Static Analysis for JavaScript – Challenges and Techniques

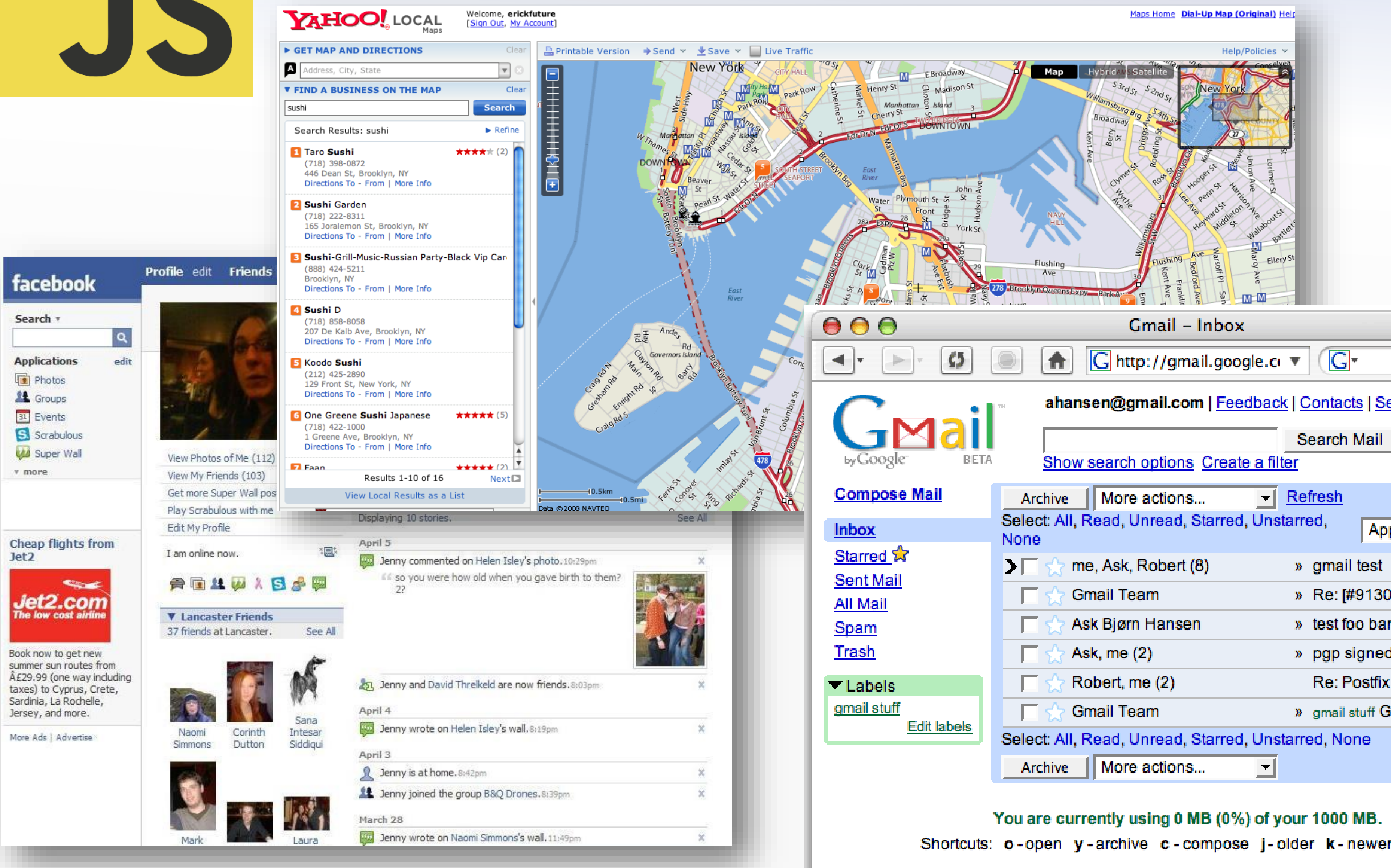
Anders Møller

Center for Advanced Software Analysis

Aarhus University

SAS 2015 – Saint-Malo

JavaScript



YAHOO! LOCAL Maps
Welcome, erickfuture
[Sign Out, My Account]
Maps Home
Dial-Up Map (Original)
GET MAP AND DIRECTIONS
FIND A BUSINESS ON THE MAP
sushi
Search
Search Results: sushi
Taro Sushi
(718) 398-0872
446 Dean St, Brooklyn, NY
Sushi Garden
(718) 222-8311
165 Joralemon St, Brooklyn, NY
Sushi-Grill-Music-Russian Party-Black Vip Car
(888) 424-5211
Brooklyn, NY
Sushi D
(718) 858-8058
207 De Kalb Ave, Brooklyn, NY
Koodo Sushi
(212) 425-2890
129 Front St, New York, NY
One Greene Sushi Japanese
(718) 422-1000
1 Greene Ave, Brooklyn, NY
Directions To - From | More Info
Results 1-10 of 16
Next
View Local Results as a List
Printable Version
Send
Save
Live Traffic
Help/Policies
Map
Hybrid
Satellite
New York
facebook
Profile
edit
Friends
Search
Applications
Photos
Groups
Events
Scrabulous
Super Wall
more
View Photos of Me (112)
View My Friends (103)
Get more Super Wall pos
Play Scrabulous with me
Edit My Profile
I am online now.
Lancaster Friends
37 friends at Lancaster.
See All
Cheap flights from Jet2
Jet2.com
The low cost airline
Book now to get new summer sun routes from Â£29.99 (one way including taxes) to Cyprus, Crete, Sardinia, La Rochelle, Jersey, and more.
More Ads | Advertise
Naomi Simmons
Corinth Dutton
Sana Intesar Siddiqui
Mark
Laura
Displaying 10 stories.
April 5
Jenny commented on Helen Isley's photo.
so you were how old when you gave birth to them? 2?
Jenny and David Threlkeld are now friends.
April 4
Jenny wrote on Helen Isley's wall.
April 3
Jenny is at home.
Jenny joined the group B&Q Drones.
March 28
Jenny wrote on Naomi Simmons's wall.
Gmail – Inbox
http://gmail.google.c
Gmail by Google BETA
ahansen@gmail.com | Feedback | Contacts
Search Mail
Show search options
Create a filter
Compose Mail
Inbox
Starred
Sent Mail
All Mail
Spam
Trash
Labels
gmail stuff
Edit labels
Archive
More actions...
Refresh
Select: All, Read, Unread, Starred, Unstarred, None
me, Ask, Robert (8)
gmail test
Gmail Team
Ask Bjørn Hansen
test foo bar
Ask, me (2)
pgp signed
Robert, me (2)
Re: Postfix
You are currently using 0 MB (0%) of your 1000 MB.
Shortcuts: o - open y - archive c - compose j - older k - newer

JavaScript needs static analysis

- **Testing** is still the main technique programmers have for finding errors in their code

- **Static analysis** can (in principle) be used for
  - bug detection (e.g. "x.p in line 7 always yields *undefined*")
  - security vulnerability detection
  - code completion and navigation in IDEs
  - optimization

JavaScript is a *dynamic language*

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

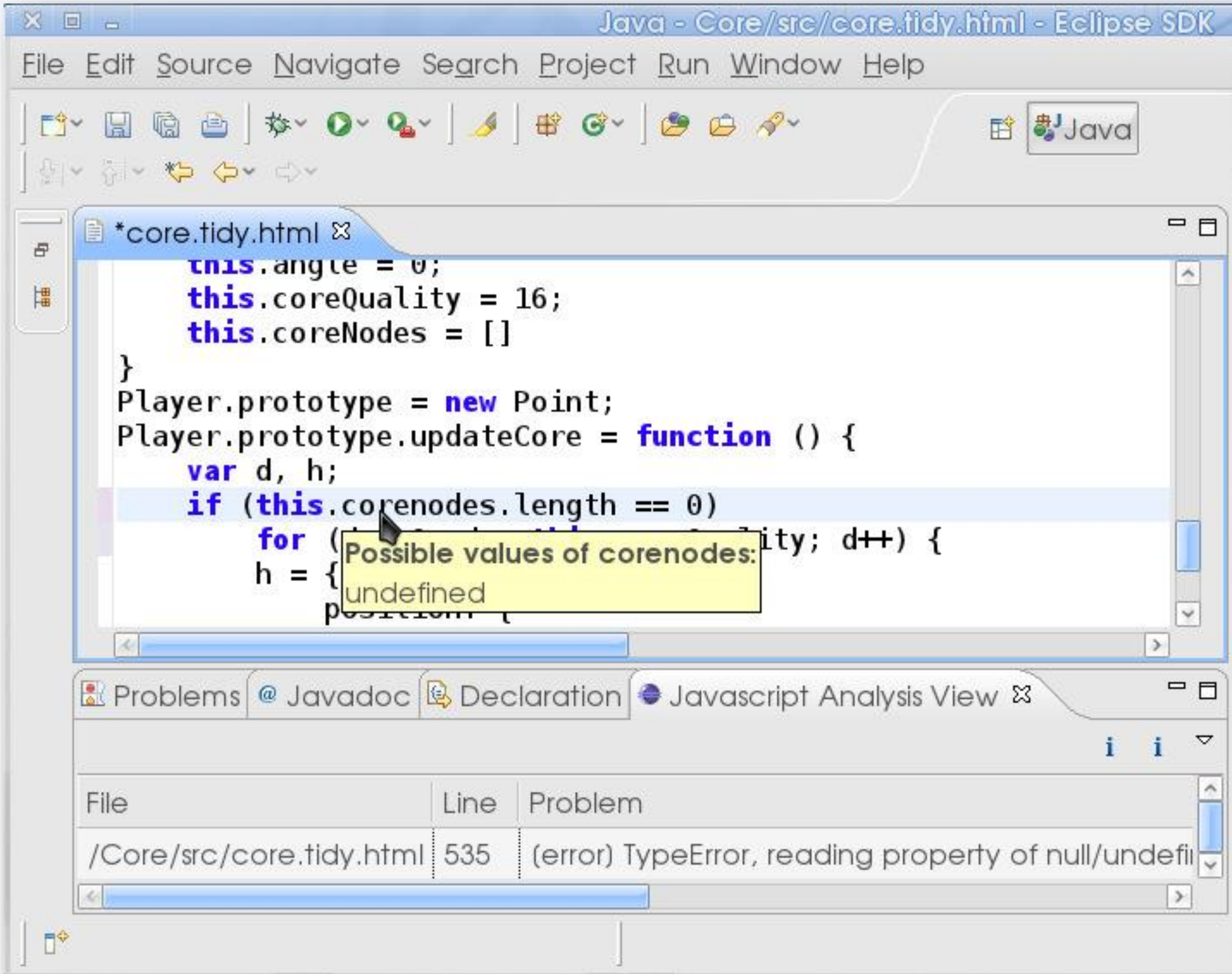
NO STATIC TYPE CHECKING
NO STATIC CLASS HIERARCHIES

TAJS Type Analysis for JavaScript

Goals:

- Catch type-related errors using **static analysis**
- Support **the full language**
- Aim for **soundness**

TAJS in Eclipse



Related static analysis tools

WALA

T. J. WATSON LIBRARIES FOR ANALYSIS

SAFE: JS Analysis Framework

JSAI: A Static Analysis Platform for JavaScript

Type-related errors in JavaScript

```
var x =
  ["Static","Analysis","Symposium"];
for (var i = 0; i < x.lenght; i++) {
  console.log(x[i]);
}
```

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.

See also *The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript*, Pradel & Sen, ECOOP 2015

Research methodology

identify interesting problem

design initial analysis

implement,
evaluate experimentally

works perfectly?

too imprecise? too slow?

identify
bottleneck

refine
analysis
design

Which way to go?

type inference?

prototype-based inheritance?

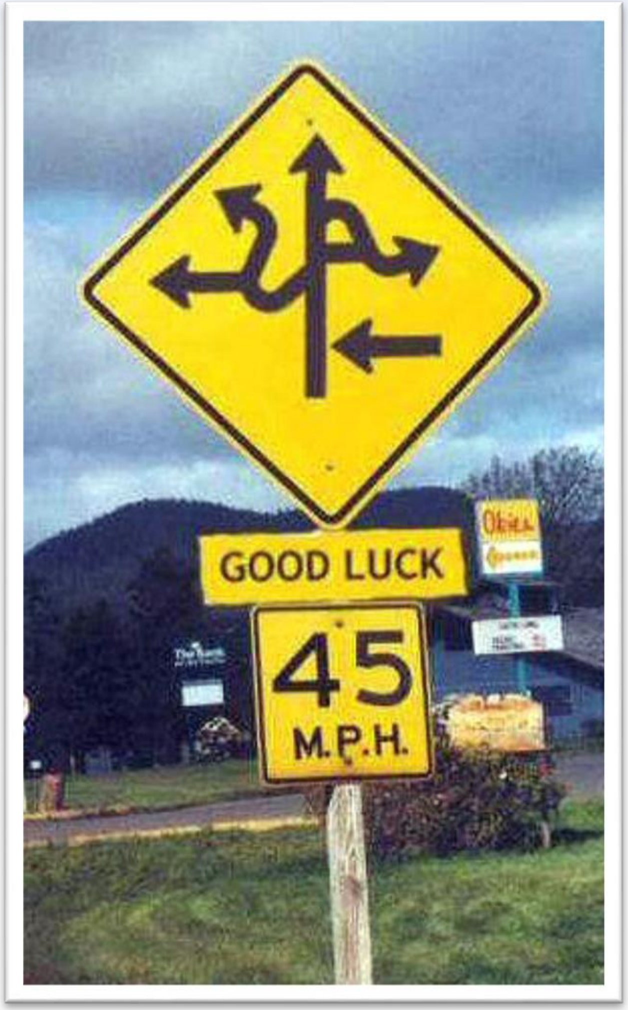
flow-sensitivity?

heap modeling?

call graph construction?

standard library?

coercion?



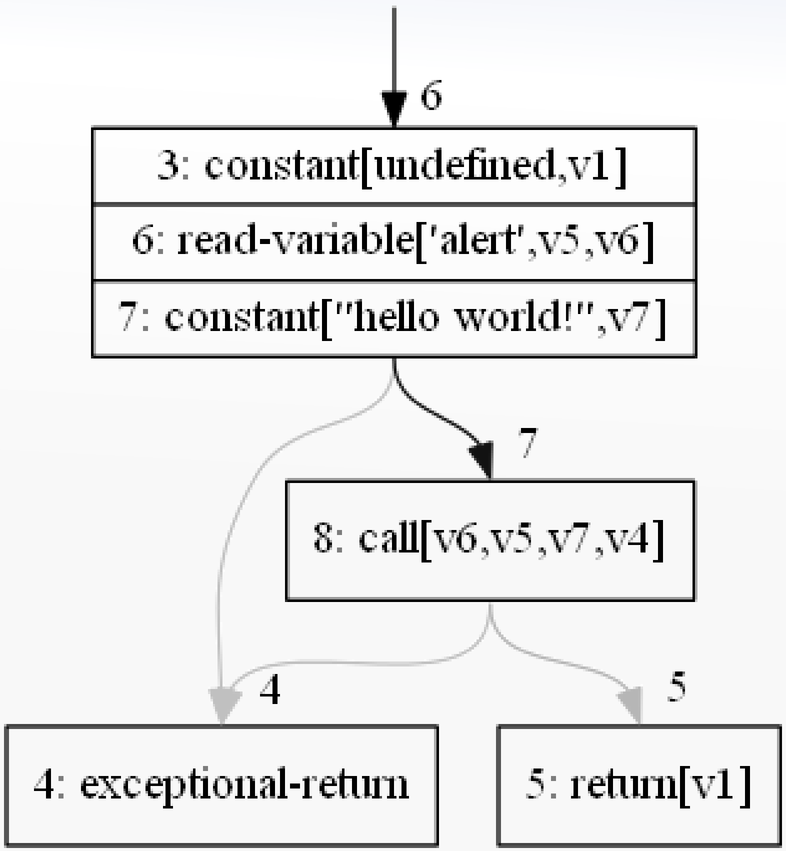
The TAJS approach

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

- Dataflow analysis / abstract interpretation using monotone frameworks
  [Kam & Ullman '77]

- The recipe:
  1. construct a **control flow graph** for each function in the program to be analyzed
  2. define an appropriate **dataflow lattice** (abstraction of data)
  3. define **transfer functions** (abstraction of operations)

Control flow graphs



- Convenient intermediate representation of JavaScript programs
- ***Nodes*** describe primitive instructions
- ***Edges*** describe *intra*-procedural control-flow
- Relatively **high-level** IR (unlike e.g. λ_{JS})

The dataflow lattice (simplified!)

- For each program point **N** an[d call context] **C**, the analysis maintains an ab
  N × C → State
- Each abstract state provide
  for each abstract object **L** a
  State = L × P → Value
- Each abstract value describe
  primitive values:
  Value = $\mathcal{P}$**(L) × Bool × Str × Num ...**
- *Details refined through trial-and-error...*

<u>Key ideas:</u>

- flow sensitivity
- context sensitivity (object sensitivity)
- pointer analysis with allocation site abstraction
- constant propagation

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");
```

declares a "class" named Person

declares a "static field" named count

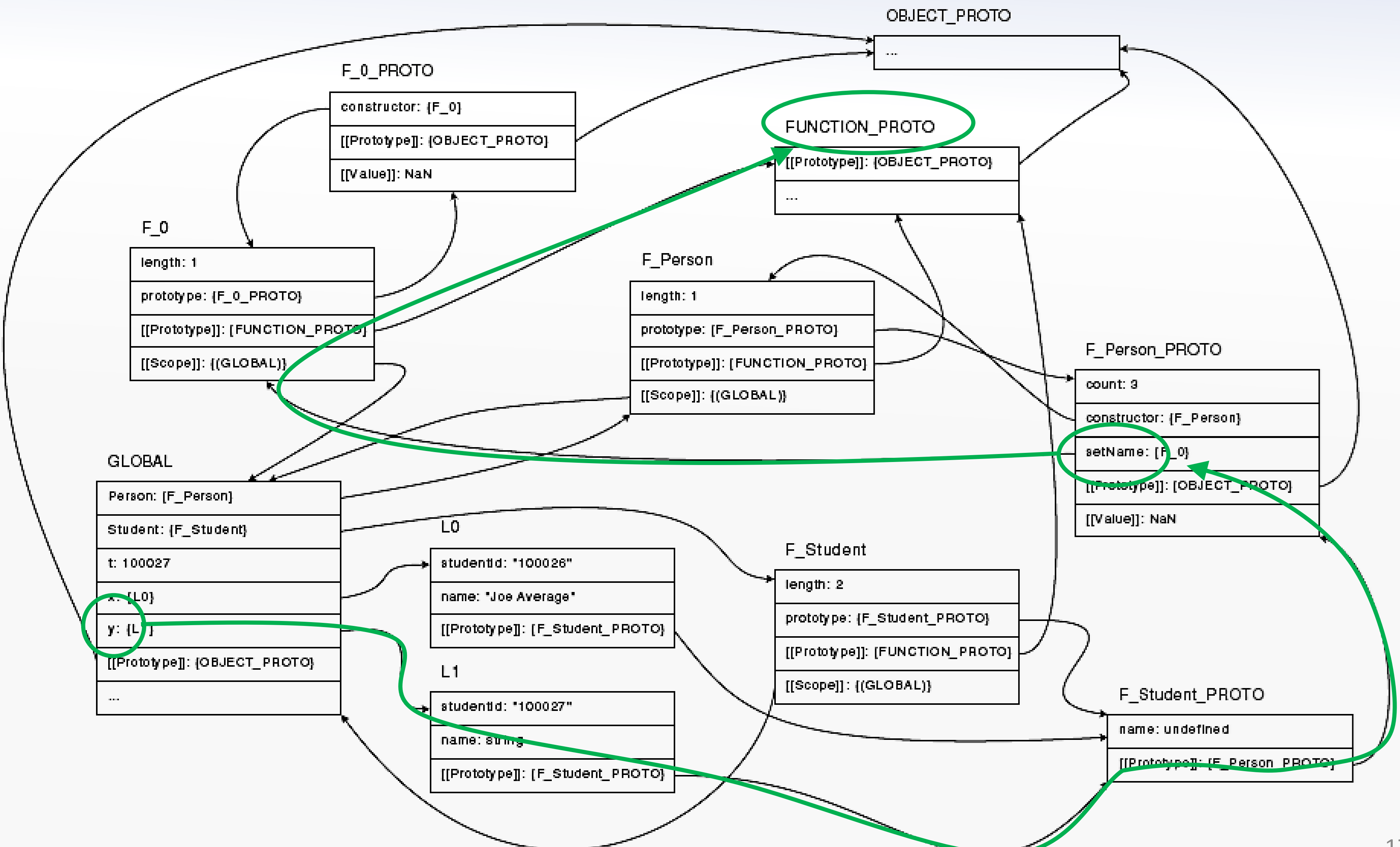
declares a shared method named setName

declares a "sub-class" named Student

creates two Student objects...

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
  - the event mechanism

around 250 abstract objects
with 500 properties
and 200 functions…

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

Some experiments

Good results on analyzing small web applications from Chrome Experiments, IE 9 Test Drive, and 10K Challenge

Some ways to measure analysis precision:

- most call sites and property reads are safe
- most call sites are monomorphic
- most expressions have a unique type
- most spelling errors cause type-related errors

General observation: **higher precision ⇒ faster analysis**

The eval of JavaScript

- `eval(`*s*`)`
  - parse the string *s* as JavaScript code, then execute it
- Challenging for static analysis
  - the string is dynamically generated
  - the generated code may have side-effects
  - and JavaScript has poor encapsulation mechanisms

Eval in practice

```
function _var_exists(name) {
  try {
    eval('var foo = ' + name + ';');
  } catch (e) {
    return false;
  }
  return true;
}

var Namespace = {
  create: function(path) {
    var container = null;
    while (path.match(/^(\w+)\.?/)) {
      var key = RegExp.$1;
      path = path.replace(/^(\w+)\.?/, "");
      if (!container) {
        if (!_var_exists(key))
          eval('window.' + key + ' = {};');
        eval('container = ' + key + ';');
      } else {
        if (!container[key]) container[key] = {};
          container = container[key];
      }
    }
  }
};
```

```
return name in window;
```

(also avoids conflicts if name is "name" or "foo")

```
window[key] = {};
```

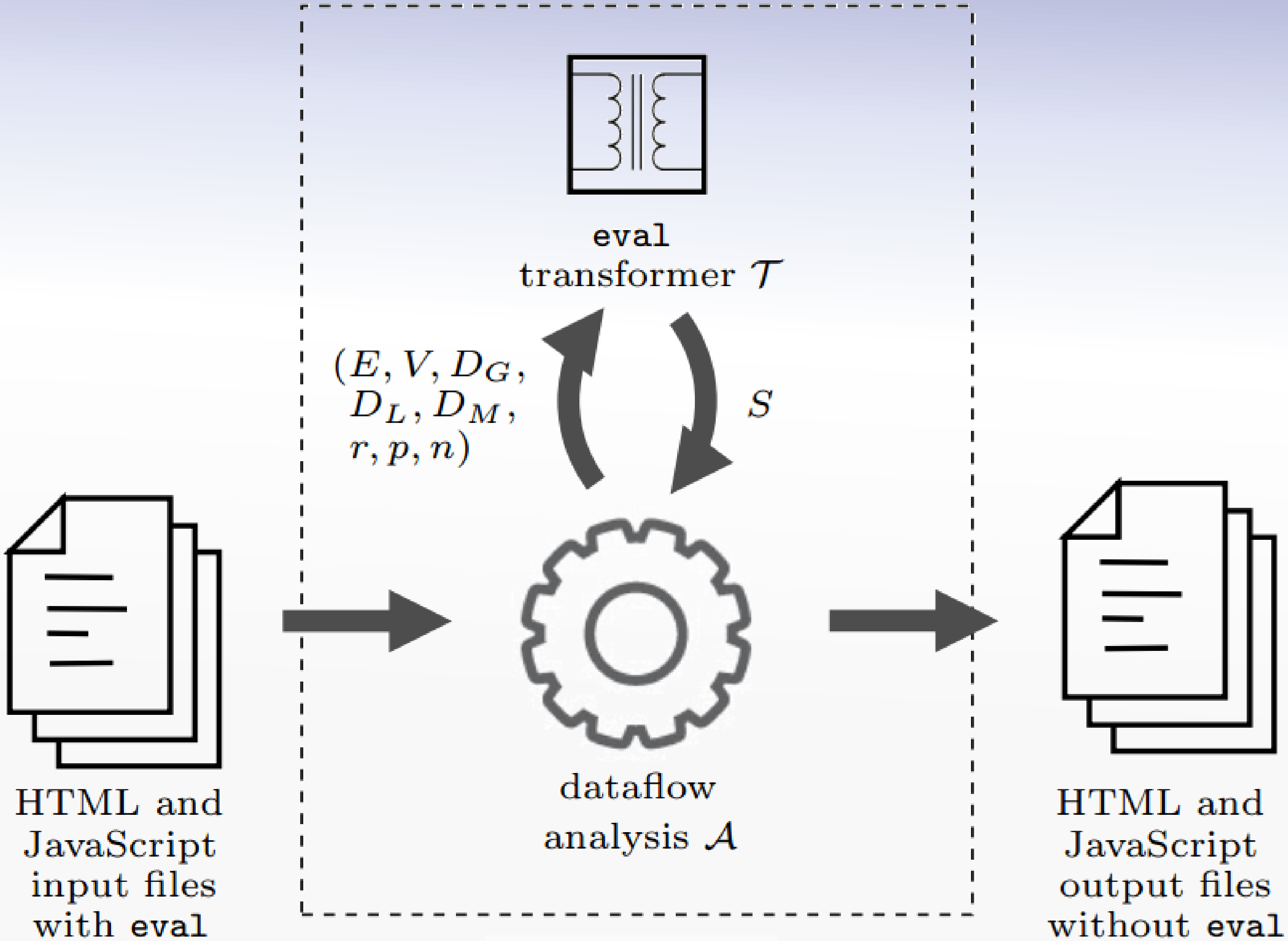
http://www.chromeexperiments.com/detail/canvas-cycle/

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 `eval`?*

Which came first?

Analysis or transformation



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

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

A simple example

```
var y = "foo"
for (i = 0; i < 10; i++) {
  eval(y + "(" + i + ")")
}
```

The dataflow analysis propagates dataflow until the fixpoint is reached

- iteration 1: y is `"foo"`, `i` is `0`

  `eval(y + "(" + i + ")")` ⇨ `foo(0)`

  (the dataflow analysis can now proceed into `foo`)
- iteration 2: y is `"foo"`, `i` is *AnyNumber*

  `eval(y + "(" + i + ")")` ⇨ `foo(i)`

  (would not work if `i` could be any string)
- …

A real-world example

```
get_cookie = function (name) {
  var ca = document.cookie.split(';');
  for (var i = 0, l = ca.length; i < l; i++) {
    if (eval("ca[i].match(/\\b" + name + "=/)"))
      return decodeURIComponent(ca[i].split('=')[1]);
  }
  return '';
}
get_cookie('clicky_olark')
get_cookie('no_tracky')
get_cookie('_jsuid')
```

TAJS tells us that name is one of these three strings!

```
     eval("ca[i].match(/\\b" + name + "=/)")
                         ⇩
name==="clicky_olark" ? ca[i].match(/\\bclicky_olark=/)
 : name==="no_tracky" ? ca[i].match(/\\bno_tracky=/)
                      : ca[i].match(/\\b_jsuid=/)
```

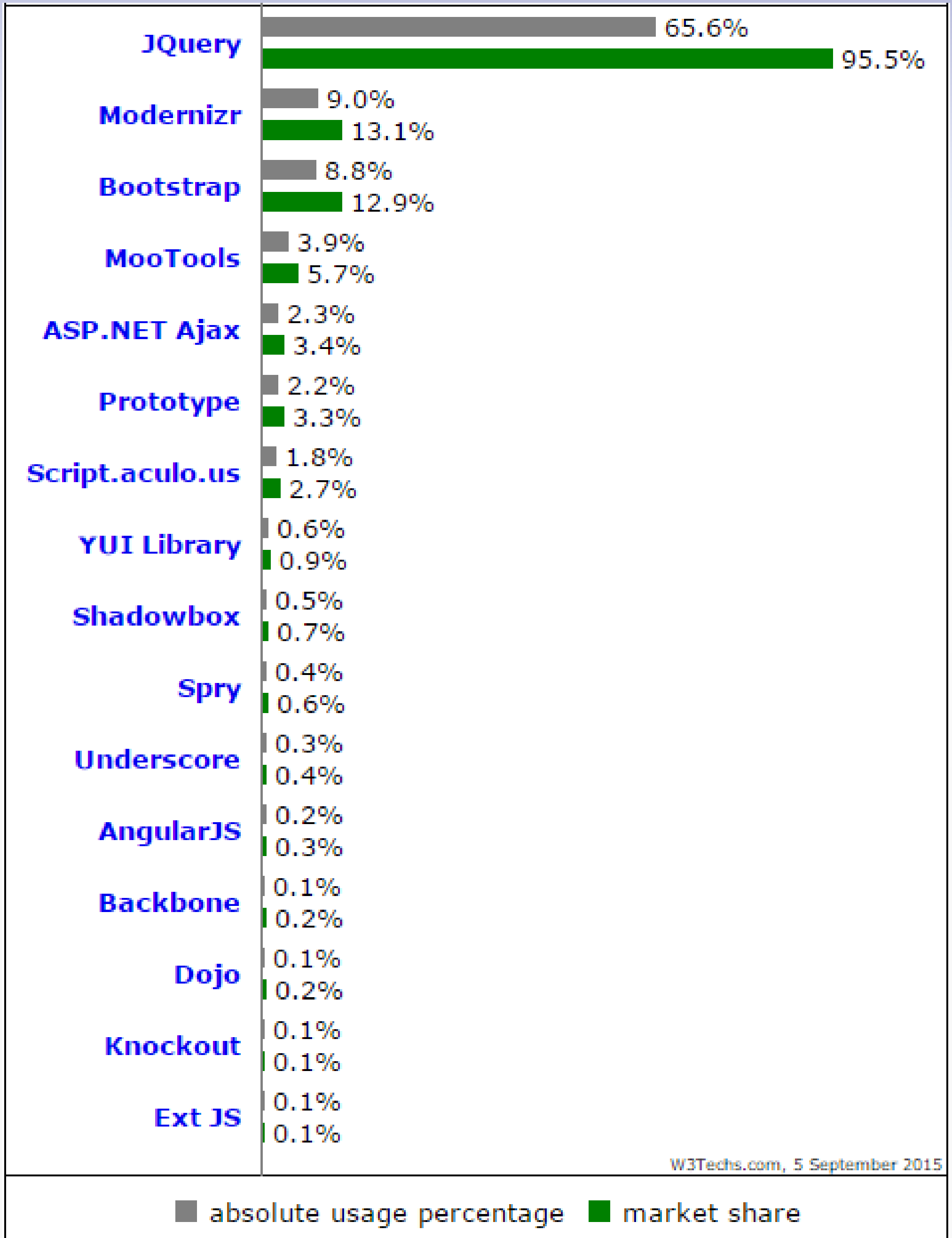
Ingredients in a static analyzer for JavaScript applications

We need to model

- ☑ the language semantics
- ☑ the standard library (incl. `eval`)
- ☑ the browser API (the HTML DOM, the event system, etc.)

Mission complete?

M[illegible] j[illegible] c[illegible]ple[illegible]?



Percentages of websites using various JavaScript libraries
Note: a website may use more than one JavaScript library

Why use jQuery (or other libraries)?

- Patches browser incompatibilities
- CSS3-based DOM navigation
- Event handling
- AJAX (client-server communication)
- 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
1.2.0	1,504	296
1.3.0	2,150	648
1.4.0	2,851	737
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

lines executed when the library initializes itself after loading

WALA

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 loads jQuery and does nothing else
- no success on jQuery 1.3 and beyond ☹

The **WALA** approach:

1) 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`, ...

consequence

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 ;) {
        if (callback.call(obj[i], i, obj[i++]) === false) {
          break;
        }
      }
    }
  }
  return obj;
}
```

Lots of
- **overloading**
- **reflection**
- **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)

- Perhaps not impressive, but progress ☺

[Andreasen and Møller, OOPSLA’14]

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**

NEW

[Andreasen and Møller, OOPSLA’14]

```
each: function (obj, callback, args) {
  var name, i = 0, length = obj.length,
  isObj = length === undefined || jQuery.isFuncti
  if (args) {
    ...
  } else {
    if (isObj) {
      for (name in obj) {
        if (callback.call(obj[name], name, obj[name]) === false) {
          break;
        }
      }
    } else {
      for (; i < length ;) {
        if (callback.call(obj[i], i, obj[i++]) === false) {
          break;
        }
      }
    }
  }
  return obj;
}
```

with ***parameter sensitivity***, these become constants

constant propagation...

branch pruning logically eliminates several branches

specializing on i effectively unrolls the loop

context-sensitive heap abstraction keeps the ajaxStart, ajaxStop, etc. functions separate

Observations

- The analysis is essentially executing the critical library code *concretely*!
  - but allowing abstract values, e.g. from the application code
- A kind of “**static** determinacy analysis”

Experiments show that

- *all* the tricks must be enabled to get positive results
- unhandled cases are likely *not* due to too much precision

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…

CENTER FOR ADVANCED SOFTWARE ANALYSIS

http://casa.au.dk/

European Research Council

Established by the European Commission