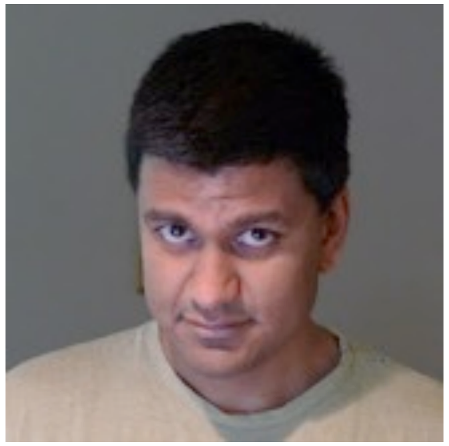
Correlation Tracking for Points-To Analysis of JavaScript


Manu Sridharan


Julian Dolby


Satish Chandra


Max Schäfer


Frank Tip

IBM Research

ECOOP 2012

Tools for Building Web Apps

- Web applications (JavaScript + HTML) increasingly popular
- Often built on rich frameworks like jquery
  - Provide high-level APIs
  - Handle many browser quirks
- Need better tools for framework-based apps
  - Bug finding, refactoring, security, ...

Importance of Pointer Analysis

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

- Pointer analysis needed for call graphs
  - Most method calls are "virtual"
  - Cannot narrow call targets via types / arity
- Analysis must be field-sensitive

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!
- Leads to <u>significant blowup in practice</u>

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
- Standard points-to analysis misses correlation
  - Analysis merges all properties of `src`
  - For frameworks, leads to massive pollution
- **Contribution**: track correlated accesses, improving precision *and* scalability

Andersen's Analysis for JavaScript

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]

For dynamic property accesses → `v = "name"`, `x[v] = y`, `y = x[v]`

For `for..in` loops → `v = x.nextProp()`

Worst-Case Complexity

- View analysis as dynamic transitive closure
  - nodes are memory locations, edges model copying
  - field reads / writes introduce new edges
  - often implemented via points-to set propagation

Java: `x.f = y`

Rule: $$\frac{o \in pt(x)}{pt(y) \subseteq pt(o.\mathtt{f})}$$

$O(N)$ new edges *
$O(N)$ locs to propagate *
$O(N)$ statements = $O(N^3)$

JavaScript: `x[v] = y`

Rule: $$\frac{o \in pt(x) \qquad \mathtt{s} \in pt(v)}{pt(y) \subseteq pt(o.\mathtt{s})}$$

$O(N^2)$ new edges *
$O(N)$ locs to propagate *
$O(N)$ statements = **$O(N^4)$**

Imprecision with Correlated Accesses

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

Andersen's normal form

```
{ prop = src.nextProp(),
  tmp = src[prop],
  dest[prop] = tmp }
```

Possible trace

```
tmp = src[prop];
prop = src.nextProp();
dest[prop] = tmp;
```

Imprecise: `prop` *re-defined between accesses*

Tracking Correlated Accesses

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

↓

```
function extend(dest,src) {
  for (var prop in src)
    if (*) {
      // copy for "foo"
      prop1 = "foo";
      dest[prop1] = src[prop1];
    } else if (*) {
      // copy for "baz"
      prop2 = "baz";
      dest[prop2] = src[prop2];
    } else ...
}
```

- Specialize code for each property name, <u>preventing conflation</u>
- But we only discover property names during analysis...

Function Extraction + Context Sensitivity

```
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);
}
```

- Analyze new functions with <u>clone per property name</u>
  - Similar to object sensitivity / CPA

ext contexts: `p == "foo", p == "baz", ...`

Details

- Detect correlated accesses with simple data flow analysis
- Function extraction handles `this`, unstructured control flow, other corner cases
- Context sensitivity handles correlated accesses across function calls
- See paper for further information

Implementation

- Built using WALA, re-using JS feature handling
  - lexical accesses
  - dynamically-computed property names
  - `Function.prototype.call()` and `apply()`
- Unsound in general (e.g., for `eval`)
  - But still useful, e.g., for bug finding

Evaluation

- Five popular web frameworks
  - Six small benchmarks for each
- Compared with built-in WALA analysis
- Ran with and without `call` / `apply` handling
  - ‘+’ enables handling, ‘-’ disables handling
- Manually transformed one jquery function

Results: Scalability

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

- Dramatic improvements with Correlations^-
  - Useful for an under-approximate call graph
- For '+' configs, issues remain with `call` / `apply`

Results: Highly-Polymorphic Calls

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

- Again, big wins with correlation tracking
- Also significant improvements under timeouts
  - More useful under-approximation

Related Work

- Other JS heap analyses: TAJS [SAS09, SAS10], JSRefactor [OOPSLA11], CFA2 / DrJS [ESOP10], Gulfstream [WebApps10]
  - Cannot analyze JS frameworks
- Complexity: Chaudhuri's technique [POPL08] may shave a log factor
- Context sensitivity: influenced by CPA [ECOOP95] and object sensitivity [TOSEM05]

Conclusions

- Scalable points-to analysis for JS is hard
  - Both in theory and in practice
- Correlated accesses cause imprecision
- Solution: track correlated accesses
  - extract into new functions
  - analyze with targeted context sensitivity
- Future work: attack remaining bottlenecks

WALA
T. J. WATSON LIBRARIES FOR ANALYSIS

http://wala.sourceforge.net