Correlation Tracking for Points-To Analysis of JavaScript

Manu Sridharan, Julian Dolby, Satish Chandra, Max Schäfer, and Frank Tip

22 de Mayo del 2015 - Grupo: Nahabedian, Postolski, Roldán

Introduction

- Why JavaScript?
  - It’s flexible object model in which object properties can be created and deleted at run-time.
  - Programmers may define a “method” on an object by assigning a function to one of its properties.

```
o.foo = function f1() { return 23; };
```

Some problems :(

- Object properties are dynamic.

```
f = p() ? "foo" : "baz";
o[f] = "Hello , world!";
```

- The presence of **eval** means that a (useful) static analysis for JavaScript cannot be sound.

- Language lacks classes and declared types for variables.
    - Type filters cannot be employed.

- Flexibility with function’s arguments.

Dynamic Property Accesses

```
function extend(destination , source) {
    for (var property in source)
        destination[property] = source[property];
    return destination;
}
```

Dynamic Property Accesses

```
function extend(destination , source) {
    for (var property in source)
        destination[property] = source[property];
    return destination;
}
```

- We should see:

Dynamic Property Accesses

```
function extend(destination , source) {
    for (var property in source)
        destination[property] = source[property];
    return destination;
}
```

- We should see:
    - The name of the property accessed by a dynamic property access expression is computed at run-time.

Dynamic Property Accesses

```
function extend(destination , source) {
    for (var property in source)
        destination[property] = source[property];
    return destination;
}
```

- We should see:
    - The name of the property accessed by a dynamic property access expression is computed at run-time.
    - A write to a property creates that property if it does not exist yet.

Dynamic Property Accesses

```
function extend(destination , source) {
    for (var property in source)
        destination[property] = source[property];
    return destination;
}
```

- We should see:
  - The name of the property accessed by a dynamic property access expression is computed at run-time.
  - A write to a property creates that property if it does not exist yet.
  - JavaScript program create objects with an ***unbounded*** number of properties,

Andersen’s points-to analysis

Statement	Constraint	
`i: x = new T()`	$\{o_i\} \subseteq pt(x)$	[New]
`x = y`	$pt(y) \subseteq pt(x)$	[Assign]
`x = y.f`	$\dfrac{o_i \in pt(y)}{pt(o_i.f) \subseteq pt(x)}$	[Load]
`x.f = y`	$\dfrac{o_i \in pt(x)}{pt(y) \subseteq pt(o_i.f)}$	[Store]

Field-Sensitive Points-To Analysis for JavaScript

- Differs from a standard Andersen-style supports tracking of property names

- *The points-to set of a program variable x as pt(x)*

Statement	Constraint	
`x = {}`^{i}	$\{o^i\} \subseteq pt(x)$	[ALLOC]
`v = "name"`	$\{\texttt{name}\} \subseteq pt(v)$	[STRCONST]
`x = y`	$pt(y) \subseteq pt(x)$	[ASSIGN]
`x[v] = y`	$\dfrac{o \in pt(x) \qquad \texttt{s} \in pt(v)}{pt(y) \subseteq pt(o.\texttt{s})}$	[STOREFIELD]
`y = x[v]`	$\dfrac{o \in pt(x) \qquad \texttt{s} \in pt(v)}{pt(o.\texttt{s}) \subseteq pt(y)}$	[LOADFIELD]
`v = x.nextProp()`	$\dfrac{o \in pt(x) \qquad o.\texttt{s}\ \text{exists}}{\{\texttt{s}\} \subseteq pt(v)}$	[PROPITER]

Complexity

- Andersen’s Points-to analysis: $O(N^3)$
  - Solving a dynamic transitive closure.

- Point-to analysis in Javascript: $O(N^4)$
  - Andersen’s analysis + finding dynamic fields for each statement

Imprecision Example (I)

1. The program creates o1 and o2

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
prop = (*) ? "ext" : "ins";
t = src[prop];
dest[prop] = t;
```

Imprecision Example (I)

1. The program creates o1 and o2
2. The program creates properties *ext* and *ins* in the object o1

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
prop = (*) ? "ext" : "ins";
t = src[prop];
dest[prop] = t;
```

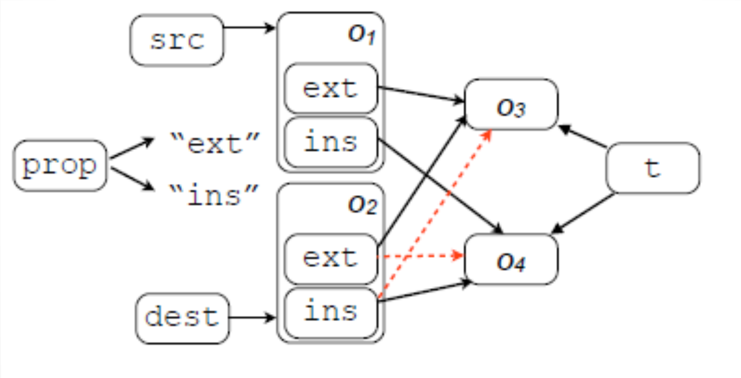
Imprecision Example (I)

1. The program creates o1 and o2
2. The program creates properties *ext* and *ins* in the object o1
3. Copies one of these properties to object o2

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
prop = (*) ? "ext" : "ins";
t = src[prop];
dest[prop] = t;
```

Imprecision Example (II)

- Finally...



Correlation Tracking

A dynamic property read *r* and a property write *w* are said to be correlated if *w* writes the value read at *r*, and both *w* and *r* must refer to the same property name

Correlation Tracking

A dynamic property read *r* and a property write *w* are said to be correlated if *w* writes the value read at *r*, and both *w* and *r* must refer to the same property name

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
prop = (*) ? "ext" : "ins";
t = src[prop];
dest[prop] = t;
```

Implementing Correlation Tracking

- Identifying Correlations
  - Def-use graph obtained by dataflow analysis.
- Extracting the snippet of code containing the two accesses into a new function with **p** as its parameter.
- Each function is analysed context sensitively with a separate context for each value of **p**

Their solution: Anonymous Function

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
if (*) {
  prop1 = "ext";
  t1 = src[prop1];
  dest[prop1] = t1;
} else {
  prop2 = "ins";
  t2 = src[prop2];
  dest[prop2] = t2;
}
```

(a)

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
prop = (*) ? "ext" : "ins";
(function(ff) {
  t = src[ff];
  dest[ff] = t;
})(prop);
```

(b)

Another possible solution: Cloning

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
prop = (*) ? "ext" : "ins";
t = src[prop];
dest[prop] = t;
```

```
src = {}
dest = {}
src["ext"] = {}
src["ins"] = {}
if (*) {
  prop1 = "ext";
  t1 = src[prop1];
  dest[prop1] = t1;
} else {
  prop2 = "ins";
  t2 = src[prop2];
  dest[prop2] = t2;
}
```

(a)

Evaluation

- Extending WALA Points-to analysis with ***correlation tracking***

Evaluation

- Extending WALA Points-to analysis with ***correlation tracking***
- Access to an ***unknown property*** is handled by WALA by using an ***abstract object property*** *that represent all the property values*

Evaluation

- Extending WALA Points-to analysis with ***correlation tracking***

- Access to an ***unknown property*** is handled by WALA by using an ***abstract object property*** *that represent all the property values*

- Is not sound
  - Cannot handle `with` blocks
  - Cannot handle `eval` and the `Function` Constructor
  - toString and valueOf methods are ignored

Evaluation

- Extending WALA Points-to analysis with ***correlation tracking***

- Access to an ***unknown property*** is handled by WALA by using an ***abstract object property*** *that represent all the property values*

- Is not sound
  - Cannot handle **`with`** blocks
  - Cannot handle **`eval`** and the **`Function`** Constructor
  - toString and valueOf methods are ignored

- Four results
  - Handling or not handling **`call`/`apply`** support ($^{+}$, $^{-}$)
  - Enable *Correlations analysis* or disable it, *Baseline*

Results

non-blank lines

correlated access pair

Framework	Home Page	Version	LOC	Correlated Pairs
dojo	`http://www.dojotoolkit.org`	1.6.1	4748	20
jquery	`http://jquery.com`	1.6.1	5896	34
mootools	`http://mootools.net`	1.4.0	3815	41
prototype.js	`http://prototypejs.org`	1.7	4956	9
yui	`http://developer.yahoo.com/yui`	2.9	24088	31

Framework	Baseline^-	Baseline^+	Correlations^-	Correlations^+
dojo	* (*)	* (*)	3.1 (30.4)	6.7 (*)
jquery	*	*	78.5	*
mootools	0.7	*	3.1	*
prototype.js	*	*	4.4	4.5
yui	*	*	2.2	2.1

Time (in seconds) to build call graphs for the benchmarks, average per framework, '✳' indicates timeout.

Results

Framework	Baseline$^-$	Baseline$^+$	Correlations$^-$	Correlations$^+$
dojo	$\geq$ 60.8% ($\geq$60.4%)	$\geq$ 60.5% ($\geq$60.1%)	16.7% (24.5%)	18.8% ($\geq$28.3%)
jquery	$\geq$ 35.9%	$\geq$ 36.2%	26.7%	$\geq$ 31.5%
mootools	9.5%	$\geq$ 35.5%	9.5%	$\geq$ 10.9%
prototype.js	$\geq$ 40.5%	$\geq$ 40.7%	17.8%	18.7%
yui	$\geq$ 16.6%	$\geq$ 16.6%	12.0%	12.2%

Percentage of functions considered reachable by our analysis, averaged by framework; '≥' indicates that the number is a lower bound due to analysis timeout.

Framework	Baseline$^-$	Baseline$^+$	Correlations$^-$	Correlations$^+$
dojo	$\geq$239.4 ($\geq$240)	$\geq$226.4 ($\geq$225)	0.0 (1)	1.0 ($\geq$11)
jquery	$\geq$244.0	$\geq$249.0	3.0	$\geq$9.0
mootools	0.0	$\geq$29.2	0.0	$\geq$0.0
prototype.js	$\geq$164.5	$\geq$166.0	0.0	0.2
yui	$\geq$29.0	$\geq$34.5	0.0	0.0

Number of highly polymorphic call sites (i.e., call sites with more than five call targets) for the benchmarks, averaged per framework; '≥' indicates that the result is a lower bound due to timeout.

Conclusions

- Points-to analysis for JavaScript is a challenging task

Conclusions

- Points-to analysis for JavaScript is a challenging task
- Correlation tracking technique aids analyzing common JavaScript frameworks
  - Scalability
  - Precision

Conclusions

- Points-to analysis for JavaScript is a challenging task

- Correlation tracking technique aids analyzing common JavaScript frameworks
  - Scalability
  - Precision

- Further work: Needs to make an analysis more sound
  - Support `call` and `apply`
  - Support arguments array

Preguntas?