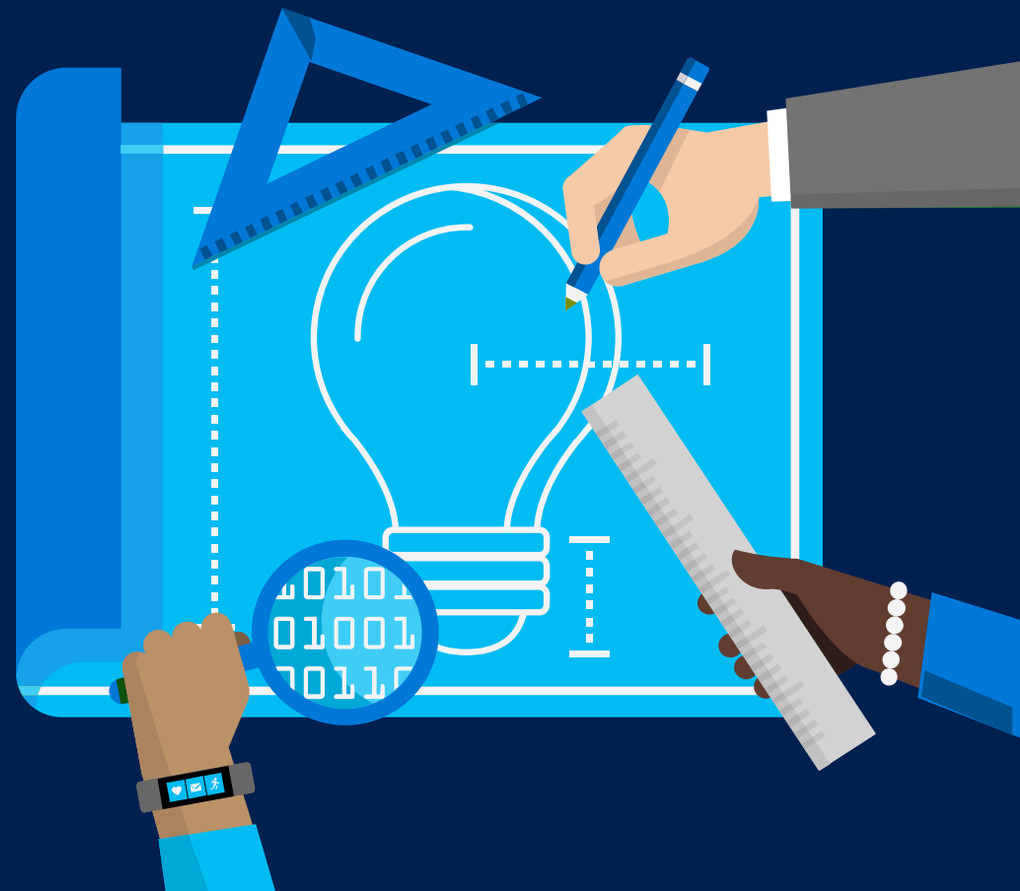
Microsoft

Machine Learning for Smart Software Engineering Tools

Miltos Allamanis

Microsoft Research, Cambridge, UK



Outline

The Big Picture

Related Work

Practical Considerations

The Boundaries of Our Methods

Code as…

Data → Software Engineering (SE) Tools

Machine Learning (ML) component → Artificial Intelligence (AI) Tool

Code Understanding

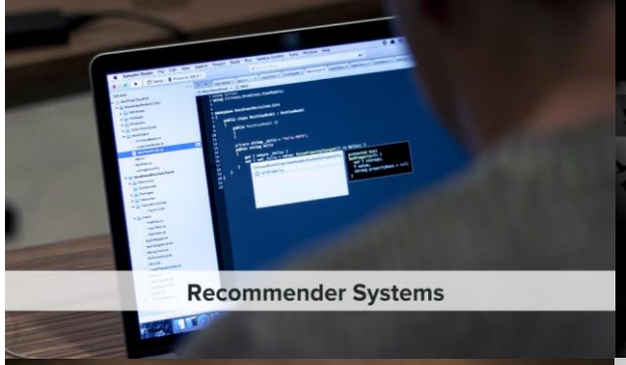
Code Synthesis

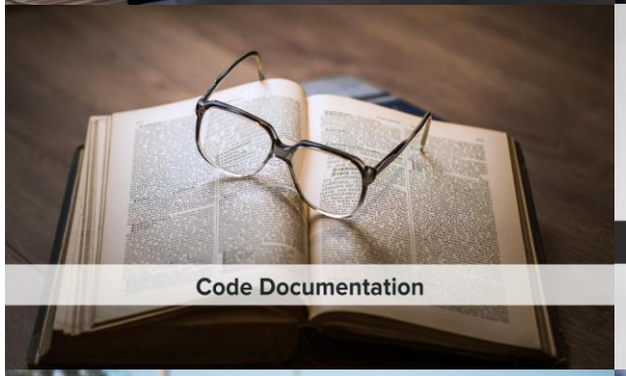
Source Code is Bimodal

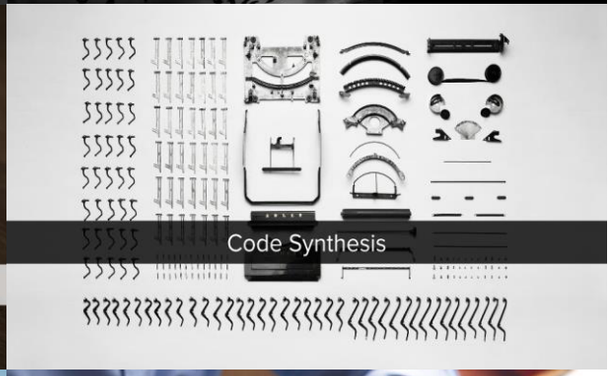
Research in ML+Code

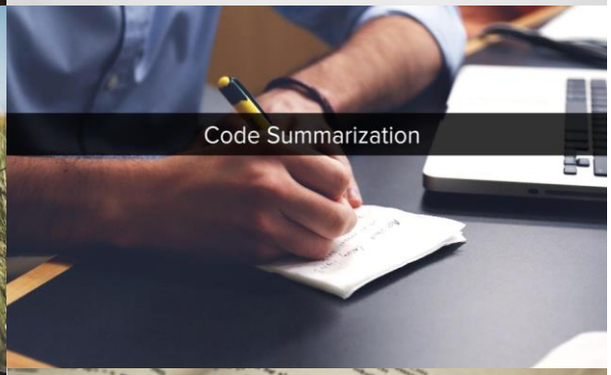
- Infer latent intent
- Ambiguous information
- Learned heuristics

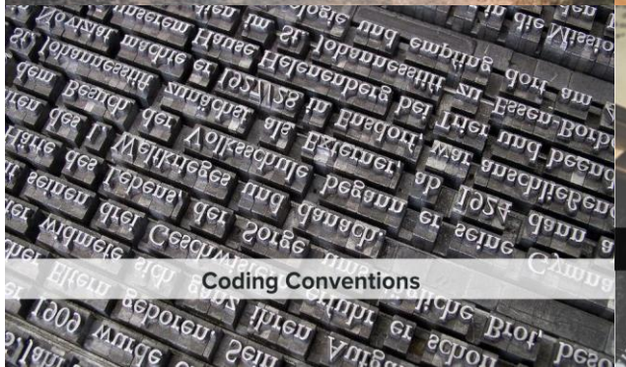
https://ml4code.github.io

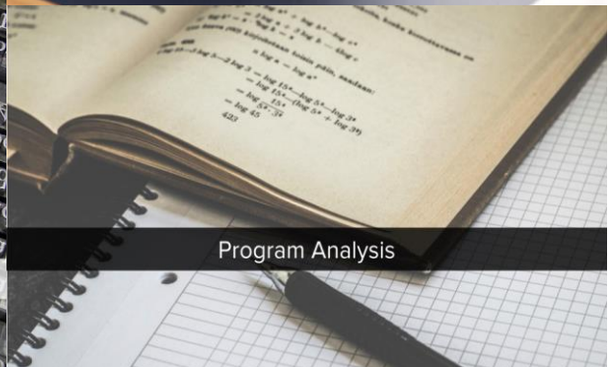












1

A Survey of Machine Learning for Big Code and Naturalness

MILTIADIS ALLAMANIS, Microsoft Research
EARL T. BARR, University College London
PREMKUMAR DEVANBU, University of California, Davis
CHARLES SUTTON, University of Edinburgh and The Alan Turing Institute

Research at the intersection of machine learning, programming languages, and software engineering has recently taken important steps in proposing learnable probabilistic models of source code that exploit code's abundance of patterns. In this article, we survey this work. We contrast programming languages against natural languages and discuss how these similarities and differences drive the design of probabilistic models. We present a taxonomy based on the underlying design principles of each model and use it to navigate the literature. Then, we review how researchers have adapted these models to application areas and discuss cross-cutting and application-specific challenges and opportunities.

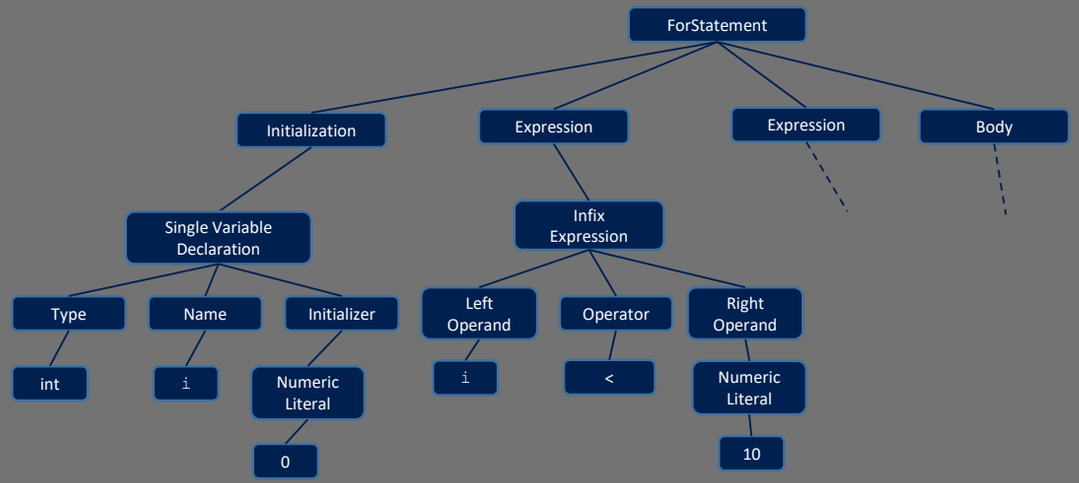
CCS Concepts: • Computing methodologies → Machine learning: Natural language processing: • Soft-

Code Representations

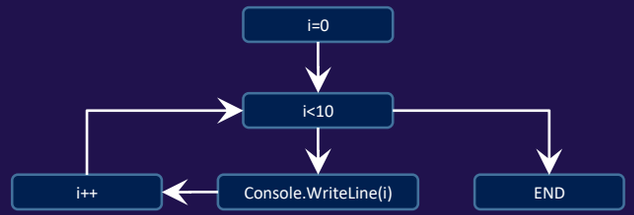
Token

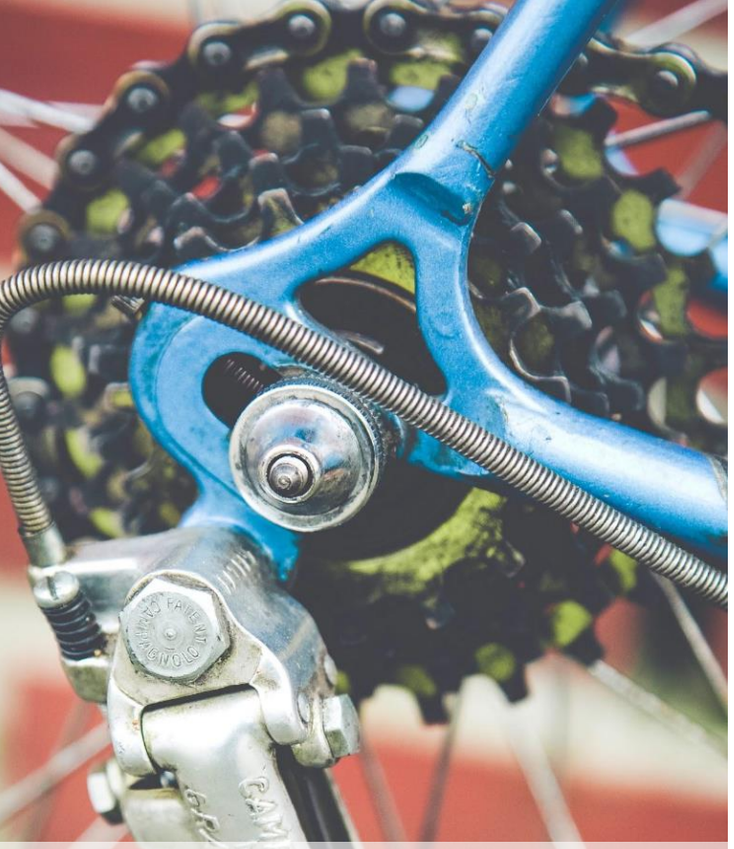
```
for (int i = 0; i < 10; i++){
    Console.WriteLine(i);
}
```

Syntax

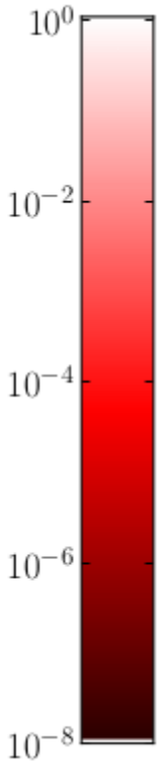


Graph





Programming Language Models

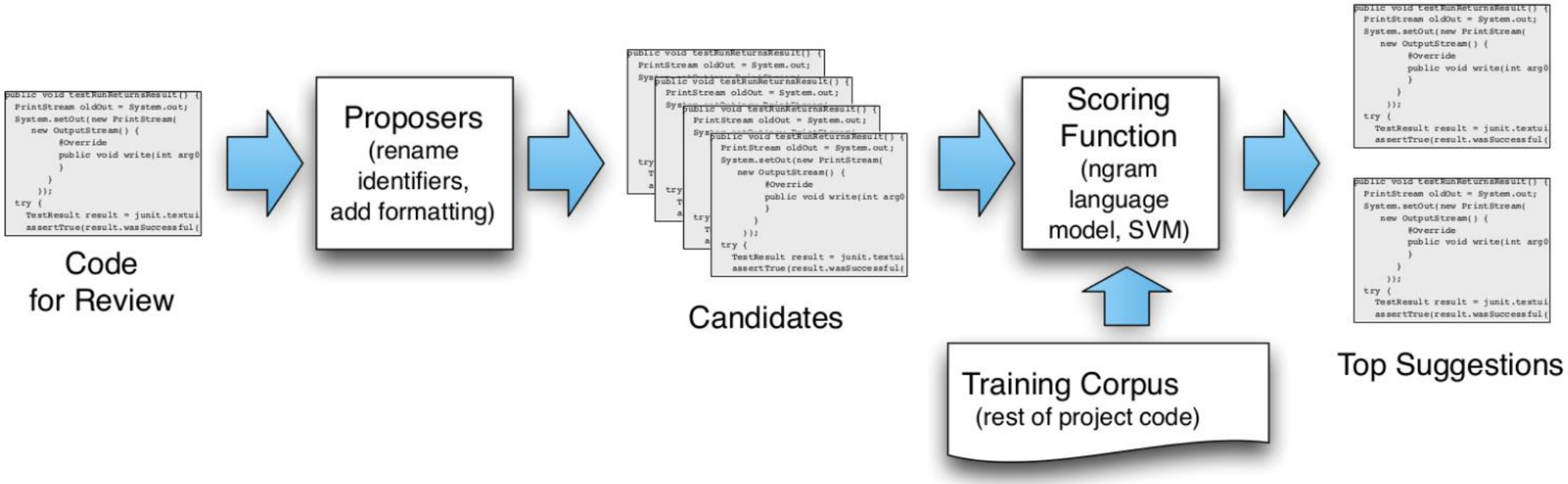



```
1  public void execute(Runnable task) {
2      if (task == null)
3          throw new NullPointerException();
4      ForkJoinTask<?> job;
5      if (task instanceof ForkJoinTask<?>) // avoid re-wrap
6          job = (ForkJoinTask<?>) task;
7      else
8          job = new ForkJoinTask.AdaptedRunnableAction(task);
9      doSubmit(job);
10 }
```

Hindle *et al*. "On the naturalness of software" ICSE 2012

Image from Allamanis & Sutton, MSR 2013

Naming Conventions



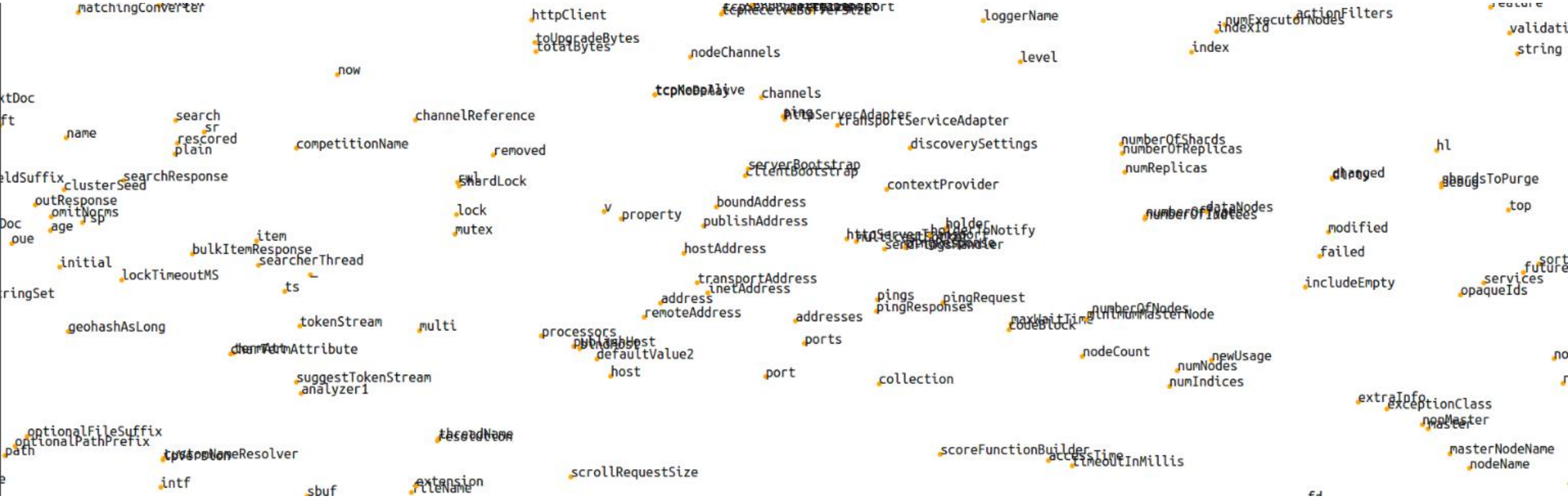

```
ForkJoinTask<?> job;
if (task instanceof ForkJoinTask<?>) // avoid re-wrap
    job = (ForkJoinTask<?>) task;
else
    job = new
ForkJoinTask.AdaptedRunnableAction(task);
externalPush(job);
```

Suggestions

1. job (30%)
2. task (20%)
3. tsk (15%)

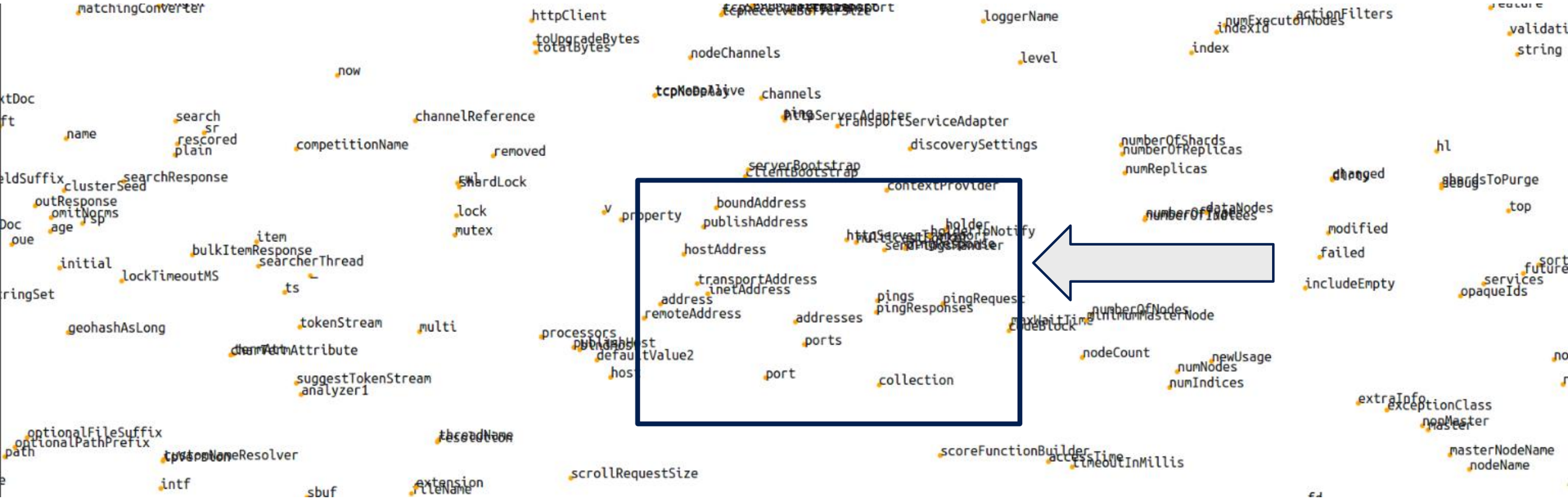
Allamanis *et al.* FSE 2014, 2015 and other recent work

Embedding Visualization



http://groups.inf.ed.ac.uk/cup/naturalize

Embedding Visualization

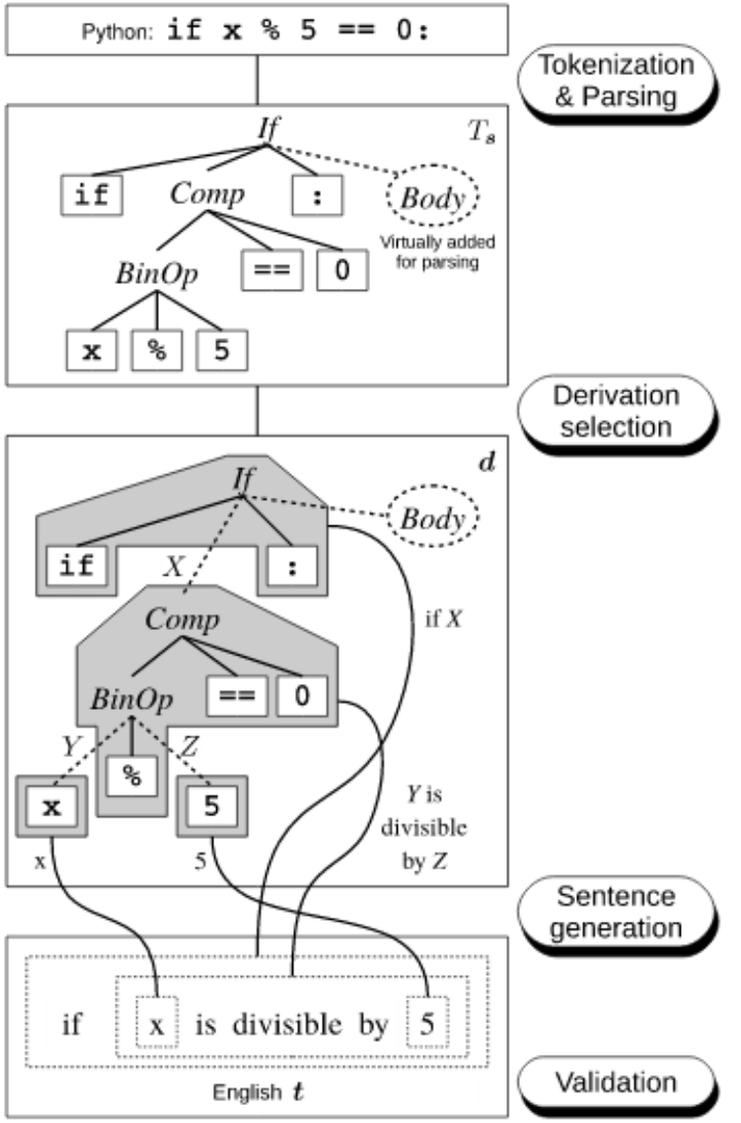


http://groups.inf.ed.ac.uk/cup/naturalize

Embedding Visualization

"Translating" Code

Subphase	Output	Example output
Parallel data collection	Method pairs	**C#:** `Console . WriteLine ( "Hello World!" ) ;` **Java:** `System . out . println ( "Hello World!" ) ;`
Word alignment	Aligned method pairs	**C#:** `Console . WriteLine ( "Hello World!" ) ;` **Java:** `System . out . println ( "Hello World!" ) ;`
Phrase table construction	Phrase table	**C# phrase** | **Java phrase** | **Score** i.e. Pr(**C# phrase** | **Java phrase**) `Console` | `System . out` | 0.8 `WriteLine` | `println` | 0.5 `.` | `.` | 0.7 `(` | `(` | 0.9 ... | ... | ...



Raychev *et al.* 2014 Nguyen *et al.* 2013,2015 Oda *et al.* ASE 2015

DeepCoder

Program DSL, operating on integers and lists:

- List first order: `head, last, take, drop, access, minimum, maximum, sum`
- List higher order: `map, filter, count, zipWith, scanl1`
- Int first order: `(+1), (-1), (*2), (/2), (+), (-), (*), min, (>0), …`

```
a ← [int]
b ← FILTER (<0) a
c ← MAP (*4) b
d ← SORT c
e ← REVERSE d
```

An input-output example:

Input:

```
[-17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11]
```

Output:

```
[-12, -20, -32, -36, -68]
```

Balog *et al.* *"DeepCoder: Learning to Write Programs"*, 2016

DeepCoder - Results

Table 1: Search speedups on programs of length $T = 3$ due to using neural network predictions.

Timeout needed	**DFS**			**Enumeration**			λ^2			**Sketch**		**Beam**
to solve	20%	40%	60%	20%	40%	60%	20%	40%	60%	20%	40%	20%
Baseline	$41ms$	$126ms$	$314ms$	$80ms$	$335ms$	$861ms$	$18.9s$	$49.6s$	$84.2s$	$>10^3s$	$>10^3s$	$>10^3s$
DeepCoder	$2.7ms$	$33ms$	$110ms$	$1.3ms$	$6.1ms$	$27ms$	$0.23s$	$0.52s$	$13.5s$	$2.13s$	$455s$	$292s$
Speedup	**15.2×**	**3.9×**	**2.9×**	**62.2×**	**54.6×**	**31.5×**	**80.4×**	**94.6×**	**6.2×**	**>467×**	**>2.2×**	**>3.4×**

Table 2: Search speedups on programs of length $T = 5$

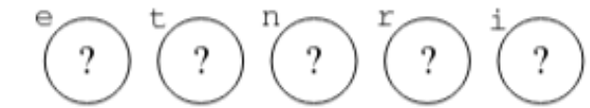
Timeout needed	**DFS**			**Enumeration**			λ^2
to solve	20%	40%	60%	20%	40%	60%	20%
Baseline	$163s$	$2887s$	$6832s$	$8181s$	$>10^4s$	$>10^4s$	$463s$
DeepCoder	$24s$	$514s$	$2654s$	$9s$	$264s$	$4640s$	$48s$
Speedup	**6.8×**	**5.6×**	**2.6×**	**907×**	**>37×**	**>2×**	**9.6×**

Balog *et al. "DeepCoder: Learning to Write Programs"*, 2016

```
function chunkData(e, t) {
  var n = [];
  var r = e.length;
  var i = 0;
  for (; i < r; i += t) {
    if (i + t < r) {
      n.push(e.substring(i, i + t));
    } else {
      n.push(e.substring(i, r));
    }
  }
  return n;
}
```

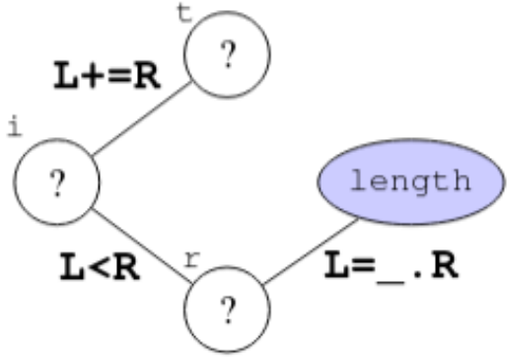
(a) JavaScript program with minified identifier names

Unknown properties (variable names):

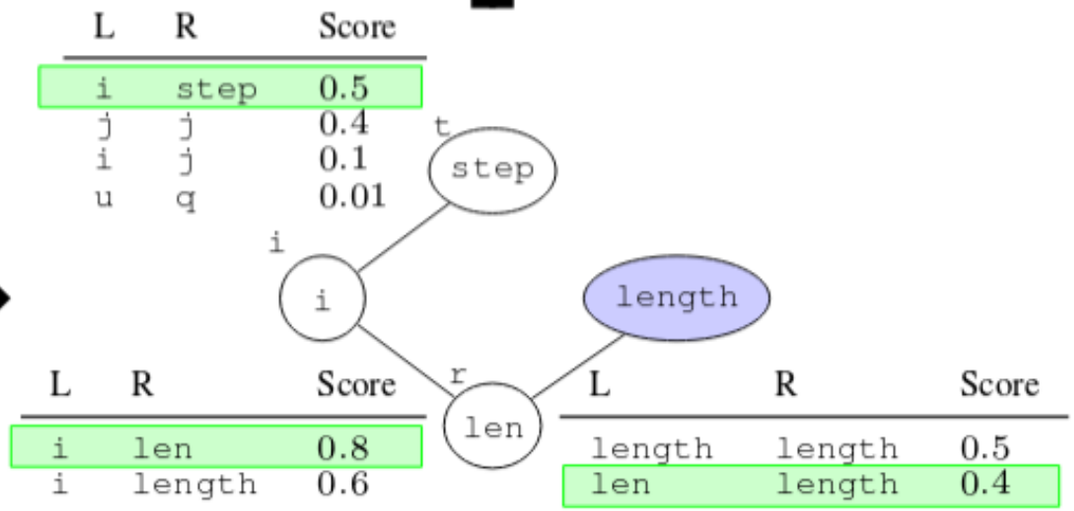


Known properties (constants, APIs):

(b) Known and unknown name properties



(c) Dependency network



(d) Result of MAP inference

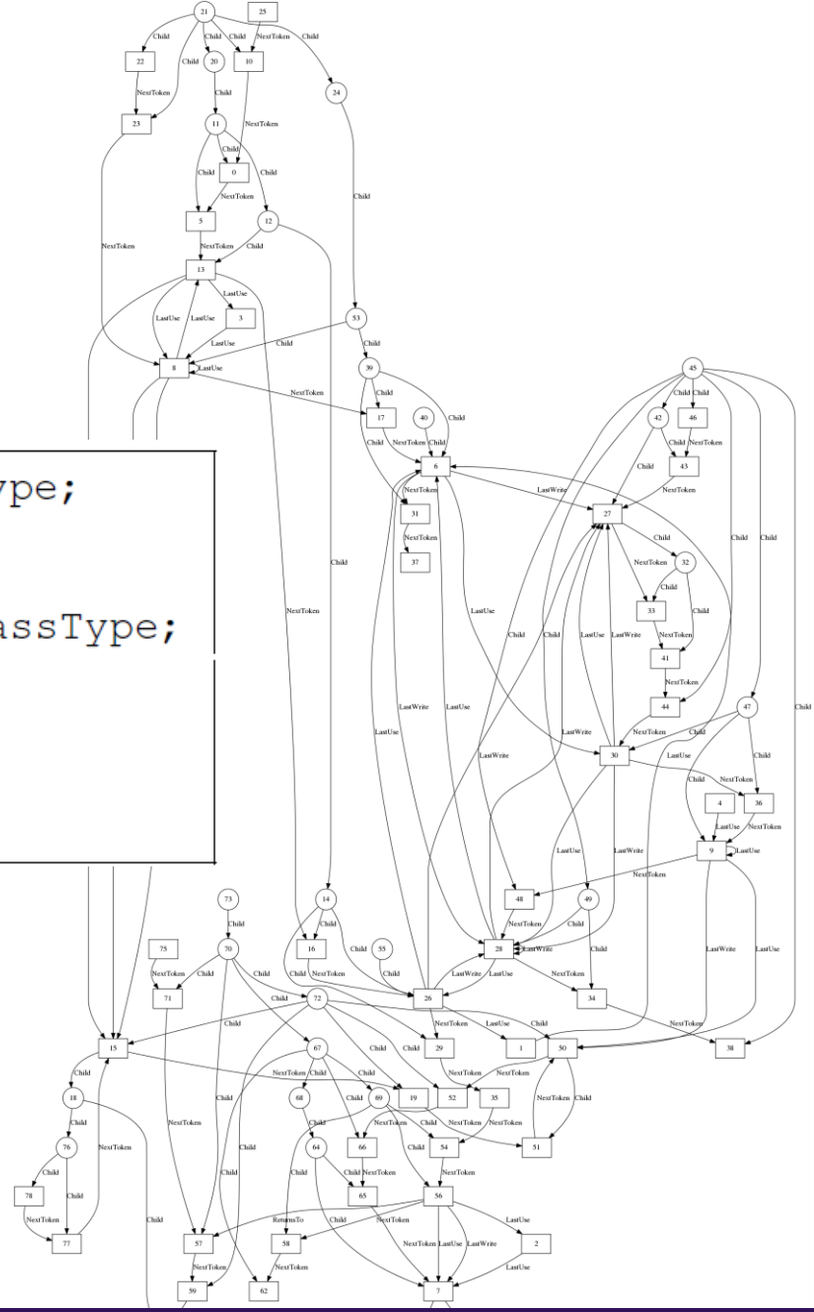
```
/* str: string, step: number, return: Array */
function chunkData(str, step) {
  var colNames = []; /* colNames: Array */
  var len = str.length;
  var i = 0; /* i: number */
  for (; i < len; i += step) {
    if (i + step < len) {
      colNames.push(str.substring(i, i + step));
    } else {
      colNames.push(str.substring(i, len));
    }
  }
  return colNames;
}
```

(e) JavaScript program with new identifier names and types

Raychev *et al.* "*Predicting Program Properties from Big Code*", 2015

Finding Variable Misuse Bugs

```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(          );

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```



Possible type-correct options: `clazz, first`

Allamanis *et al. "Learning to Represent Programs with Graphs"*, 2017

```
359         private string GetTestOutputFilePath(string filepath)
360         {
361             string outputFilePath = @"Z:\";
362 
363             try
364             {
365                 outputFilePath = Path.GetDirectoryName(_filePath);
366             }
367             catch (ArgumentException)
368             {
369             }
370 
371             if (string.IsNullOrEmpty(outputFilePath))
372             {
373                 outputFilePath = @"Z:\";
374             }
375 
376             return this.CompilationOptions == null ? "" : Path.Combine(outputFilePath, this.AssemblyN
377         }
378     }
379 }
```

https://github.com/dotnet/roslyn/pull/23437

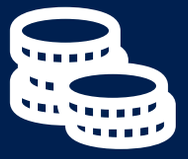
Practical Considerations

Explainability

Metrics

Train-Use & Feedback Loop

Low-Resource Environments



Train/Use Costs

Confidentiality

The Frontiers of Our Methods

```
float getAspectRatio()
```

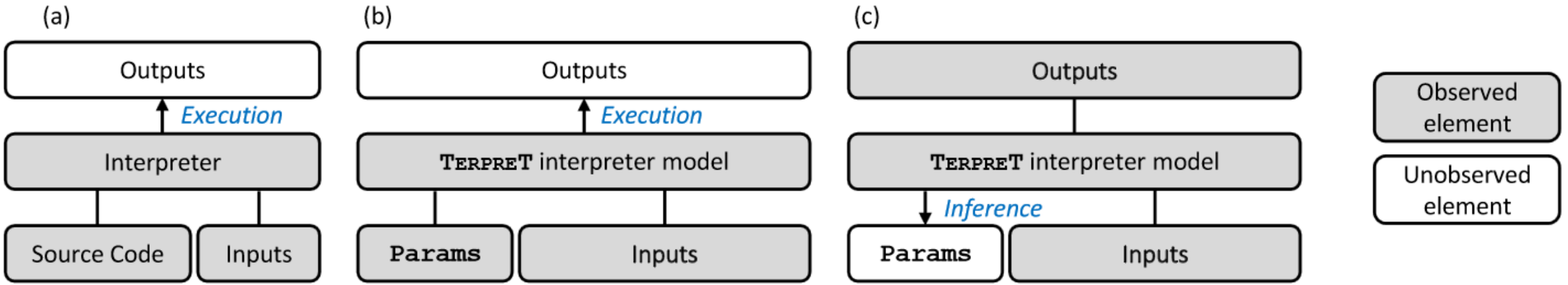
```
return (height == 0) ?
    Float.NaN : width / height;
```

Predictions

- `get, UNK` (9%)
- `get, height` (8.7%)
- `get, width` (6.5%)
- `get` (5.7%)
- `get,size` (4.2%)

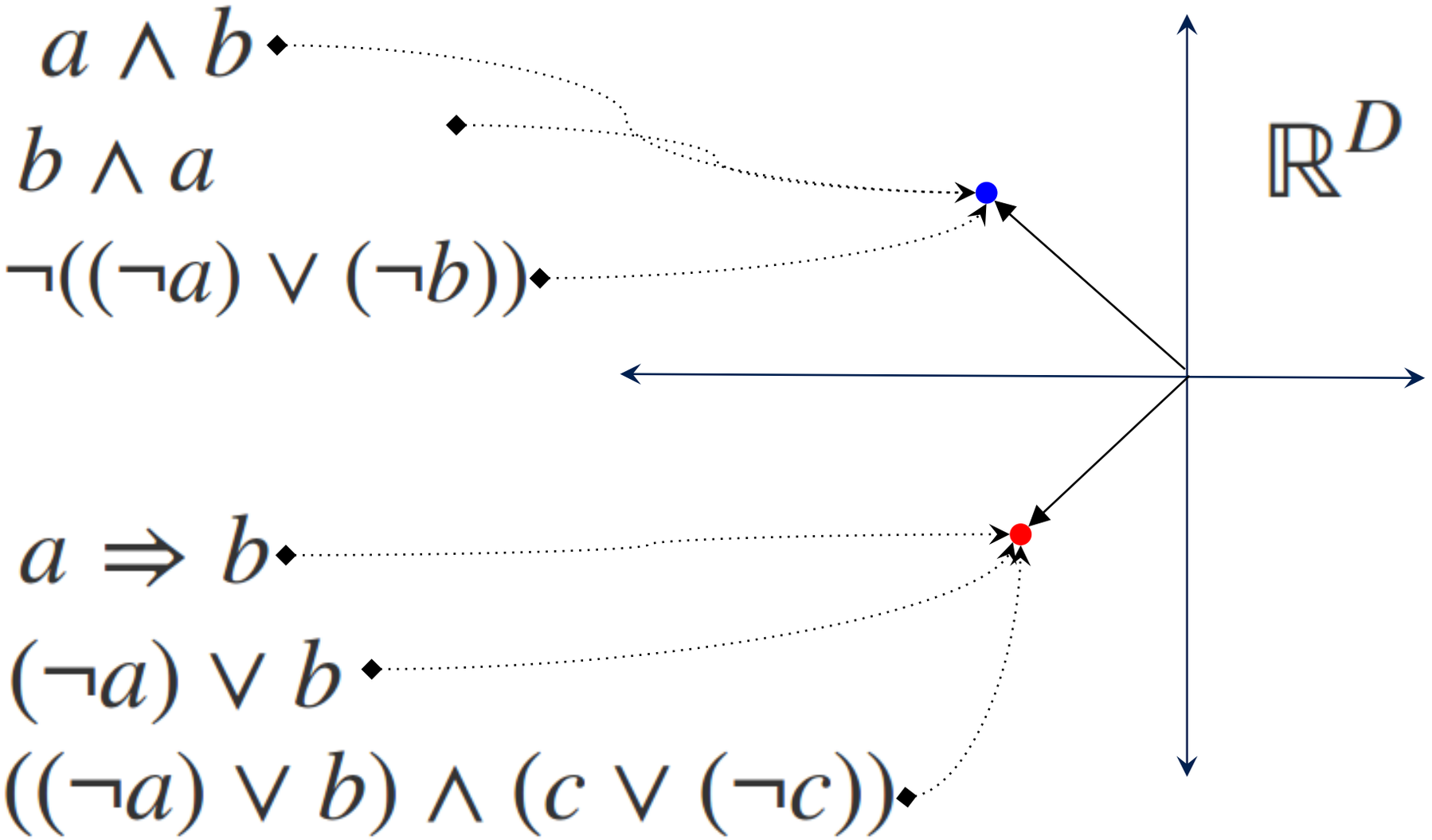
Allamanis *et al*, "A Convolutional Attention Network for Extreme Summarization of Source Code", ICML 2016

TerpreT



Gaunt *et al*, "TerpreT: A Probabilistic Programming Language for Program Induction", 2016

Semantic Continuous Representations



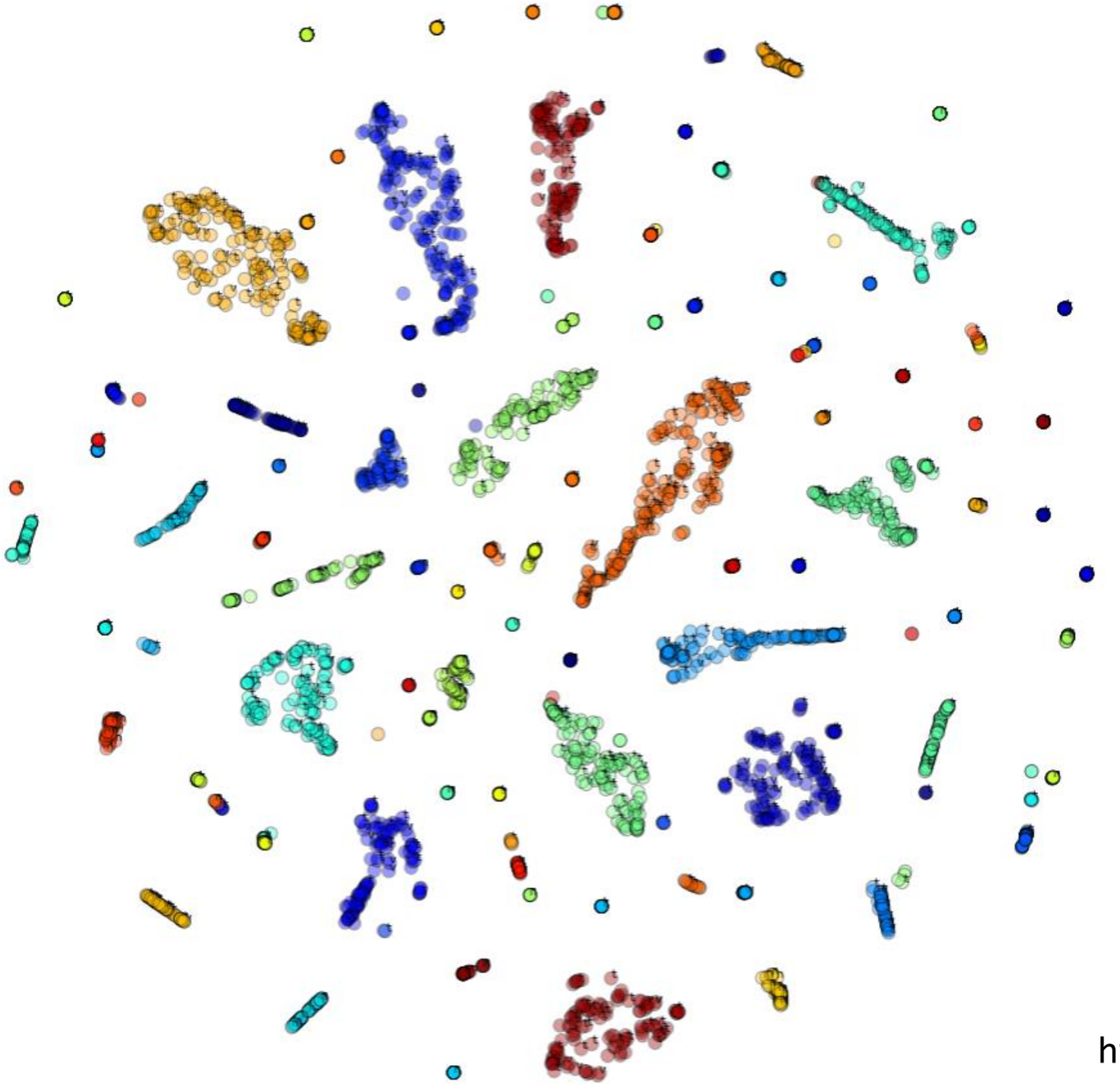
Syntax-driven but **small** changes in syntax can lead in **large** changes in semantics

Zaremba *et al.* 2014
Allamanis et al. 2017

BOOL8

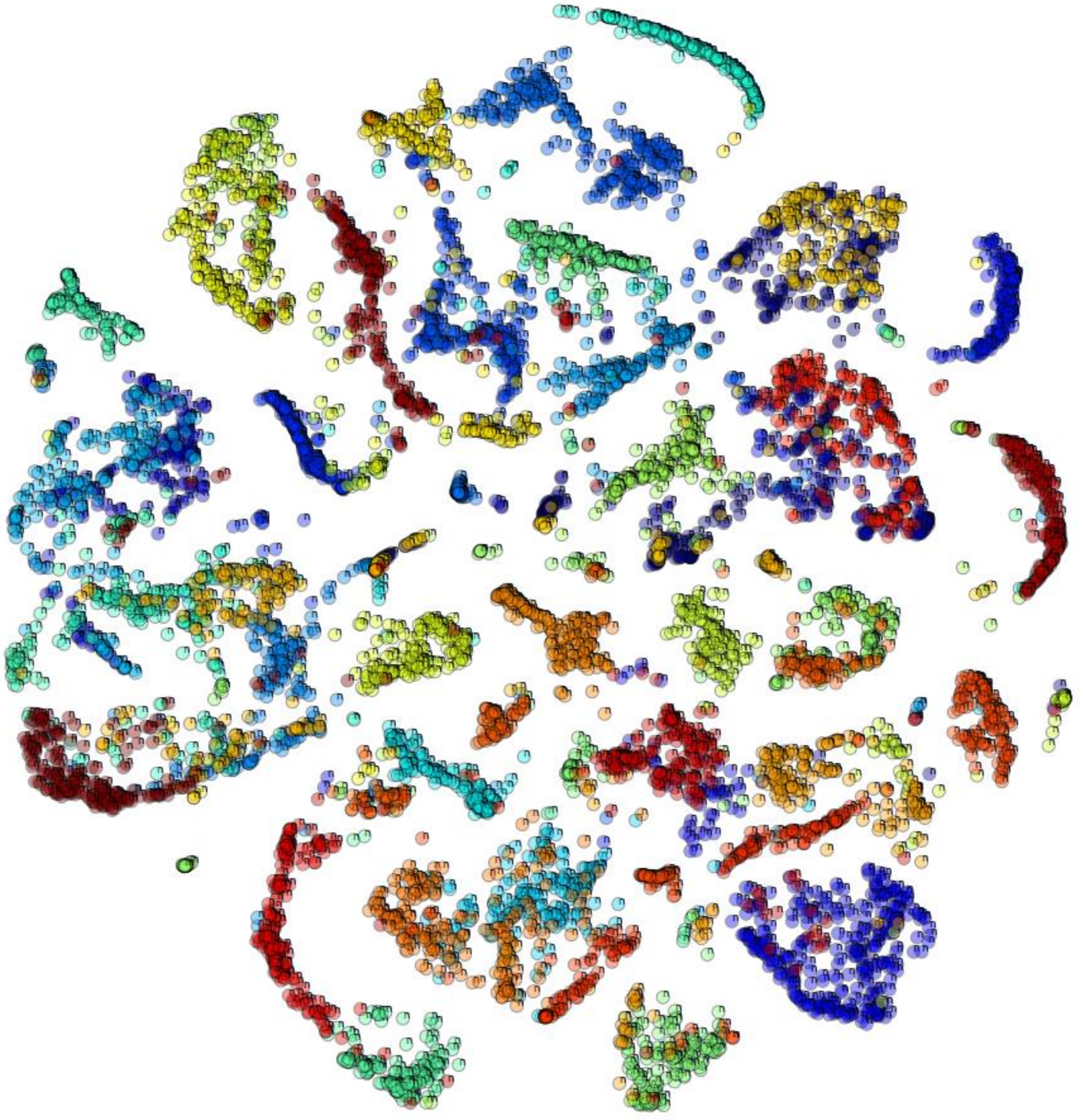
$(\neg a) \wedge (\neg b)$	$(\neg a \wedge \neg c) \vee (\neg b \wedge a \wedge c) \vee (\neg c \wedge b)$	$(\neg a) \wedge b \wedge c$
$a \neg((\neg a) \Rightarrow ((\neg a) \wedge b))$	$c \oplus (((\neg a) \Rightarrow a) \Rightarrow b)$	$\neg((\neg b) \vee ((\neg c) \vee a))$
$\neg((b \vee (\neg(\neg a))) \vee b)$	$\neg((b \oplus (b \vee a)) \oplus c)$	$((a \vee b) \wedge c) \wedge (\neg a)$
$(\neg a) \oplus ((a \vee b) \oplus a)$	$\neg((\neg(b \vee (\neg a))) \oplus c)$	$(\neg((\neg(\neg b)) \Rightarrow a)) \wedge c$
$(b \Rightarrow (b \Rightarrow a)) \wedge (\neg a)$	$((b \vee a) \oplus (\neg b)) \oplus c)$	$(c \wedge (c \Rightarrow (\neg a))) \wedge b$
$((\neg a) \Rightarrow b) \Rightarrow (a \oplus a)$	$(\neg((b \oplus a) \wedge a)) \oplus c$	$b \wedge (\neg(b \wedge (c \Rightarrow a)))$

False	$(\neg a) \wedge (\neg b) \vee (\wedge c)$	$\neg a \vee b$
$(a \oplus a) \wedge (c \Rightarrow c)$	$(a \Rightarrow (\neg c)) \oplus (a \vee b)$	$a \Rightarrow ((b \wedge (\neg c)) \vee b)$
$(\neg b) \wedge (\neg(b \Rightarrow a))$	$(a \Rightarrow (c \oplus b)) \oplus b$	$\neg(\neg((b \vee a) \Rightarrow b))$
$b \wedge ((a \vee a) \oplus a)$	$b \oplus (a \Rightarrow (b \oplus c))$	$(\neg a) \oplus (\neg(b \Rightarrow (\neg a)))$
$((\neg b) \wedge b) \oplus (a \oplus a)$	$(b \vee a) \oplus (x \Rightarrow (\neg a))$	$b \vee (\neg((\neg b) \wedge a))$
$c \wedge ((\neg(a \Rightarrow a)) \wedge c)$	$b \oplus ((\neg a) \vee (c \oplus b))$	$\neg((a \Rightarrow (a \oplus b)) \wedge a)$



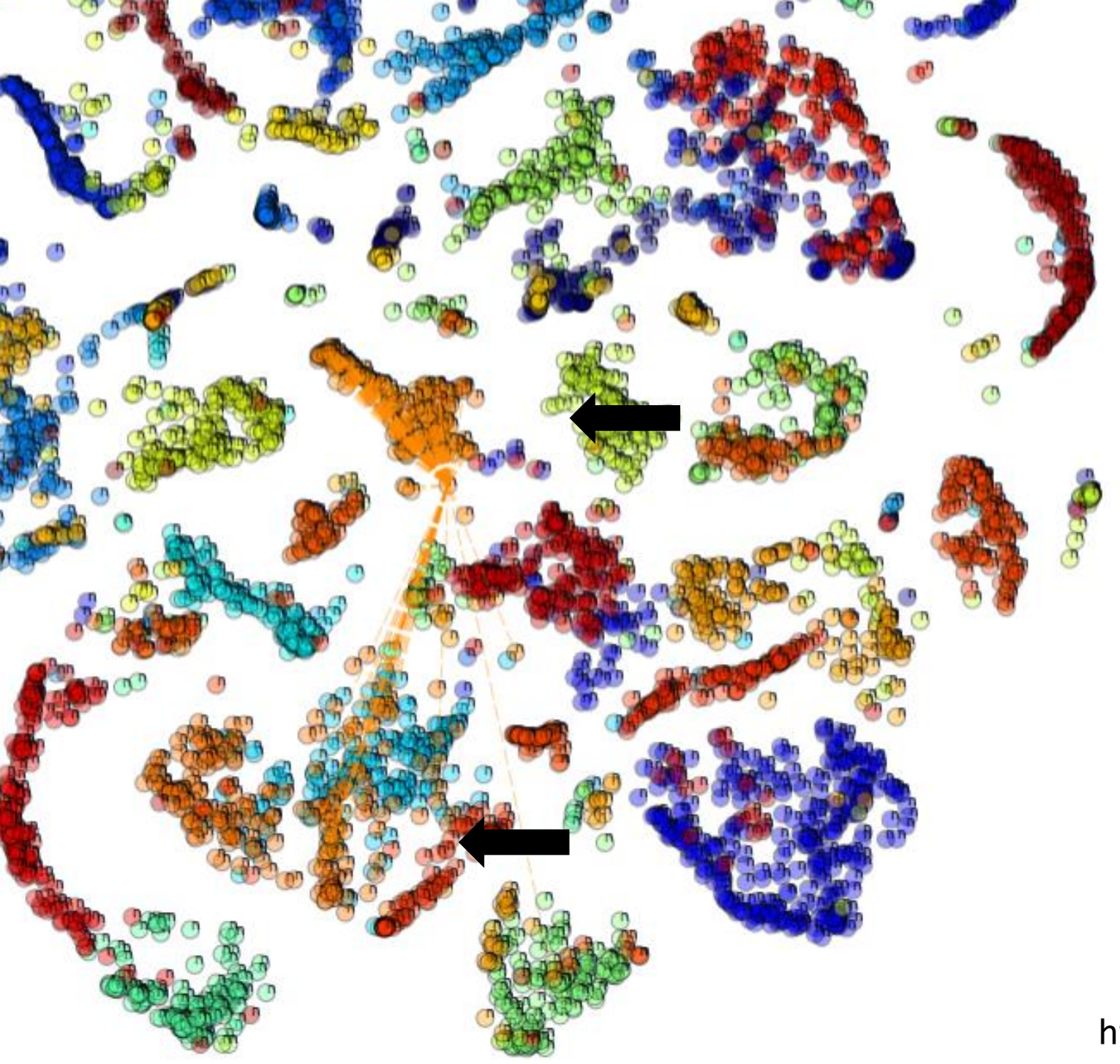
t-SNE Visualization
SimpPoly8

http://groups.inf.ed.ac.uk/cup/semvec/



t-SNE Visualization
Bool10-UnseenTest

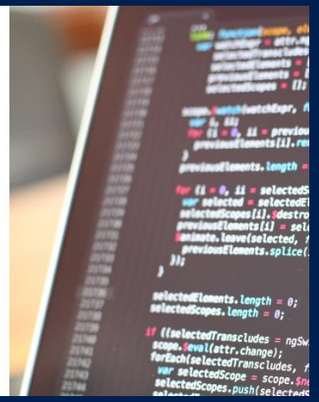
http://groups.inf.ed.ac.uk/cup/semvec/



t-SNE Visualization
Bool10-UnseenTest

http://groups.inf.ed.ac.uk/cup/semvec/

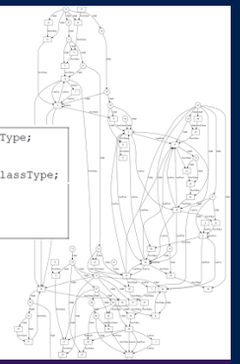
Source Code is Bimodal



Finding Variable Misuse Bugs

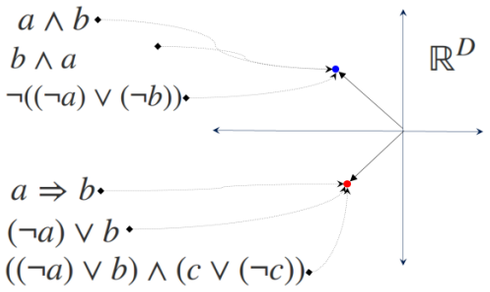
```
var clazz=classTypes["Root"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull(clazz);

var first=classTypes["RecClass"].Single() as JsonCodeGenerator.ClassType;
Assert.NotNull([redacted]);

Assert.Equal("string", first.Properties["Name"].Name);
Assert.False(clazz.Properties["Name"].IsArray);
```

Possible type-correct options: `clazz, first`



Allamanis *et al.* "*Learning to Represent Programs with Graphs*", 2017

Semantic Continuous Representations



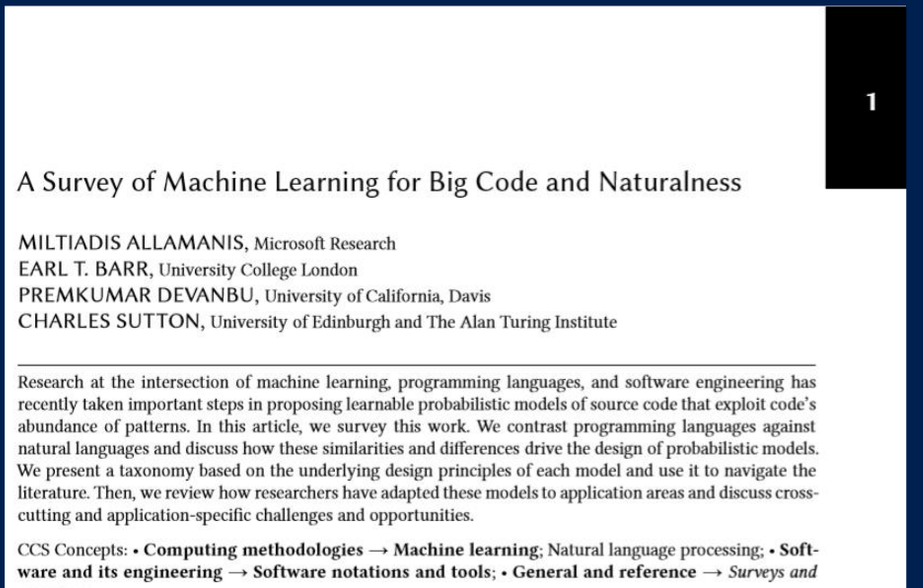
Syntax-driven but **small** changes in syntax can lead in **large** changes in semantics

Zaremba *et al.* 2014
Allamanis et al. 2017

Practical Considerations

- Explainability
- Metrics
- Train-Use & Feedback Loop
- Low-Resource Environments
- Train/Use Costs
- Confidentiality

https://ml4code.github.io



1

A Survey of Machine Learning for Big Code and Naturalness

MILTIADIS ALLAMANIS, Microsoft Research
EARL T. BARR, University College London
PREMKUMAR DEVANBU, University of California, Davis
CHARLES SUTTON, University of Edinburgh and The Alan Turing Institute

Research at the intersection of machine learning, programming languages, and software engineering has recently taken important steps in proposing learnable probabilistic models of source code that exploit code's abundance of patterns. In this article, we survey this work. We contrast programming languages against natural languages and discuss how these similarities and differences drive the design of probabilistic models. We present a taxonomy based on the underlying design principles of each model and use it to navigate the literature. Then, we review how researchers have adapted these models to application areas and discuss cross-cutting and application-specific challenges and opportunities.

CCS Concepts: • **Computing methodologies** → **Machine learning**; Natural language processing; • **Software and its engineering** → **Software notations and tools**; • **General and reference** → *Surveys and*

Closing Thoughts