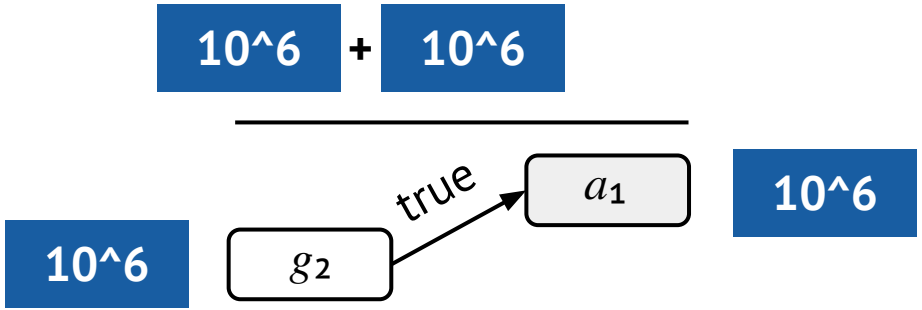
Key Idea: Synthesise Programs in Parts



Learning Algorithm

$l \in \mathcal{L} := a \mid$ **if** g **then** l **else** l

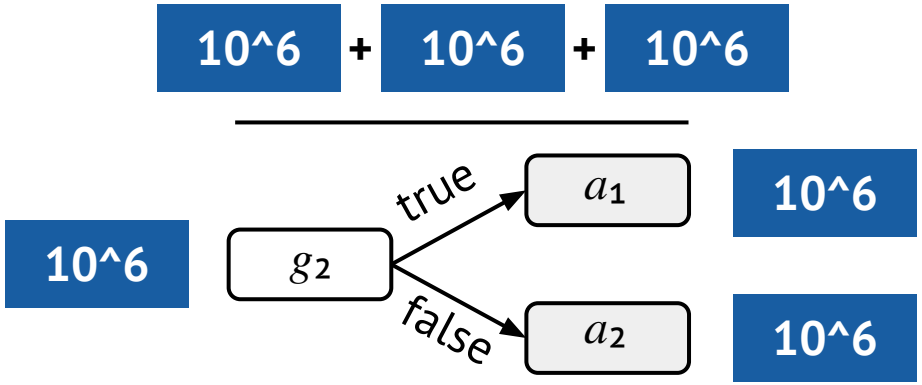
Key Idea: Synthesise Programs in Parts



Learning Algorithm

$l \in \mathcal{L} \coloneqq a \mid$ **if** g **then** l **else** l

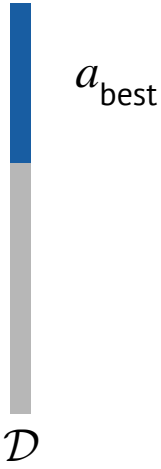
Key Idea: Synthesise Programs in Parts



Learning Algorithm

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

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



refine analysis

$cost(\mathcal{D}, a_{\text{best}}) = 0$

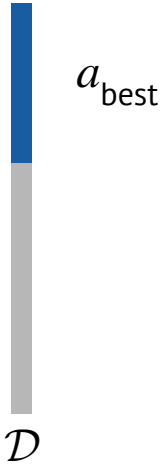
no errors
return a_{best}

Learning Algorithm

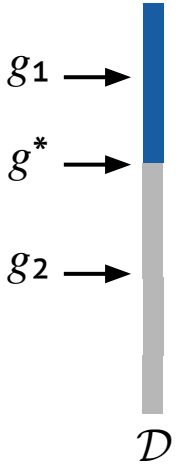
$$g_{\text{best}} = \underset{g \in \mathit{Guards}}{\arg\max}\ \mathit{InfGain}(\mathcal{D}, g, a_{\text{best}})$$

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

refine analysis



Find split that separates a_{best}

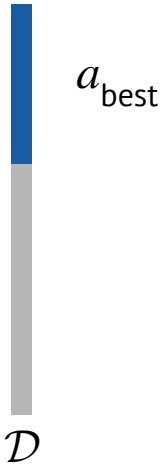


Learning Algorithm

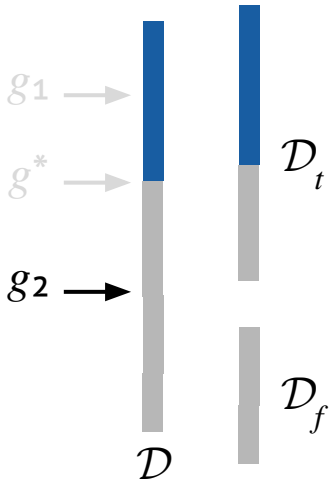
$$g_{\text{best}} = \underset{g \in \mathit{Guards}}{\arg\max}\ \mathit{InfGain}(\mathcal{D}, g, a_{\text{best}})$$

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

refine analysis



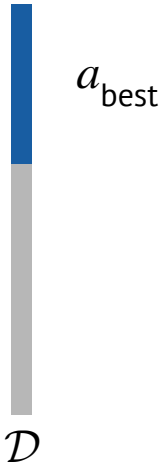
Find split that separates a_{best}



Learning Algorithm

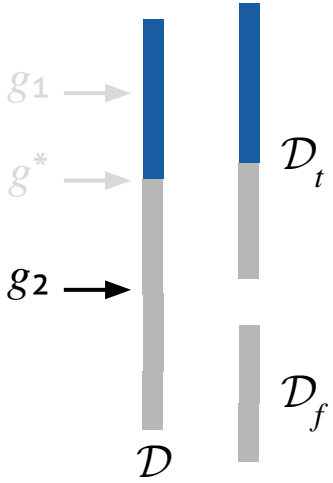
$$g_{\text{best}} = \arg\max_{g \in \mathit{Guards}} \mathit{InfGain}(\mathcal{D}, g, a_{\text{best}})$$

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



refine analysis

Find split that separates a_{best}



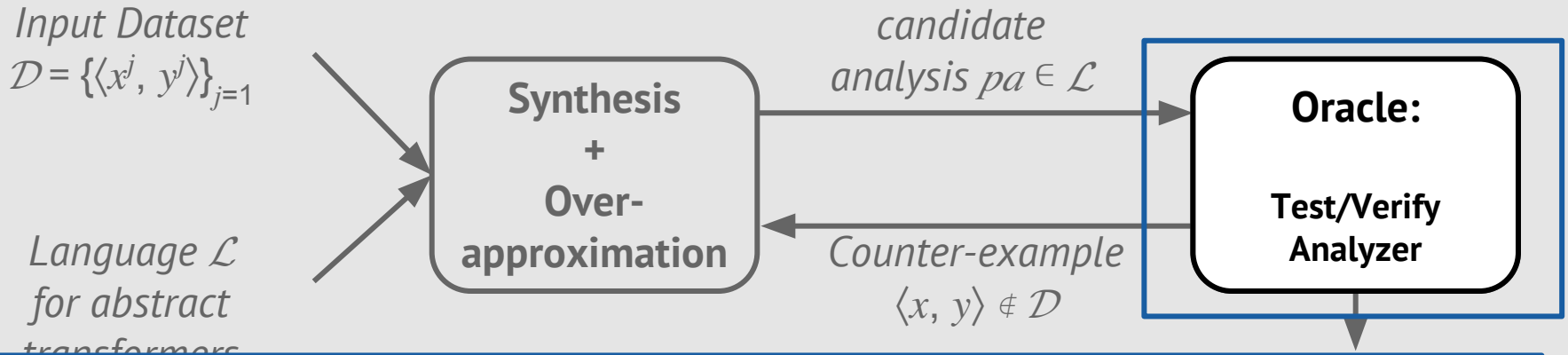
$\mathit{InfGain}(\mathcal{D}, g, a_{\text{best}}) = 0$

no split reduces entropy

↓

approximate($\mathcal{D}$)

Learning: Decision Trees + CEGIS



How to find complex counter-examples quickly?
How to efficiently explore hard to find corner cases?

Naive Approach: Random Fuzzing

1. Pick a random training example

$$\langle x, y \rangle \in \mathcal{D}$$

2. Mutate the input randomly

x → x'

3. Obtain the correct label

Execute x' → $\mathcal{D}'$

4. Check for correctness

$$\forall \langle x, y \rangle \in \mathcal{D}' \,.\, \alpha(y) \sqsubseteq pa(x)$$

5. Repeat

Naive 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

Exponential Number of Choices

Slow

When to stop?

The Oracle: Testing an Analyzer

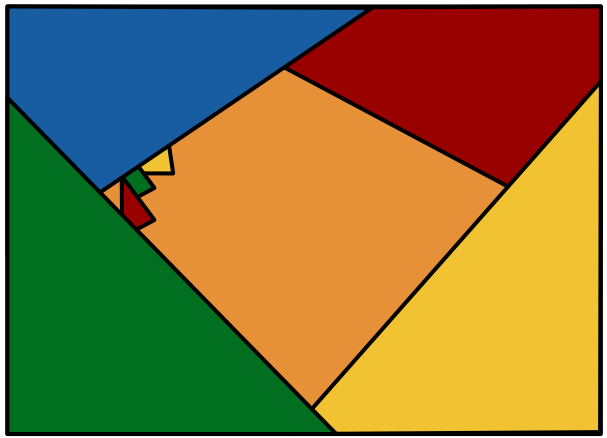
Key Idea: Take advantage of candidate analysis pa

$\mathcal{T}$

How to sample from space of all programs?

The Oracle: Testing an Analyzer

execution path coverage of pa

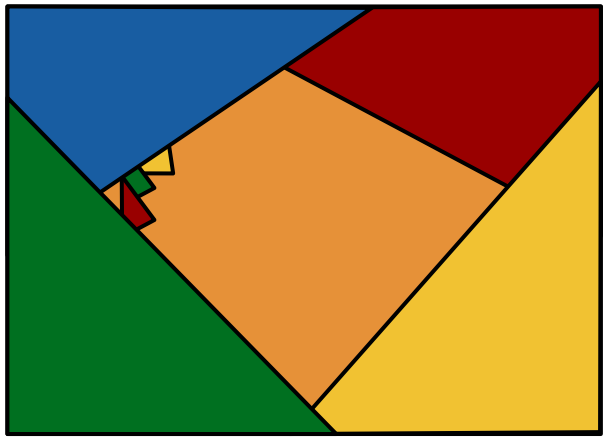


$\mathcal{T}$

The Oracle: Testing an Analyzer

execution path coverage of pa

mutate only parts that affect pa



$\mathcal{T}$

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

Query

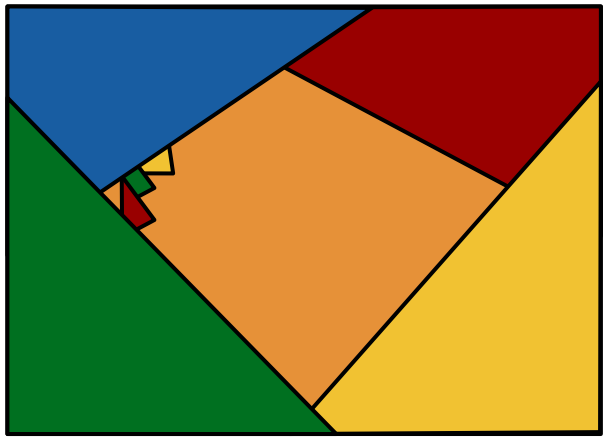
Locations accessed by the analysis

The Oracle: Testing an Analyzer

execution path coverage of pa

mutate only parts that affect pa

select relevant program mutations



$\mathcal{T}$

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

Query

Locations accessed by the analysis

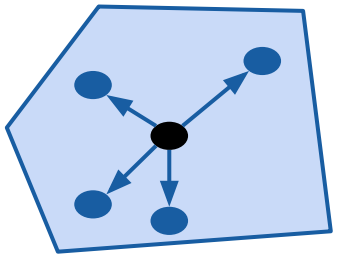
Modification via Equivalence Modulo Abstraction (EMA)

Modification via Global Jumps

The Oracle: Testing an Analyzer

Modifications via Equivalence Modulo Abstraction (EMA)

Semantic preserving mutations



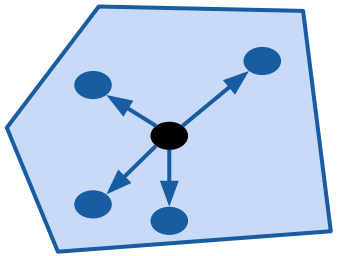
Adding dead code
Renaming variables
Renaming user defined functions
Side-effect free expressions

$\mathcal{T}$

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

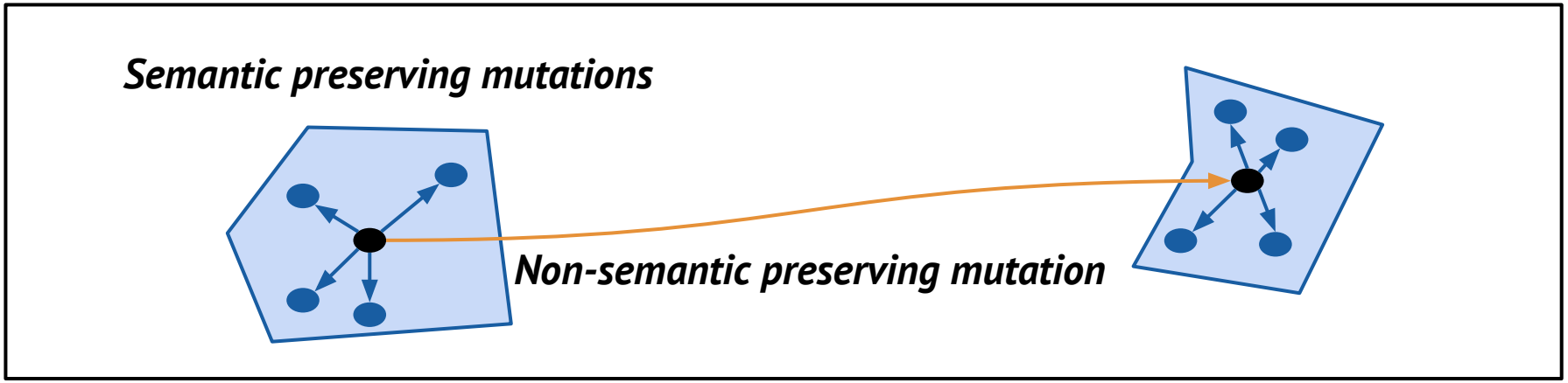
labels y can be reused

$\mathcal{T}$

The Oracle: Testing an Analyzer

Modifications via Equivalence Modulo Abstraction (EMA)

Modifications via Global Jumps



$\mathcal{T}$

Evaluation

ECMAScript (ECMA-262) Conformance Suite

15 675

Programs

Points-to Analysis

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

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

Allocation Site Analysis

```
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) { ... }
```

Approach Instantiation for Points-to Analysis

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

$y \leftarrow$ **concrete object id**

Language $\mathcal{L}$ for abstract transformers

```
a ∈ Actions   ::= ε | Move; a
Move_Core     ::= Up, Left, Right,
                  DownFirst, DownLast, Top
g ∈ Guards    ::= ε | Move; g | Write; g
WriteOp       ::= WriteValue, WriteType, WritePos,
                  HasLeftSibling, HasRightSibling, HasChild
```

Synthesis

$(\mathcal{H}, \sqsubseteq)$, $\boldsymbol{\alpha}$, $\boldsymbol{\gamma}$

Oracle

semantic preserving mutations	**non-semantic preserving mutations**
Adding dead code	Add method arguments
Renaming variables	Add method parameters
Renaming user defined functions	Change program constants
Side-effect free expressions	

Learned Points-to Analysis

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

rules missed by Facebook Flow

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

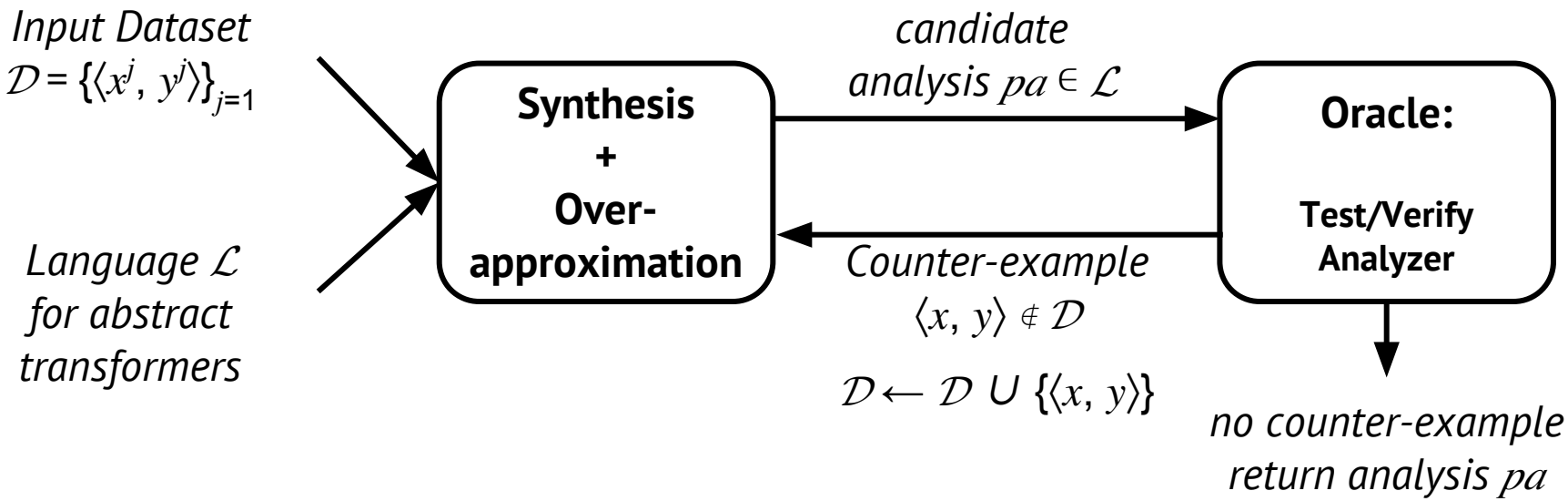
7 hours
time to find counter-examples

Overview of Learned Analysis

```
if HasPrevNodeValue then ⊤
elif WriteType == CallExpression then
  if Up WriteType == ExpressionStatement then
    ⊤       // 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
  ...
```

Learning a Static Analyzer from Data

Learns practical abstract transformers missed by existing state-of-the-art analyzers



Learning Points-to Analysis: Intuition

```
var = new ...
to = from
to = base.fld
base.fld = from
...
```

```
Alloc(var: V, heap: H, inMeth: M)
Move(to: V, from: V)
Load(to: V, base: V, fld: F)
Store(base: V, fld: F, from: V)
...
```