Static Analysis and Reflection

Manu Sridharan
(and many collaborators)
IBM Research
What is Reflection?

class Factory {
    Object make(String x) {
        return Class.forName(x).newInstance();
    }
}

What is Reflection?

```java
class Factory {
    Object make(String x) {
        return Class.forName(x).newInstance();
    }
}
```

- Operating on code entities via strings
  - Allocation, invoking methods, accessing fields
What is Reflection?

```java
class Factory {
    Object make(String x) {
        return Class.forName(x).newInstance();
    }
}
```

- Operating on code entities via strings
  - Allocation, invoking methods, accessing fields
- Why?
  - Control via configuration files (frameworks)
  - Meta-programming (generic toString)
  - No good reason (some uses of JavaScript eval)
### Reflection Dilemma

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
<th>Ignore</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reflection Dilemma

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
<th>Ignore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>
## Reflection Dilemma

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
<th>Ignore</th>
<th>(key behaviors?)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>✘</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>✓</td>
<td>✘</td>
<td></td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>✘</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
**Reflection Dilemma**

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
<th>Ignore</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precision</strong></td>
<td>![X]</td>
<td>![√]</td>
</tr>
<tr>
<td><strong>Recall</strong></td>
<td>![√]</td>
<td>![X] (key behaviors?)</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>![X]</td>
<td>![√]</td>
</tr>
</tbody>
</table>

**Challenge:** Strike right balance for client
Our Reflection Story

- **Clients**
  - Java taint analysis
  - JavaScript taint analysis (call graphs)
  - JavaScript IDE tools

- **Approaches**
  - **Static**: model via code analysis
  - **Specifications**: use additional artifacts
  - **Dynamic**: observe behavior, record or generalize
Java Taint Analysis
Reflection Handling in TAJ
Reflection Handling in TAJ

- Taint Analysis for Java
  - Pointer analysis for call graphs + aliasing
  - Soundness not required
  - But, ignoring reflection is too unsound
Reflection Handling in TAJ

- Taint Analysis for Java
  - Pointer analysis for call graphs + aliasing
  - Soundness not required
  - But, ignoring reflection is too unsound
- Enhanced pointer analysis with reflection handling
  - Track string constants, Class / Method objects
  - Generate synthetic IR for reflective operations
    - For `c.newInstance()`, if `c` is `Class<Foo>`, model as `new Foo()`
  - As in Livshits et al., APLAS’05
A Vicious Cycle...

Imprecise value flow

Polluted reflection handling
A Vicious Cycle...

- Imprecise value flow
- Polluted reflection handling

Quadratic blowup
- Huge analysis time / memory
- Highly imprecise result
“Fixing” the problem

- Tried bounding pointer analysis, but fragile
- In the end, dumped pointer analysis
  - Instead, heuristic type-based call graph
  - Track aliases during taint analysis
  - See Tripp et al., FASE’13
- Hand-tuned reflection handling for frameworks
  - But many frameworks in practice...
  - Nasty reflection based on config files
A Framework for Frameworks (OOPSLA’11)
A Framework for Frameworks (OOPSLA’11)
A Framework for Frameworks (OOPSLA’11)

- Spring Handler
- ASP.NET Handler
- Framework handlers

config. info ➔ Spring Handler ➔ Taint analysis ➔ vulnerabilities

Application ➔ Framework handlers ➔ Taint analysis ➔ vulnerabilities
A Framework for Frameworks (OOPSLA’11)

- Spring Handler
- ASP.NET Handler
- Taint analysis
- Vulnerabilities

config. info

Web Application Framework Language (WAFL) specification

application
A Framework for Frameworks (OOPSLA’11)

- Easy to support new frameworks

Diagram:
- config. info
  - Spring Handler
  - ASP.NET Handler
  - Framework handlers
  - Web Application Framework Language (WAFL) specification
- application
  - Taint analysis
  - vulnerabilities
A Framework for Frameworks (OOPSLA’11)

Easy to support new frameworks

Expressive, yet easy to integrate

Config. info → Framework handlers

Spring Handler → ASP.NET Handler → ...

Web Application Framework Language (WAFL) specification

Application → Taint analysis → Vulnerabilities
A Framework for Frameworks (OOPSLA’11)

Key to precision: Easy to support new frameworks

Config. info → Framework handlers

Spring Handler → ASP.NET Handler → ...

Application → Taint analysis → Vulnerabilities

Web Application Framework Language (WAFL) specification

Expressive, yet easy to integrate
Example: edit profile
Example: edit profile

<table>
<thead>
<tr>
<th>Code</th>
<th>Configuration</th>
</tr>
</thead>
</table>

Example: edit profile

Code

```java
// for user data
class UserForm {
    String firstName, lastName;
    // getters and setters...
}

// updates profile
class UserAction implements IAction {
    String exec(HttpRequest req, Object form) {
        UserForm uf = (UserForm) form;
        updateDB(uf);
        ...
    }
}
```

Configuration
Example: edit profile

**Code**

```java
// for user data
class UserForm {
    String firstName, lastName;
    // getters and setters...
}

// updates profile
class UserAction implements IAction {
    String exec(HttpRequest req, Object form) {
        UserForm uf = (UserForm) form;
        updateDB(uf);
        ...
    }
}
```

**Configuration**

```
<action url="/edit"
    type="UserAction"
    formtype="UserForm">
</action>
```

*In English*: When “/edit” is visited, create a UserForm object (reflection), set its fields using request data (reflection), and pass it to UserAction.exec() (reflection).
WAFL: synthetic methods

```java
fun entrypoint UserAction_entry(request) {
    UserForm f = new UserForm();
    f.setFirstName(request.getParam("firstName"));
    f.setLastName(request.getParam("lastName"));
    (new UserAction()).exec(request, f);
}
```
WAFL: synthetic methods

```java
fun entrypoint UserAction_entry(request) {
    UserForm f = new UserForm();
    f.setFirstName(request.getParam("firstName"));
    f.setLastName(request.getParam("lastName"));
    (new UserAction()).exec(request, f);
}
```

- **Simple structure**: no branches, loops, etc.
- Eases integration with analysis engine
- Taint analysis usually flow insensitive anyway
- Based on both app code and config info
What about dynamic?

- The Tamiflex approach (Bodden et al., ICSE’11)
  - Log runtime reflective operations
  - Use log to transform code
  - Ensures soundness for tested inputs
- Difficulties
  - Running server code can be hard!
  - Need inputs to cover behaviors
Can we do better?

- F4F quite successful
- But, requires writing framework handlers
- Can we further automate?
- Maybe generalize from dynamic I/O?
- **Important problem**
JavaScript Taint Analysis
(or, Getting Tamed by jQuery)
var x = {};

// initialize object properties
x.foo = function f1() { return 23; }
x.bar = function f2() { return 42; }
x.foo(); // invokes f1
Pointer Analysis Needed

```javascript
var x = {};  
// initialize object properties
x.foo = function f1() { return 23; }  
x.bar = function f2() { return 42; }  
x.foo();  // invokes f1
```

- No declared types; objects can gain or lose fields
var x = {};
// initialize object properties
x.foo = function f1() { return 23; }
x.bar = function f2() { return 42; }
x.foo(); // invokes f1

- No declared types; objects can gain or lose fields
- **Pointer analysis needed** for call graphs
- Most method calls are “virtual”
- Cannot narrow call targets via types / arity
var f = p() ? "foo" : "baz";
// writes to o.foo or o.baz
o[f] = "Hello!";
Dynamic Property Accesses

```javascript
var f = p() ? "foo" : "baz";
// writes to o.foo or o.baz
o[f] = "Hello!";
```

- Used frequently inside frameworks
Dynamic Property Accesses

var f = p() ? "foo" : "baz";
// writes to o.foo or o.baz
o[f] = "Hello!";

- Used frequently inside frameworks
- Increases worst-case analysis complexity!
Dynamic Property Accesses

```javascript
var f = p() ? "foo" : "baz";
// writes to o.foo or o.baz
o[f] = "Hello!";
```

- Used frequently inside frameworks
- Increases worst-case analysis complexity!
- Leads to significant blowup in practice
function extend(dest, src) {
    for (var prop in src) {
        // correlated accesses
        dest[prop] = src[prop];
    }
}
Correlated Accesses

function extend(dest, src) {
  for (var prop in src) {
    // correlated accesses
    dest[prop] = src[prop];
  }
}

- Correlated: prop has same value at both accesses
Correlated Accesses

```javascript
function extend(dest, src) {
  for (var prop in src) {
    dest[prop] = src[prop];
  }
}
```

- **Correlated**: `prop` has the same value at both accesses.
- **Standard points-to analysis misses correlation**
  - Analysis merges all properties of `src`
  - For frameworks, leads to “quadratic blowup”
Function Extraction + Context Sensitivity

```javascript
function extend(dest, src) {
  for (var prop in src)
    dest[prop] = src[prop];
}
```
Function Extraction + Context Sensitivity

```javascript
function extend(dest, src) {
    for (var prop in src)
        dest[prop] = src[prop];
}

function extend(dest, src) {
    for (var prop in src)
        // extract accesses into
        // fresh function
        (function ext(p) {
            dest[p] = src[p];
        })(prop);
}
```
Function Extraction + Context Sensitivity

```javascript
function extend(dest, src) {
    for (var prop in src) {
        // extract accesses into fresh function
        // src[prop] = src[prop];
        dest[prop] = src[prop];
    }
}
```

```javascript
function extend(dest, src) {
    for (var prop in src) {
        // extract accesses into fresh function
        (function ext(p) {
            dest[p] = src[p];
        })(prop);
    }
}
```

ext contexts: p == “foo”, p == “baz”, ...
Function Extraction + Context Sensitivity

```javascript
function extend(dest, src) {
    for (var prop in src) {
        // extract accesses into fresh function
        // dest[prop] = src[prop];
        (function ext(p) {
            dest[p] = src[p];
        })(prop);
    }
}
```

- Analyze new functions with clone per property name
- Similar to object sensitivity / CPA
- Details in ECOOP’12

`ext contexts: p == “foo”, p == “baz”, …`
Results: Scalability

<table>
<thead>
<tr>
<th>Framework</th>
<th>Baseline&lt;sup&gt;−&lt;/sup&gt;</th>
<th>Baseline&lt;sup&gt;+&lt;/sup&gt;</th>
<th>Correlations&lt;sup&gt;−&lt;/sup&gt;</th>
<th>Correlations&lt;sup&gt;+&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>dojo</td>
<td>* (*)</td>
<td>*</td>
<td>3.1 (30.4)</td>
<td>6.7 (*)</td>
</tr>
<tr>
<td>jquery</td>
<td>*</td>
<td>*</td>
<td>78.5</td>
<td>*</td>
</tr>
<tr>
<td>mootools</td>
<td>0.7</td>
<td>*</td>
<td>3.1</td>
<td>*</td>
</tr>
<tr>
<td>prototype.js</td>
<td>*</td>
<td>*</td>
<td>4.4</td>
<td>4.5</td>
</tr>
<tr>
<td>yui</td>
<td>*</td>
<td>*</td>
<td>2.2</td>
<td>2.1</td>
</tr>
</tbody>
</table>

- Dramatic improvements with Correlations<sup>−</sup>
- Useful for an under-approximate call graph
- For ‘+’ configs, issues remain with call / apply
var e = "blur, focus, load".split(",");

for(var i=0; i<e.length; i++) {
    var o = e[i];
    jQuery.fn[o] = function() { ... };
    jQuery.fn["un"+o] = function() { ... };
    jQuery.fn["one"+o] = function() { ... };
}

Uses of dynamic property writes are inessential, could be flattened out!
Unnecessary Reflection

```javascript
var e = "blur, focus, load".split("","");

for (var i=0; i<e.length; i++) {
    var o = e[i];
    jQuery.fn[o] = function() { ... };
    jQuery.fn["un"+o] = function() { ... };
    jQuery.fn["one"+o] = function() { ... };
}

jQuery.fn.blur = function() { ... };
jQuery.fn.unblur = function() { ... };
jQuery.fn.oneblur = function() { ... };
jQuery.fn.focus = function() { ... };
jQuery.fn.unfocus = function() { ... };
jQuery.fn.onefocus = function() { ... };
jQuery.fn.load = function() { ... };
jQuery.fn.unload = function() { ... };
jQuery.fn.oneload = function() { ... };
```
Dynamic Determinacy Analysis

- Much reflection can be rewritten away
  - E.g., `eval` of constant, jQuery initialization
Dynamic Determinacy Analysis

- Much reflection can be rewritten away
  - E.g., `eval` of constant, jQuery initialization
- Pure static detection hard (needs a call graph!)
Dynamic Determinacy Analysis

- Much reflection can be rewritten away
  - E.g., `eval` of constant, jQuery initialization
- Pure static detection hard (needs a call graph!)
- Idea: Prove fixed behavior based on dynamic analysis
  - Find expressions “untainted” by inputs
  - Similar to dynamic information flow
  - See PLDI’13 paper for details
Dynamic Determinacy Analysis

- Much reflection can be rewritten away
  - E.g., `eval` of constant, jQuery initialization
- Pure static detection hard (needs a call graph!)
- **Idea:** Prove fixed behavior based on *dynamic* analysis
  - Find expressions “untainted” by inputs
  - Similar to dynamic information flow
  - See PLDI’13 paper for details
- Analyzed jQuery! But...version 1.0
  - Challenge: non-deterministic event handlers
JavaScript IDE Tools
Challenges

‣ Developers demand rich IDE functionality
  ‣ Code navigation (jump to declaration)
  ‣ Smart completion
  ‣ Refactoring
  ▶ Hard to build these features for JavaScript
    ▶ Reflection, lack of types, etc.
  ▶ For IDE, must be fast
Idea: Under-Approximate
Idea: Under-Approximate

- *Deliberately ignore* “hard” features of JS
  - *i.e.*, ignore reflection
Idea: Under-Approximate

- Deliberately ignore “hard” features of JS
  - I.e., ignore reflection
- Ok to miss some behaviors in IDEs
  - Even Java refactorings ignore reflection
Idea: Under-Approximate

- Deliberately ignore “hard” features of JS
  - I.e., ignore reflection
- Ok to miss some behaviors in IDEs
  - Even Java refactorings ignore reflection
- Design analysis to scale well and capture most behaviors
Field-Based Call Graphs
Field-Based Call Graphs

```plaintext
function A(x) {
    x.f = foo;
}

function B(y) {
    y.f = bar;
}
```
Field-Based Call Graphs

```
function A(x) {
    x.f = foo;
}

function B(y) {
    y.f = bar;
}
```

Ignore dynamic accesses `x[p]`
Why Does This Work?

```javascript
function extend(dst, src) {
  for (var x in src) {
    dst[x] = src[x];
  }
}

foo.f = function() { /*...*/ };;
extend(bar, foo);
bar.f();
```
Why Does This Work?

1 function extend(dst, src) {
2     for (var x in src) {
3         dst[x] = src[x];
4     }
5 }

6 foo.f = function() { /*...*/ };

7 extend(bar, foo);

8 bar.f();
Why Does This Work?

function extend(dst, src) {
    for (var x in src) {
        dst[x] = src[x];
    }
}

foo.f = function() { /*...*/ };
extend(bar, foo);
bar.f();
Evaluation

- Compared with dynamic call graphs
- No other usable static technique
- Best-effort coverage
- Measured precision, recall, and runtime
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Framework</th>
<th>LOC</th>
<th>Precision</th>
<th>Recall</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3dmodel</td>
<td>none</td>
<td>4.9k</td>
<td>93%</td>
<td>100%</td>
<td>0.26s</td>
</tr>
<tr>
<td>pacman</td>
<td>none</td>
<td>3.5k</td>
<td>94%</td>
<td>100%</td>
<td>0.47s</td>
</tr>
<tr>
<td>pdfjs</td>
<td>none</td>
<td>31.7k</td>
<td>77%</td>
<td>99%</td>
<td>5.62s</td>
</tr>
<tr>
<td>coolclock</td>
<td>jQuery</td>
<td>6.9k</td>
<td>89%</td>
<td>98%</td>
<td>1.32s</td>
</tr>
<tr>
<td>fullcalendar</td>
<td>jQuery</td>
<td>12.3k</td>
<td>84%</td>
<td>93%</td>
<td>2.85s</td>
</tr>
<tr>
<td>htmledit</td>
<td>jQuery</td>
<td>3.6k</td>
<td>81%</td>
<td>84%</td>
<td>0.80s</td>
</tr>
<tr>
<td>markitup</td>
<td>jQuery</td>
<td>6.5k</td>
<td>82%</td>
<td>94%</td>
<td>1.28s</td>
</tr>
<tr>
<td>pong</td>
<td>jQuery</td>
<td>3.6k</td>
<td>78%</td>
<td>93%</td>
<td>0.83s</td>
</tr>
<tr>
<td>flotr</td>
<td>Prototype</td>
<td>4.9k</td>
<td>72%</td>
<td>83%</td>
<td>1.76s</td>
</tr>
<tr>
<td>beslimed</td>
<td>MooTools</td>
<td>4.8k</td>
<td>78%</td>
<td>84%</td>
<td>1.06s</td>
</tr>
</tbody>
</table>
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Framework</th>
<th>LOC</th>
<th>Precision</th>
<th>Recall</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3dmodel</td>
<td>none</td>
<td>4.9k</td>
<td>93%</td>
<td>100%</td>
<td>0.26s</td>
</tr>
<tr>
<td>pacman</td>
<td>none</td>
<td>3.5k</td>
<td>94%</td>
<td>100%</td>
<td>0.47s</td>
</tr>
<tr>
<td>pdfjs</td>
<td>none</td>
<td>31.7k</td>
<td>77%</td>
<td>99%</td>
<td>5.62s</td>
</tr>
<tr>
<td>coolclock</td>
<td>jQuery</td>
<td>6.9k</td>
<td>89%</td>
<td>98%</td>
<td>1.32s</td>
</tr>
<tr>
<td>fullcalendar</td>
<td>jQuery</td>
<td>12.3k</td>
<td>84%</td>
<td>93%</td>
<td>2.85s</td>
</tr>
<tr>
<td>htmledit</td>
<td>jQuery</td>
<td>3.6k</td>
<td>81%</td>
<td>84%</td>
<td>0.80s</td>
</tr>
<tr>
<td>markitup</td>
<td>jQuery</td>
<td>6.5k</td>
<td>82%</td>
<td>94%</td>
<td>1.28s</td>
</tr>
<tr>
<td>pong</td>
<td>jQuery</td>
<td>3.6k</td>
<td>78%</td>
<td>93%</td>
<td>0.83s</td>
</tr>
<tr>
<td>flotr</td>
<td>Prototype</td>
<td>4.9k</td>
<td>72%</td>
<td>83%</td>
<td>1.76s</td>
</tr>
<tr>
<td>beslimed</td>
<td>MooTools</td>
<td>4.8k</td>
<td>78%</td>
<td>84%</td>
<td>1.06s</td>
</tr>
</tbody>
</table>

*Seconds, not hours*
### Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Framework</th>
<th>LOC</th>
<th>Precision</th>
<th>Recall</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3dmodel</td>
<td>none</td>
<td>4.9k</td>
<td>93%</td>
<td>100%</td>
<td>0.26s</td>
</tr>
<tr>
<td>pacman</td>
<td>none</td>
<td>3.5k</td>
<td>94%</td>
<td>100%</td>
<td>0.47s</td>
</tr>
<tr>
<td>pdfjs</td>
<td>none</td>
<td>31.7k</td>
<td>77%</td>
<td>99%</td>
<td>5.62s</td>
</tr>
<tr>
<td>coolclock</td>
<td>jQuery</td>
<td>6.9k</td>
<td>89%</td>
<td>98%</td>
<td>1.32s</td>
</tr>
<tr>
<td>fullcalendar</td>
<td>jQuery</td>
<td>12.3k</td>
<td>84%</td>
<td>93%</td>
<td>2.85s</td>
</tr>
<tr>
<td>htmledit</td>
<td>jQuery</td>
<td>3.6k</td>
<td>81%</td>
<td>84%</td>
<td>0.80s</td>
</tr>
<tr>
<td>markitup</td>
<td>jQuery</td>
<td>6.5k</td>
<td>82%</td>
<td>94%</td>
<td>1.28s</td>
</tr>
<tr>
<td>pong</td>
<td>jQuery</td>
<td>3.6k</td>
<td>78%</td>
<td>93%</td>
<td>0.83s</td>
</tr>
<tr>
<td>flotr</td>
<td>Prototype</td>
<td>4.9k</td>
<td>72%</td>
<td>83%</td>
<td>1.76s</td>
</tr>
<tr>
<td>beslimed</td>
<td>MooTools</td>
<td>4.8k</td>
<td>78%</td>
<td>84%</td>
<td>1.06s</td>
</tr>
</tbody>
</table>

>70% Precision
# Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Framework</th>
<th>LOC</th>
<th>Precision</th>
<th>Recall</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3dmodel</td>
<td>none</td>
<td>4.9k</td>
<td>93%</td>
<td>100%</td>
<td>0.26s</td>
</tr>
<tr>
<td>pacman</td>
<td>none</td>
<td>3.5k</td>
<td>94%</td>
<td>100%</td>
<td>0.47s</td>
</tr>
<tr>
<td>pdfjs</td>
<td>none</td>
<td>31.7k</td>
<td>77%</td>
<td>99%</td>
<td>5.62s</td>
</tr>
<tr>
<td>coolclock</td>
<td>jQuery</td>
<td>6.9k</td>
<td>89%</td>
<td>98%</td>
<td>1.32s</td>
</tr>
<tr>
<td>fullcalendar</td>
<td>jQuery</td>
<td>12.3k</td>
<td>84%</td>
<td>93%</td>
<td>2.85s</td>
</tr>
<tr>
<td>htmlledit</td>
<td>jQuery</td>
<td>3.6k</td>
<td>81%</td>
<td>84%</td>
<td>0.80s</td>
</tr>
<tr>
<td>markitup</td>
<td>jQuery</td>
<td>6.5k</td>
<td>82%</td>
<td>94%</td>
<td>1.28s</td>
</tr>
<tr>
<td>pong</td>
<td>jQuery</td>
<td>3.6k</td>
<td>78%</td>
<td>93%</td>
<td>0.83s</td>
</tr>
<tr>
<td>flotr</td>
<td>Prototype</td>
<td>4.9k</td>
<td>72%</td>
<td>83%</td>
<td>1.76s</td>
</tr>
<tr>
<td>beslimed</td>
<td>MooTools</td>
<td>4.8k</td>
<td>78%</td>
<td>84%</td>
<td>1.06s</td>
</tr>
</tbody>
</table>

>80% Recall
Smart Completion

```javascript
var x = { f: 3, g: "Manu" };
var y = x.

<table>
<thead>
<tr>
<th>f</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>String</td>
</tr>
</tbody>
</table>

(Eclipse Orion: eclipse.org/orion)
Smart Completion

- Again, flow analysis required, hard to scale
  - Unlike call graph, needs flows of all objects

(Eclipse Orion: eclipse.org/orion)
Smart Completion

In the image, there's a JavaScript example:
```
var x = { f: 3, g: "Manu" };
var y = x.
```

- Again, flow analysis required, hard to scale
  - Unlike call graph, needs flows of all objects
- **Observation**: most nasty reflection *occurs in libraries*
Smart Completion

Again, flow analysis required, hard to scale
- Unlike call graph, needs flows of all objects

**Observation**: most nasty reflection *occurs in libraries*

**Approach**: dynamic API inference for libraries
- Run instrumented library on unit tests
- Record observed types and use for flow analysis

(Eclipse Orion: eclipse.org/orion)
Smart Completion

- Again, flow analysis required, hard to scale
  - Unlike call graph, needs flows of all objects
- **Observation**: most nasty reflection *occurs in libraries*
- **Approach**: dynamic API inference for libraries
  - Run instrumented library on unit tests
  - Record observed types and use for flow analysis
- Compared to hand-written models, less effort and more complete

(Eclipse Orion: eclipse.org/orion)
Conclusions
Lessons Learned
Lessons Learned

- Details of reflection handling matter!
- Can dominate more common analysis parameters (*-sensitivity)
Lessons Learned

- Details of reflection handling matter!
- Can dominate more common analysis parameters (*-sensitivity)
- Best solutions specialized to client
- Varying performance, soundness needs
Lessons Learned

- Details of reflection handling matter!
- Can dominate more common analysis parameters (*-sensitivity)
- Best solutions specialized to client
- Varying performance, soundness needs
- Quadratic blowup complicates debugging
- Delta debugging helps
Next Steps

- Better abstractions / guidance for clients
- Refine under-approximate approaches
  - With help from user?
- Better language constructs?
  - E.g., MorphJ (Huang and Smaragdakis, PLDI’08)
Thanks!