Statistical Synthesis from “Big Code”
So far: synthesis from examples

Input:

program with holes
requirements: program inputs/outputs

Output:
synthesized program

Already seen in project. Often encoded to a complex solver. No control of which solution to return if multiple solutions exist.
“Big Code”

A new still untapped corpus of data.

Plenty of well-maintained code repositories

Using the data still requires program analysis
Non-statistical synthesis

Problems:

Dealing with multiple solutions. Ranking.

Solver may be too slow

Works only with very few unknowns (holes).
“Big Code” @ ETH Zurich

Statistical Code Completion, PLDI’14 (V. Raychev, M. Vechev, E. Yahav)

Intent i = new Intent();
ctx.sendBroadcast(i);

Statistical PL Translation, Onward’14 (S. Karaivanov, V. Raychev, M. Vechev)

P( Java | C# )
P( C# | Java )
P( Java )

Predicting Program Properties from “Big Code”, POPL’15 (V. Raychev, A. Krause, M. Vechev)

Slang

Statistical API code completion system

Motivated by large number of shared APIs used by many projects. Developed for Java/Android
Slang: Capabilities

```java
Camera camera = Camera.open();
camera.setDisplayOrientation(90);

MediaRecorder rec = new MediaRecorder();

rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);

rec.setOutputFile("file.mp4");
...```
Slang: Capabilities

```java
Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();

MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");

...
Key insight

Regularities in code are similar to regularities in natural languages

We want to learn that

```java
MediaRecorder rec = new MediaRecorder();
```

is before

```java
rec.setCamera(camera);
```

like in natural languages

```plaintext
Hello
```

is before

```plaintext
World!
```
The Slang system

Completion phase

Program Analysis → Query → Combine

sentences with holes → completed sentences

Training phase

Program Analysis → Train Language Model

sentences → Language model

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();

MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();

MediaRecorder rec = new MediaRecorder();
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

github

Atlassian Bitbucket
From programs to sentences

First: Flow-insensitive **alias** analysis

```java
void method(MediaRecorder r1, MediaRecorder r2) {
    MediaRecorder r3;
    r3 = r1;
    Camera camera = Camera.open();
    camera.setDisplayOrientation(90);
}
```

**Abstract objects**

<table>
<thead>
<tr>
<th>r1, r3</th>
<th>r2</th>
<th>camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaRecorder</td>
<td>MediaRecorder</td>
<td>Camera</td>
</tr>
</tbody>
</table>
From programs to sentences

```
she = new X();
me = new Y();
me.sleep();
if (random()) {
    me.eat();
}
she.enter();
me.talk(she);
```

Second: combine alias and typestate Analysis

Abstract object `me`:
- \( Y_{\text{init}} \) sleep talk
- \( Y_{\text{init}} \) sleep eat talk

Abstract object `she`:
- \( X_{\text{init}} \) enter talk\( _{\text{param1}} \)
Under-/over-approximation

Both under- and over-approximation

Over:
   analyze methods individually
   ... 

Under:
   assume function parameters point to different objects
   inline loops to only a fixed number of iterations 
   ...
Learning

Most important:

- Code is converted to sentences
- Regularities in sentences $\Leftrightarrow$ regularities in API usage

e.g. if we see many sequences like to:

$$X_{\text{init}} \text{ enter} \text{ talk}_{\text{param1}}$$

learn that $\text{talk}_{\text{param1}}$ is often after $\text{enter}$
Given a sentence $s = w_1 w_2 w_3 \ldots w_n$

estimate $P( w_1 w_2 w_3 \ldots w_n )$

Decomposed to conditional probabilities

$P( w_1 w_2 w_3 \ldots w_n ) = \prod_{i=1..n} P( w_i | w_1 \ldots w_{i-1} )$

$P( \text{The quick brown fox jumped} ) =$

$P( \text{The} ) \cdot P( \text{quick} | \text{The} ) \cdot P( \text{brown} | \text{The quick} )$

$P( \text{fox} | \text{The quick brown} ) \cdot P( \text{jumped} | \text{The quick brown fox} )$
**N-gram language models**

Conditional probability only on previous \( n-1 \) words

\[
P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})
\]

There are standard techniques for handling \( n \)-grams with 0 or few occurrences in the training data (not discussed here)

- **smoothing**, **discounting**
Another type of language model. Learns dependencies beyond the prior several words.

A neural network with internal state that stores probabilistic information about all previous words in a sentence.

With this, it can capture relationship between quick and jumped in:

\[ P(\text{jumped} \mid \text{The quick brown fox}) \]
The Slang System

Training phase

Program Analysis

Train Language Model

Sentences

Completion phase

Program Analysis

Query

Combine

Sentences with holes

Completed sentences

We are here

Language model

Incomplete program

camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

Completed program

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

Combine

Query

Program Analysis

Sentences with holes

Completed sentences

Language model

Incomplete program

camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

Training phase

Program Analysis

Train Language Model

Sentences
Program analysis with holes

```java
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
  list = smsMgr.divideMessage(message);
  ? {smsMgr, list} // (Hole H1)
} else {
  ? {smsMgr, message} // (Hole H2)
}
```

Abstract object `smsMgr`:
- `getDefault` result `divideMessage` H1
- `getDefault` result H2

Abstract object `list`:
- `divideMessage` result H1

Abstract object `message`:
- `length` `divideMessage` param1
- `length` H2
## Sentences with holes

<table>
<thead>
<tr>
<th>getDefaul\textsubscript{result}</th>
<th>divideMessage</th>
<th>H1</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDefaul\textsubscript{result}</td>
<td>H2</td>
<td></td>
</tr>
<tr>
<td>divideMessage\textsubscript{result}</td>
<td>H1</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td>H2</td>
<td></td>
</tr>
<tr>
<td>Method Combination</td>
<td>Completion Probability</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td><code>getDefault</code> <code>divideMessage</code> <code>sendMultipartTextMessage</code></td>
<td>0.0033</td>
<td></td>
</tr>
<tr>
<td><code>getDefault</code> <code>divideMessage</code> <code>sendTextMessage</code></td>
<td>0.0016</td>
<td></td>
</tr>
<tr>
<td><code>getDefault</code> <code>sendTextMessage</code></td>
<td>0.0073</td>
<td></td>
</tr>
<tr>
<td><code>getDefault</code> <code>sendMultipartTextMessage</code></td>
<td>0.0010</td>
<td></td>
</tr>
<tr>
<td><code>divideMessage</code> <code>sendMultipartTextMessage</code></td>
<td>0.0821</td>
<td></td>
</tr>
<tr>
<td><code>length</code> <code>length</code></td>
<td>0.0132</td>
<td></td>
</tr>
<tr>
<td><code>length</code> <code>split</code></td>
<td>0.0080</td>
<td></td>
</tr>
<tr>
<td><code>length</code> <code>sendTextMessage</code> <code>param3</code></td>
<td>0.0017</td>
<td></td>
</tr>
<tr>
<td>Completion probability</td>
<td>sentence completions</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>0.0033</td>
<td><code>getDefaultr</code> <code>divideMessage</code> <code>sendMultipartTextMessage</code></td>
<td></td>
</tr>
<tr>
<td>0.0016</td>
<td><code>getDefaultr</code> <code>divideMessage</code> <code>sendTextMessage</code></td>
<td></td>
</tr>
<tr>
<td>0.0073</td>
<td><code>getDefaultr</code> <code>sendTextMessage</code></td>
<td></td>
</tr>
<tr>
<td>0.0010</td>
<td><code>divideMessage</code> <code>sendMultipartTextMessage</code> <code>param3</code></td>
<td></td>
</tr>
<tr>
<td>0.0132</td>
<td><code>length</code> <code>length</code></td>
<td></td>
</tr>
<tr>
<td>0.0080</td>
<td><code>length</code> <code>split</code></td>
<td></td>
</tr>
<tr>
<td>0.0017</td>
<td><code>length</code> <code>sendTextMessage</code> <code>param3</code></td>
<td></td>
</tr>
</tbody>
</table>

**Not a feasible solution:**
completions disagree on selected method

**The solution must satisfy program constraints**
<table>
<thead>
<tr>
<th>Completion probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0033</td>
</tr>
<tr>
<td>0.0016</td>
</tr>
<tr>
<td>0.0073</td>
</tr>
<tr>
<td>0.0010</td>
</tr>
<tr>
<td>0.0821</td>
</tr>
<tr>
<td>0.0132</td>
</tr>
<tr>
<td>0.0080</td>
</tr>
<tr>
<td>0.0017</td>
</tr>
</tbody>
</table>
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    list = smsMgr.divideMessage(message);
    smsMgr.sendMultipartTextMessage(...list...);
} else {
    smsMgr.sendTextMessage(...message...);
}
The Slang System

Completion phase

Program Analysis

sentences with holes

Query

completed sentences

Combine

~100MB

Training phase

sentences

Language model

Program Analysis

Train Language Model

~700MB

84 testing samples

The correct completion in top 3 for ~90% of cases

1M Java methods

~700MB

The Slang System

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();

MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");

...
Results: completion quality

More data - better accuracy

Better program analysis: learn more from the data
Probabilistic Program Analysis
“Big Code” @ ETH Zurich

Statistical Code Completion, PLDI’14
(V. Raychev, M. Vechev, E. Yahav)

Intent i = new Intent();
    ?
ctx.sendBroadcast(i);

Statistical PL Translation, Onward’14
(S. Karaivanov, V. Raychev, M. Vechev)

Predicting Program Properties from “Big Code”, POPL’15
(V. Raychev, A. Krause, M. Vechev)

Example predictions

What is this doing?

```javascript
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;
}
```

Types hard to predict:
Google Closure Compiler, TypeScript, Facebook Flow all infer `{?}`

```javascript
/**
 * @param {string} str
 * @param {number} step
 * @return {?
 */
function chunkData(str, step)
{
    /* @type {Array} */
    var colNames = [];
    var len = str.length;
    /* @type {number} */
    var i = 0;
    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;
}
```

easy to infer
predict variable names

www.JSNice.org

Ingvar Stepanyan @RReverser · Aug 6
JSNice.org became my must-have tool for code deobfuscation.
Predicting names

What can be “soundly” renamed in JavaScript?

```javascript
function f(a) {
    var b = document.getElementById(a);
    return b;
}
```

unknown facts:  

known facts:  

- a
- b
- f
- document
- getElementById
Goal: predict **unknown** facts given these **known** facts
Predicting type annotations

Type annotations for Google Closure Compiler

(available training data)

```javascript
function inc(a) {
    return a + 1;
}
```

Standard type inference returns:

unknown facts: $a: \top$

to be predicted

known facts: 1: `number`

**Key point:** All facts are related (for both, names and types)
Challenges

- Predicted facts are dependent
- Must learn from huge codebases
- Prediction should be fast (real time)
General prediction approach

We phrased the general problem of predicting program facts into a framework based on:

- **Model:** Conditional Random Fields (CRFs)
- **Query:** MAP inference
- **Learning:** Structured SVM
Example: choose the right words

I go to a lecture in Varna by bus.

constraints:

1) It is May 21 ⇒ lecture
2) Varna has no trams
Example: choose the right words

I go to a \textcolor{green}{\text{lecture}} in \textcolor{yellow}{\text{Varna}} by \textcolor{yellow}{\text{bus}}

constraints:

1) It is May 21 $\Rightarrow$ lecture
2) Varna has no trams
A possible choice (the most likely):

I go to a **lecture** in **Zurich** by **tram**

Constraints:

1) It is May 21 ⇒ lecture
2) Varna has no trams
Conditional Random Fields
(J. Lafferty, A. McCallum, F. Pereira, ICML 2001)

- Undirected graphical model
- Captures **dependencies** between facts to be predicted
- Captures **conditional distribution** on known facts

\[ \mathbf{P}(c,t|v) = \varphi_1(v,c) \times \varphi_2(c,t) / Z(v) \]

\(v\) - known fact, \(c\) and \(t\) - random variables (unknown facts)
\(\varphi_1, \varphi_2\) - scoring functions

makes \(\mathbf{P}\) a valid probability distribution
very expensive to compute
I go to a lecture in Varna by bus tram.

Key Point: CRFs model conditional probabilities a.k.a a discriminative model.
CRF notation

CRF:

\[ P(y|x) = \frac{1}{Z} \prod \varphi_i(x,y) \]  

(factor graph)

known program facts: \( x \)   
unknown: \( y \)

Key Point: CRFs model conditional probabilities  
a.k.a discriminative model
General prediction approach

- **Model**: Conditional Random Fields (CRFs)
- **Query**: MAP inference
- **Learning**: Structured SVM
Our goal is to find the most likely assignment of \( y \) that satisfies the constraints, also known as **MAP inference**:

\[
y = \arg\max_y \ P(y'|x) = \arg\max_y \ 1/Z \ \prod \ \varphi_i(x,y)
\]

**Good news:** for this query the expensive partition function \( Z(x) \) is unnecessary
Our goal is to find the most likely assignment of $y$ that satisfies the constraints, also known as MAP inference:

$$y = \arg\max_y P(y'|x) = \arg\max_y \Pi \varphi_i(x,y)$$

**Good news:** for this query the expensive partition function $Z(x)$ is unnecessary

**Bad news:** computing the argmax is still NP-hard (Max-SAT)

**Our solution:** approximate algorithms
**MAP inference example**

\[
\text{argmax}_{c, t} \mathbf{P}(c, t|v=\text{talk})
\]

<table>
<thead>
<tr>
<th>(v)</th>
<th>(c)</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>lecture</td>
<td>Varna</td>
<td>2</td>
</tr>
<tr>
<td>lecture</td>
<td>Zurich</td>
<td>10</td>
</tr>
</tbody>
</table>

\[\varphi_1\]

<table>
<thead>
<tr>
<th>(c)</th>
<th>(t)</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>_</td>
<td>bus</td>
<td>5</td>
</tr>
<tr>
<td>Zurich</td>
<td>tram</td>
<td>10</td>
</tr>
<tr>
<td>Varna</td>
<td>tram</td>
<td>0</td>
</tr>
</tbody>
</table>

\[\varphi_2\]
MAP inference example

$$\arg\max_{c,t} P(c,t|v=\text{talk})$$

Maximize product of scores:

$$2 \times 5 = 10$$
MAP inference example

$\arg\max_{c,t} P(c,t|v=\text{talk})$

Maximize product of scores:

$10 \times 5 = 50$
MAP inference example

$$\arg\max_{c,t} P(c,t|v=talk)$$

\[
\begin{array}{|c|c|c|}
\hline
v & c & \text{Score} \\
\hline
\text{lecture} & \text{Varna} & 2 \\
\text{lecture} & \text{Zurich} & 10 \\
\hline
\end{array}
\]

Maximize product of scores:

$$10 \times 10 = 100$$

\[
\begin{array}{|c|c|c|}
\hline
\text{c} & \text{t} & \text{Score} \\
\hline
\_ & \text{bus} & 5 \\
\text{Zurich} & \text{tram} & 10 \\
\text{Varna} & \text{tram} & 0 \\
\hline
\end{array}
\]

Guaranteed to satisfy constraints

No partition function
General prediction approach

- **Model:** Conditional Random Fields (CRFs)
- **Query:** MAP inference
- **Learning:** Structured SVM
Learning

CRF - two equivalent notations

\[ P(y|x) = \frac{1}{Z} \prod \varphi_i(x,y) \]

\[ P(y|x) = \frac{1}{Z} \exp \sum \lambda_i f_i(x,y) \]

<table>
<thead>
<tr>
<th>( \varphi_1 )</th>
<th>v</th>
<th>c</th>
<th>Score</th>
<th>( f_1 = 1 ) if v=lecture ( \land ) c=Varna,</th>
<th>0 otherwise</th>
<th>( \lambda_1 = \log 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>lecture</td>
<td>Varna</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lecture</td>
<td>Zurich</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \varphi_2 )</th>
<th>v</th>
<th>c</th>
<th>Score</th>
<th>( f_2 = 1 ) if v=lecture ( \land ) c=Zurich,</th>
<th>0 otherwise</th>
<th>( \lambda_2 = \log 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>lecture</td>
<td>Varna</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lecture</td>
<td>Zurich</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Learning finds weights $\lambda_i$ from training data

$$P(y|x) = \frac{1}{Z} \exp \sum \lambda_i f_i(x, y)$$

$D = \{ x^{(j)}, y^{(j)} \}_{j=1..n}$

programs with facts of interest already manually annotated

Big codebase to learn from
Programmers have spent countless hours to develop, maintain and annotate
Structured SVM

Generalizes SVM, learns weights such that:

\[ \forall j \, \forall y \sum \lambda_i f_i(x^{(j)}, y^{(j)}) \geq \sum \lambda_i f_i(x^{(j)}, y) + \Delta(y, y^{(j)}) \]

for all training data samples

the given prediction is better than any other prediction by at least a margin

Training procedure:

N. Ratliff, J. Bagnell, M. Zinkevich: (Online) Subgradient Methods for Structured Prediction, AIStats'07

- Stochastic (sub-) gradient descent
- Uses MAP inference as a subroutine
- no partition function Z
General approach: summary

Prediction phase

Training phase

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

/\* @param {string} str
 * @param {number} step
 * @return {Array}
/\*
function chunkData(str, step) {
    var colNames = [];
    var len = str.length;
    var i = 0;
    for (; i < len; i += step) {
        colNames.push(str.substring(i, i + step));
    } else {
        colNames.push(str.substring(i, len));
    }
    return colNames;
}
Predicting names and types are two **instantiations** of the **general approach** discussed before.

Remaining:
We need to build CRF graphs from programs.

```javascript
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;
}
```
Features

Edges in CRF graph define features

We put edges depending on AST relationships of elements

- some typing rules are subset of these features

AST:

```
a = b
```

```
a b
```

```
a feature b
```
Training data for names and types

10'517 JavaScript projects from GitHub

Simple detector for minified files removes code with no good names and types

325'501 JavaScript files
Learned 7'627'484 features for names and 70'052 features for types

Finally, we took 50 different JavaScript projects to evaluate on (from bitbucket)
## Evaluation results

<table>
<thead>
<tr>
<th></th>
<th>Names Accuracy</th>
<th>Types Precision</th>
<th>Types Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>25.3%</td>
<td>37.8%</td>
<td>-</td>
</tr>
<tr>
<td>Independent predictions</td>
<td>54.1%</td>
<td>84.0%</td>
<td>56.0%</td>
</tr>
<tr>
<td>JSNice</td>
<td>63.4%</td>
<td>81.6%</td>
<td>66.9%</td>
</tr>
</tbody>
</table>

**Structured Prediction is Critical**

**Types:** more programs typecheck with predicted types than with manually provided types
http://jsnice.org/

http://nice2predict.org/

http://www.srl.inf.ethz.ch/