Program Analysis for JavaScript – Challenges and Techniques

Anders Møller

Center for Advanced Software Analysis Aarhus University

Joint work with

Esben Andreasen, Simon Holm Jensen, Peter A. Jonsson, Magnus Madsen, and Peter Thiemann

JavaScript

TAHOO LOCAL Wekcome ericktatere

Jenny wrote on Naoni Simmons's sial. 11-time

facebook Search +

Applications

Photos States

II Events Scrabulous A Super Wall

1.300108

Jet2

Cheap flights from

et2.com

Book now to get new summer sun routes from A£29.99 (one way including taxes) to Cyprus, Crete, Sardinia, La Rochelle,

Jersey, and more More Adz.) Advertice:

1 10 2 21

Q.

edit

Haza Itoma Dial-Up Hap (Original) that

	YAH	OO! LOCAL	Welcone, erickfature (Sign.Oxf. Hz.Account)		Hans Harten Diel-Vie Haas (Orlighteit) Hen		
	+ GET HAP A	ND DIRECTIONS	Printable Var	min +land - ±lave - 🔄 Live Tr	effic	State/1	Pukries -
	A	in the second		New York	as the standing	Map Right Barriston	
	TTND A BUS	SINESS ON THE MAP	Carr		and & Emine W & Marken St	Sharp Same New York	29
	6425761		Seatch	AS "	A L Barry State	1 Contraction of Party	
	Search Rea	alte: sushi	• mma	the maders of Friday	I TOTAL CONTON		
	Taro Sue (718) 308- 445 Dean Directions	hi enza enza St, brownyn, Ar Te - From / More loho					Lawrence St.
	Sushi Ga (718) 122- 288 Jentin Directoria	rden 1931 Turr 11, Brocklyn, Mr. To - From J More Info		**	The second secon	1980 N 198	22
Profile edit Friends	Sushi-Gr (1991) 424- Brooktyn, Directiana	il-Music-Russian Party-Blac 5013 10 To - Frien Here bifu	k Vo Car	1 =		Thates and the second s	Tuer's T
	C Sushi D (718) #50 207 De Ka Directions	8558 In Ave, Brooktyn, MT To - Fram & Park John		and S	000	Gmail – Inbox	
10 10	E Keede Su (212) 425- 125 Print	ahi 2890 St. New York, MV	1 3	E Sall		http://gmail.goog	gle.ci 🔻 💽
247	Direction One Gree (718) 432 1 Greene 7	To - Trum) Hore bills no Swehi Japanese # 2009 Urc. Browkin, NY		NI	GMai	ahansen@gmail.com <u>Fee</u>	<u>dback Contacts Search Mail</u>
View Photos of Me (112) View My Friends (103)	El Faan	Results 1-10 of 16	Next3		byGoogle BET	A Show search options Create	a filter
Get more Super Wall pos		Valle Local Results as a Uni	d Ske	+ d+ 5 - 1 V	Compose Mail	Archive More actions	 Refresh
Play Scrabulous with me		Displaying 10 stories.	care south abortion	See A3	Inhau	Select: All, Read, Unread, Starred,	Unstarred,
Edit My Profits		April 5			xodni	None	ГАр
I am online now.	(IEF)	Jenny commented o	on Helen Isley's photo. shittee	*	Starred 🛱	> me. Ask. Robert (8)	» amail test
会通11日1日	3 률 🖾	If so you were ho	wold when you gave birth to then	C LA CONT	Sent Mail		# gridit toot
		21		TANK TO BE	All Mail	📔 😭 Gmail Team	» Re: [#9130
V Lancaster Friends	Can M				Spam	🔲 ☆ Ask Bjørn Hansen	» test foo bar
and the set of setting a	244.44				Trash	$\Box \bigtriangleup Ask ma (2)$	» pap signer
March 1	. 6			18	<u>Indan</u>) Ask, me (2)	» pgp signed
	109	28 Jerry and David Th	veheld are now friends. Littlym		▼Labels	🔲 🎡 Robert, me (2)	Re: Postfix
	W	April 4			gmail stuff	🗔 🗇 Gmail Team	» amail stuff G
Napiti Comith	Intesar	Jenny wrote on He	len Jaley's wall. It there	×	Edit labels	Select: All Read Upread Starred	Unstarred None
Seemons Dutton	Seddqu	April 3				Colour All, Read, Officad, Staffed,	chatarred, None
1 mar 1		A Jenny is at home.	dipm			Archive More actions	_
18	COLUMN STREET	1 Jenny joined the gr	oup 86Q Drohes. 5 39pm	× .			
	1.11	Hardh 28				You are currently using 0 MB (0%)	of your 1000 MB.

Shortcuts: o-open y-archive c-compose j-older k-newer

JavaScript needs program analysis

- **Testing** is still the only technique programmers have for finding errors in their code
- Program analysis can (in principle) be used for
 - bug detection (e.g. "x.p in line 7 always yields undefined")
 - code completion
 - optimization

None	11.6%			
JavaScript	88.1%			
Flash	15.3%			
Silverlight	0.2%			
Java	0.1%			
	W3Techs.com, 31 January 2014			
Percentages of websites using various client-side programming languages Note: a website may use more than one client-side programming language				

JavaScript is a dynamic language

- Object-based, properties created on demand
- Prototype-based inheritance
- First-class functions, closures
- Runtime types, coercions

NO STATIC TYPE CHECKING NO STATIC CLASS HIERARCHIES

TAJS Type Analysis for JavaScript

The goal: Catch type-related errors using **program analysis**

- Support the full language
- Aim for soundness

Statically detecting type-related errors in JavaScript programs

X 🗉 🗉	Java - Core/src/core.fidy.html - Eclips	e SDK				
<u>Eile Edit Source Navigate Search Project Run Window H</u> elp						
<mark>™~ 🗐 🕲 🖆 ‡~ 0~ %</mark> ∮ ~ ≬ ~ 🏷 🗇~ ⇔~	~] 🌛] 🖶 🎯~] 🤒 😂 🛷 🛛 🔛 🔡 🛃 Java					
📑 🗈 *core.tidy.html 🛛		- 8				
<pre>this.angle = 0; this.coreQuality = 16; this.coreNodes = [] } Player.prototype = new Point; Player.prototype.updateCore = function () { var d, h; if (this.corenodes.length == 0) for (Possible values of corenodes: ity; d++) { h = {</pre>						
🗟 Problems 🔍 Javadoc	B Declaration ● Javascript Analysis View ¤i	- □ i ▼				
File	Line Problem	^				
/Core/src/core.tidy.htm	I 535 (error) TypeError, reading property of null/un	defii 🗸				
N		>				

Likely programming errors

- 1. invoking a non-function value (e.g. undefined) as a function
- 2. reading an absent variable
- 3. accessing a property of null or undefined
- 4. reading an absent property of an object
- 5. writing to variables or object properties that are never read
- 6. calling a function object both as a function and as a constructor, or passing function parameters with varying types
- 7. calling a built-in function with an invalid number of parameters, or with a parameter of an unexpected type

etc.

Which way to go?



type inference? prototype-based inheritance? flow-sensitivity?

heap modeling?

call graph construction?

standard library? coercion?

The TAJS approach

[Jensen, Møller, and Thiemann, SAS'09]

- Dataflow analysis (abstract interpretation) using the monotone framework [Kam & Ullman '77]
- The recipe:
 - 1. construct a **control flow graph** for each function in the program to be analyzed
 - define an appropriate dataflow lattice (abstraction of data)
 - 3. define **transfer functions** (abstraction of operations)

The dataflow lattice (simplified!) Key ideas:

- The analysis maintains an abstract each program point N and call content N × C → State
- Each abstract state provides an for each abstract object L and p.
 State = L × P → Value
- flow sensitivity context sensitivity (object sensitivity)
- pointer analysis with allocation site abstraction
- constant propagation
- recency abstraction
- lazy propagation
- Each abstract value describes pointers and primitive values:
 Value = P(L) × Bool × Str × Num ...
- Details refined through trial-and-error...

Transfer functions, example

- A dynamic property read: **X**[**y**]
- 1. Coerce **x** to objects
- 2. Coerce y to strings
- 3. Descend the object prototype chains to find the relevant properties
- 4. Join the property values

A tiny example...

```
function Person(n) {
  this.setName(n);
  Person.prototype.count++;
}
Person.prototype.count = 0;
Person.prototype.setName = function(n) { this.name = n; }
function Student(n,s) {
  this.b = Person;
  this.b(n);
  delete this.b;
  this.studentid = s.toString();
}
Student.prototype = new Person;
var t = 100026;
var x = new Student("Joe Average", t++);
var y = new Student("John Doe", t);
y.setName("John Q. Doe");
```

does y have a **setName** method at this program point?

An abstract state (as produced by TAJS)



JavaScript web applications

• Modeling JavaScript code is not enough...

- The environment of the JavaScript code:
 - -the ECMAScript standard library
 - -the browser API
 - -the HTML DOM
- around 250 abstract objectswith 500 propertiesand 200 functions...
- -the event mechanism

[Jensen, Madsen, and Møller, ESEC/FSE'11]

Eval in JavaScript

• eval(*S*)

- parse the string S as JavaScript code, then execute it

- Challenging for JavaScript static analysis
 - the string is dynamically generated
 - the generated code may have side-effects
 - and JavaScript has poor encapsulation mechanisms



Eval is evil

- ... but most uses of eval are not very complex
- So let's transform eval calls into other code!
- How can we soundly make such transformations if we cannot analyze code with eva 7?

Which came first?

Analysis or transformation





Whenever TAJS detects new dataflow to eval, the eval transformer is triggered

[Jensen, Jonsson, and Møller, ISSTA'12]

An example

The dataflow analysis propagates dataflow until the fixpoint is reached

– iteration 1: y is "foo", i is 0

(the dataflow analysis can now proceed into foo)

- iteration 2: y is "foo", i is AnyNumber

eval
$$(y + "(" + i + ")") \Rightarrow foo(i)$$

(would not work if i could be any string)

Ingredients in a static analyzer for JavaScript applications

We need to model The language semantics The standard library (incl. eval) The browser API (the HTML DOM, the event system, etc.)

Mission complete?





Why use jQuery (or other libraries)?

- **★** Patches browser incompatibilities
- **★** CSS3-based DOM navigation
- **★** Event handling
- **★** AJAX (client-server communication)
- \bigstar UI widgets and animations
- **★** 1000s of plugins available

An appetizer

Which code fragment do you prefer?

```
var checkedValue;
var elements = document.getElementsByTagName('input');
for (var n = 0; n < elements.length; n++) {
    if (elements[n].name == 'someRadioGroup' &&
        elements[n].checked) {
        checkedValue = elements[n].value;
    }
}
```

var checkedValue = \$('[name="someRadioGroup"]:checked').val();

Investigating the beast

jQuery version	LOC	load-LOC	
1.0.0	996	272	
1.1.0	1, 141	300	lines executed
1.2.0	1,504	296	when the library
1.3.0	2,150	648	initializes itself
1.4.0	2,851	737	after loading
1.5.0	3,610	924	
1.6.0	3,923	1,003	
1.7.0	4,096	1,118	
1.8.0	4,075	1,157	
1.9.0	4,122	1, 161	
1.10.0	4,144	1,193	
2.0.0	3,775	1,101	





T. J. WATSON LIBRARIES FOR ANALYSIS

[Schäfer, Sridharan, Dolby, Tip. Dynamic Determinacy Analysis, PLDI'13]

Experimental results for jQuery with WALA:

- can analyze a JavaScript program that <u>loads jQuery and does nothing else</u>
- no success on jQuery 1.3 and beyond $\boldsymbol{\varpi}$

The **WALA** approach:

- dynamic analysis to infer *determinate* expressions that always have the same value in any execution (but for a specific calling context)
- 2) exploit this information in context-sensitive pointer analysis

Example of imprecision that explodes

A dynamic property read: **X**[**y**]

- if x may evaluate to the global object
- and y may evaluate to a unknown string
- then x[y] may yield
 eval, document, Array, Math, ...





jQuery: sweet on the outside, bitter on the inside

A representative example from the library initialization code:

```
jQuery.each("ajaxStart ajaxStop ... ajaxSend".split("
function(i, o) {
    jQuery.fn[o] = function(f) {
        return this.on(o, f);
    };
});
```

which could have been written like this:

```
jQuery.fn.ajaxStart = function(f) { return this.on("ajaxStart", f); };
jQuery.fn.ajaxStop = function(f) { return this.on("ajaxStop", f); };
...
jQuery.fn.ajaxSend = function(f) { return this.on("ajaxSend", f); };
```

```
each: function (obj, callback, args) {
  var name, i = 0, length = obj.length,
  isObj = length === undefined || jQuery.isFunction(obj);
  if (args) {
     ... // (some lines omitted to make the example fit on one slide)
  } else {
    if (isObj) {
      for (name in obj) {
        if (callback.call(obj[name], name, obj[name]) === false) {
          break;
        }
      }
    } else {
      for (; i < length ;) {</pre>
        if (callback.call(obj[i], i, obj[i++]) === false) {
          break;
                                                  Lots of
        }
      }

    overloading

    }

    reflection

  }
  return obj;
                                                    callbacks
}
```

Our recent results, by improving TAJS

- **TAJS** can now analyze (in reasonable time)
 - the load-only program for **11** of 12 versions of jQuery
 - 27 of 71 small examples from a jQuery tutorial
- Very good precision for type analysis and call graphs
- Analysis time: 1-24 seconds (average: 6.5 seconds)

TAJS analysis design

- Whole-program, flow-sensitive dataflow analysis
- Constant propagation
- Heap modeling using allocation site abstraction
- Object sensitivity (a kind of context sensitivity)
- Branch pruning (eliminate dataflow along infeasible branches)
- Parameter sensitivity
- Loop specialization



Context-sensitive heap abstraction

[Andreasen and Møller, OOPSLA'14]

```
each: function (obj, callback, args) {
                                                  with parameter
  var name, i = 0, length = obj.length,
                                                  sensitivity, these
  isObj = length === undefined || jQuery.isFuncti
                                                  become constants
  if (args) {
  } else {
                              constant propagation...
    if (isObj)
      for (name in obj) {
        if (callback.call(obj[name], name, obj[name]) === false) {
          break;
        }
                           branch pruning logically
      }
                           eliminates several branches
    } else {
      for (; i < length ;) {</pre>
        if (caliback.call(obj[i], i, obj[i++]) === false) {
          break;
                            specializing on i effectively
        }
      }
                            unrolls the loop
    }
  }
                  context-sensitive heap abstraction keeps the
  return obj;
                  ajaxStart, ajaxStop, etc. functions separate
}
```

The technical side...

- The analysis maintains an abstract state for each program point N and call context C:
 N × C → State
- Old TAJS:
 C = P(L) (object sensitivity)
 L = N (L: abstract memory locations)
- New TAJS: $C = \mathcal{P}(L) \times (A \rightarrow Value) \times (B \rightarrow Value)$ $L = N \times C$ $Dop specialization (B: selected local variables) \downarrow \downarrow$ $C = \mathcal{P}(L) \times (A \rightarrow Value) \times (B \rightarrow Value)$

context-sensitive heap abstraction

Conclusion

- JavaScript programmers need better tools!
- Static program analysis can detect type-related errors, find dead code, build call graphs, etc.
 - dataflow analysis to model the ECMAScript standard
 - model of the standard library, browser API, and HTML DOM
 - rewrite calls to eval during analysis
 - handle complex libraries by boosting analysis precision
- Progress, but far from a full solution...