Probabilistic Higher Order Grammar: Probabilistic Model for Code

Pavol Bielik, Veselin Raychev, Martin Vechev

ETH zürich

Write new code:
Code Completion

```
Camera camera = Camera.Open();
camera.SetOrientation(90);
              ?
```

Debug code:
Statistical Bug Detection

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

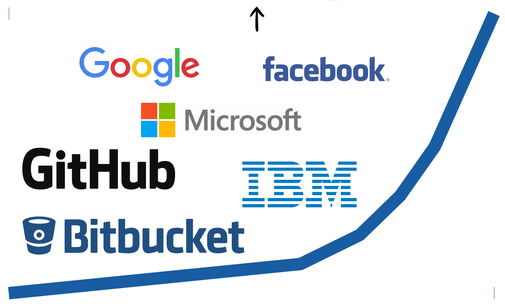
Port code:
Programming Language Translation

C#
```
Console.WriteLine('Hi');
...
```

Java
```
System.out.println('Hi');
...
```

↖ ↑ ↗ applications

Probabilistic Model



number of repositories
last 8 years

15 million repositories

Billions of lines of code

High quality, tested, maintained programs

How can we learn probabilistic models directly from data?

Efficient Learning
Trained as efficiently
as PCFGs and n-gram models

Widely Applicable
Agnostic to
programming language

Probabilistic Model

Flexible Representation
Conditioning for the predictions
is determined dynamically

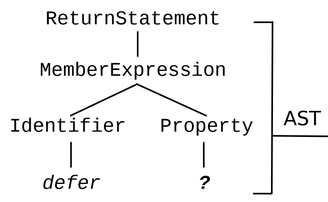
The Need for Better Conditioning

Context Free Grammar

$\alpha \rightarrow \beta_1 \dots \beta_n$

poor context ↘

	P
Property → x	0.05
Property → y	0.03
Property → promise	0.001



AST

```
awaitReset = function(){
  ...
  return defer.promise;
}

awaitRemoved = function(){
  fail(function(error){
    if (error.status === 401){
      ...
    }
    defer.reject(error);
  });
  ...
  return defer.?
}
```

richer context on which to condition

$\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n$

↑

richer context

	P
Property → promise [defer, reject, promise]	**0.67**
Property → notify [defer, reject, promise]	0.12
Property → resolve [defer, reject, promise]	0.11

Using Programs to Explain Data

In general:
Unrestricted programs
(Turing complete)

Our Work:
TCond Language for navigating over trees and accumulating context

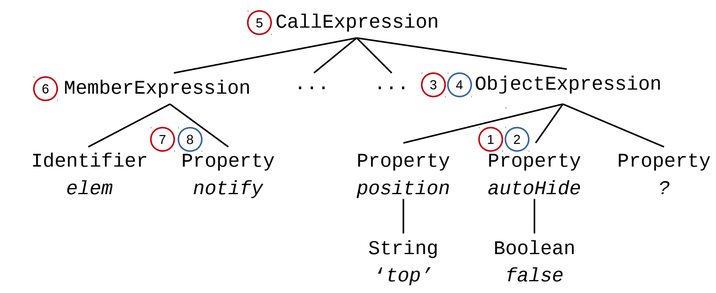
```
TCond   ::=  ε | WriteOp TCond | MoveOp TCond
MoveOp  ::=  Up, Left, Right, DownFirst, DownLast,
             NextDFS, PrevDFS, NextLeaf, PrevLeaf
             PrevNodeType, PrevNodeValue, PrevNodeContext
WriteOp ::=  WriteValue, WriteType, WritePos
```

Example TCond program for predicting property names

① Left, ② WriteValue, ③ Up, ④ WritePos, ⑤ Up, ⑥ DownFirst, ⑦ DownLast, ⑧ WriteValue

↑ Previous Property — ↑ Parameter Position — ↑ API name

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



Probabilistic Higher Order Grammar (PHOG)

Condition the predictions on richer context →

program that dynamically obtains the context for given query →

N: set of non-terminal symbols

Σ: set of terminal symbols

s: start symbol

R: set of rules in form: $\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n$

C: conditioning set where $\gamma \in C$

p: AST → C

q: R → $\mathbb{R}^+$ ← valid probability distribution

$$\sum_{\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n \in R} q(\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n) = 1$$

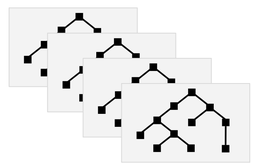
Source Code → AST → Execute program p → $p(\quad) = \gamma$ → Select most likely production rule

$$\arg\max_{\beta_1 \dots \beta_n \mid \alpha[\gamma] \rightarrow \beta_1 \dots \beta_n \in R} q(\alpha[\gamma] \rightarrow \beta_1 \dots \beta_n)$$

Example query for most likely prediction

Learning

Existing programs as trees



```
TCond   ::=  ε | WriteOp TCond | MoveOp TCond
MoveOp  ::=  Up, Left, Right, ...
WriteOp ::=  WriteValue, WriteType, ...
```

TCond Language

$$p_{best} = \arg\min cost(D, p) + \lambda \bullet \Omega(p)$$
$$p \in TCond$$

Search technique
Enumerative search
Genetic programming

↑ $|d| \ll |D|$
$|cost(d, p) - cost(D,p)| < \varepsilon$

↑ Regularization to avoid too complex programs

Representative sampling

Raychev, V., Bielik, P., Vechev, M. and Krause, A.
Learning Programs from Noisy Data. POPL '16, ACM

Evaluation

GitHub — 150k JavaScript Programs
http://www.srl.inf.ethz.ch/js150.php →
100k: Training Set (1.07 • 10^8 completion queries)
50k: Evaluation Set (5.3 • 10^7 completion queries)

Code Completion Error Rate

176 unique values ↖ ↗ 10^9 unique values

	Non-Terminals	Terminals
PCFG	48.5%	49.9%
n-gram	30.8%	28.7%
Naive Bayes	41.6%	45.8%
SVM	32.5%	29.5%
PHOG	**25.9%**	**18.5%**

	Training Time	Queries per Second
PCFG	1 min	71 000
n-gram	4 min	15 000
Naive Bayes	3 min	10 000
SVM	36 hours	12 500
PHOG	**162 + 3 min**	**50 000**

Code Completion Examples

Completion Kind	Error Rate	Completion Example
Identifier	38%	`contains = jQuery …`
Property	35%	`start = list.length;`
String	48%	`'[' + attrs + ']'`
Number	36%	`canvas(xy[0], xy[1], …)`
RegExp	34%	`line.replace(/(	)+/, …)`
UnaryExpr	3%	`if (!events || !…)`
BinaryExpr	26%	`while (++index ≤ …)`
LogicalExpr	8%	`frame = frame || …`

ICML @ NYC
JUNE 19-24 2016