PHOG: Probabilistic Model for Code

Pavol Bielik, Veselin Raychev, Martin Vechev
Software Reliability Lab
Department of Computer Science
ETH Zurich
Vision

- 15 million repositories
- Billions of lines of code
- High quality, tested, maintained programs

number of repositories

Statistical Programming Tool

Probabilistic Model

15 million repositories

Billions of lines of code

High quality, tested, maintained programs

last 8 years
Understand code/security [POPL’15]:
JavaScript Deobfuscation
Type Prediction

Write new code [PLDI’14]:
Code Completion

Port code [ONWARD’14]:
Programming Language Translation

Debug code:
Statistical Bug Detection

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

All of these benefit from the probabilistic model for code.

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

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

www.jsnice.org
Understand code [POPL’15]:
JavaScript Deobfuscation
Type Prediction

for x in range(a):
    print a[x]

Debug code:
Statistical Bug Detection
Statistical Programming Tools

Write new code [PLDI’14]:
Code Completion

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

Port code [ONWARD’14]:
Programming Language Translation

Programming Languages
+
Machine Learning

www.jsnice.org

All of these benefit from the probabilistic model for code.
Model Requirements

Existing Programs

Learning

Model

- Widely Applicable
- Efficient Learning
- High Precision
- Explainable Predictions

Probabilistic Model
Model Requirements

Existing Programs

Learning

Model

Probabilistic Model

PHOG: Probabilistic Higher Order Grammar

- Widely Applicable
- Efficient Learning
- High Precision
- Explainable Predictions
Example Query

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

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

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

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

Long distance dependencies

---

**Correct prediction**

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>promise</td>
<td>0.67</td>
</tr>
<tr>
<td>notify</td>
<td>0.12</td>
</tr>
<tr>
<td>resolve</td>
<td>0.11</td>
</tr>
<tr>
<td>reject</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Challenges

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

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

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

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

**Syntactic**

[Hindle et al., 2012]
[Allamanis et al., 2015]

$\text{arg max } P( x | \text{features})$

$\text{label} \quad \text{conditioning context}$

Bad fit for programs
Existing Approaches for Code

**Syntactic**

[Nguyen et al., 2013]
[Allamanis et al., 2014]

(label) conditioning context

Bad fit for programs

**Semantic**

[Nguyen et al., 2013]
[Allamanis et al., 2014]

(templates) defer reject promise

**Hard-coded heuristics**

Task & Language specific
PHOG: Concepts

Program synthesis learns a function that explains the data. The function returns a conditioning context for a given query.

Use function to build a probabilistic model. Generalizes PCFGs to allow conditioning on richer context.
Generalizing PCFG

Context Free Grammar
$\alpha \rightarrow \beta_1 \ldots \beta_n$

$P$

Property $\rightarrow x$ 0.05
Property $\rightarrow y$ 0.03
Property $\rightarrow$ promise 0.001
PHOG: Generalizes PCFG

<table>
<thead>
<tr>
<th>Context Free Grammar</th>
<th>Higher Order Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha \rightarrow \beta_1 \ldots \beta_n$</td>
<td>$\alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>$p$</th>
<th>Property</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rightarrow x$</td>
<td>$0.05$</td>
<td>$\rightarrow \text{reject, promise}$</td>
<td>$0.67$</td>
</tr>
<tr>
<td>$\rightarrow y$</td>
<td>$0.03$</td>
<td>$\rightarrow \text{promise}$</td>
<td>$0.67$</td>
</tr>
<tr>
<td>$\rightarrow \text{promise}$</td>
<td>$0.001$</td>
<td>$\rightarrow \text{notify}$</td>
<td>$0.12$</td>
</tr>
<tr>
<td>$\rightarrow \text{resolve}$</td>
<td>$0.11$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conditioning on Richer Context

\[ \alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n \]

What is the best conditioning context?
Conditioning on Richer Context

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

What is the best conditioning context?

- APIs
- Identifiers
- Control Structures
- Fields
- Constants
- ...
Conditioning on Richer Context

\[ \alpha[\gamma] \rightarrow \beta_1...\beta_n \]

What is the best conditioning context?

- APIs
- Fields
- Identifiers
- Constants
- Control Structures
- ...

Source Code \[ \rightarrow ? \rightarrow \gamma \]

Conditioning Context
Higher Order Grammar

Production Rules R:
\[ \alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n \]

Function:
\[ f: \rightarrow \gamma \]

Parametrize the grammar by a function used to dynamically obtain the context
Higher Order Grammar

Production Rules R:
\[ \alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n \]

Function:
\[ f: \text{AST} \rightarrow \gamma \]

Parametrize the grammar by a function used to dynamically obtain the context
Higher Order Grammar

Production Rules $R$:

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

Function:

$$f: \text{AST} \rightarrow \gamma$$
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
Expressing functions: TCond Language

TCond ::=
\epsilon \mid WriteOp \ TCond \mid MoveOp \ TCond

MoveOp ::= Up, Left, Right, DownFirst, DownLast, NextDFS, PrevDFS, NextLeaf, PrevLeaf, PrevNodeType, PrevNodeValue, PrevNodeContext

WriteOp ::= WriteValue, WriteType, WritePos
elem.notify(
    ..., 
    ..., 
    {
        position: 'top',
        hide: false,
        ...
    }
);
elem.notify(
  ..., ,
  ..., ,
  {
    position: 'top',
    hide: false,
    ?
  }
);
Example

Query

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

TCond

Left
WriteValue
Up
WritePos

γ

{} 
{hide}  
{hide}  
{hide, 3}
Example

Query

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

TCond

```
Left
WriteValue
Up
WritePos
Up
DownFirst
DownLast
WriteValue
{hide, 3, notify}
```
elem.notify(
  ...
  ...
  {
    position: 'top',
    hide: false,
    ?
  }
);

Query

TCond

\[ \gamma \]

\begin{align*}
  \text{Left} & \quad \text{WriteValue} \\
  \text{Up} & \quad \text{WritePos} \\
  \text{Up} & \quad \text{DownFirst} \\
  \text{DownLast} & \quad \text{WriteValue} \\
\end{align*}

\{ Previous Property, Parameter Position, API name \}
Learning PHOG

\[
T\text{Cond} \ ::= \ \epsilon \mid \text{WriteOp } T\text{Cond} \mid \text{MoveOp } T\text{Cond}
\]

\[
\text{MoveOp} \ ::= \ \text{Up, Left, Right, } \ldots
\]

\[
\text{WriteOp} \ ::= \ \text{WriteValue, WriteType, } \ldots
\]

Existing Dataset

Program Synthesis
Enumerative search
Genetic programming

\[
f_{\text{best}} = \arg \min_{f \in T\text{Cond}} \text{cost}(D, f)
\]

Representative sampling

Learning Programs from Noisy Data.

POPL ’16, ACM.
Evaluation

Probabilistic Model of JavaScript Language

20k TCond learning  100k PHOG training  50k Blind Set

GitHub
## Evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Code Completion Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFG</td>
<td>49.9%</td>
</tr>
<tr>
<td>n-gram</td>
<td>28.7%</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>45.8%</td>
</tr>
<tr>
<td>SVM</td>
<td>29.5%</td>
</tr>
<tr>
<td>PHOG</td>
<td>18.5%</td>
</tr>
<tr>
<td>Identifier</td>
<td>Error Rate</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Identifier</td>
<td>38%</td>
</tr>
<tr>
<td>Property</td>
<td>35%</td>
</tr>
<tr>
<td>String</td>
<td>48%</td>
</tr>
<tr>
<td>Number</td>
<td>36%</td>
</tr>
<tr>
<td>RegExp</td>
<td>34%</td>
</tr>
<tr>
<td>UnaryExpr</td>
<td>3%</td>
</tr>
<tr>
<td>BinaryExpr</td>
<td>26%</td>
</tr>
<tr>
<td>LogicalExpr</td>
<td>8%</td>
</tr>
</tbody>
</table>
## Evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Queries per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFG</td>
<td>1 min</td>
<td>71 000</td>
</tr>
<tr>
<td>n-gram</td>
<td>4 min</td>
<td>15 000</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>3 min</td>
<td>10 000</td>
</tr>
<tr>
<td>SVM</td>
<td>36 hours</td>
<td>12 500</td>
</tr>
<tr>
<td>PHOG</td>
<td>162 + 3 min</td>
<td>50 000</td>
</tr>
</tbody>
</table>
Key Ideas:

- Learn a function that explains the data. The function dynamically obtains the best conditioning context for a given query.

\[ f_{\text{best}} = \arg \min_{f \in \text{TCond}} \text{cost}(D, f) \]

- Define a new generative model that is parametrized by such learned function.

PHOG(\( f_{\text{best}} \))