Modern Type Systems for Dynamic Languages

Ravi Chugh
Web Platform

Untrusted Code

Scripting Languages
Goal: Rigorous Foundations

Web Platform

Untrusted Code

Scripting Languages

Types + Logic
1995  Web Page = Static HTML + Links + Client-Side Code

```html
<html>
  <script type="javascript">
    loginButton.onclick = function () {
      if (passwordBox.value == "") {
        alert("Enter password!");
      } else {
        ...
      }
    }
  </script>
</html>
```
1995

No Static Types

“Dynamic” Features Make it Easy to Experiment...

... an easy-to-use “scripting language” as a companion to Java

Brendan Eich, JavaScript creator
## 2014

### Gmail

<table>
<thead>
<tr>
<th>Name</th>
<th>Message</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy, me, Westley</td>
<td>Lunch on Wednesday - Hey Jeff, I'll be in meetings...</td>
<td>3:03pm</td>
</tr>
<tr>
<td>David Alderman</td>
<td>How's it going? - Hey there, I'm just se...</td>
<td>11:28am</td>
</tr>
<tr>
<td>Wyatt, Lisa, Mike (4)</td>
<td>Reference call - OK, I'll add it to my calendar. Thanks!...</td>
<td>9:45am</td>
</tr>
<tr>
<td>Cindy, me (2)</td>
<td>WidgetDesigns Prospectus 2011 - Please see attached ...</td>
<td>9:12am</td>
</tr>
<tr>
<td>me, Carlos (3)</td>
<td>Wednesday movie - We can see if there ...</td>
<td>Jul 21</td>
</tr>
<tr>
<td>Twitter</td>
<td>Bob Johnson (bjohnson??) is now following you...</td>
<td>Jul 21</td>
</tr>
<tr>
<td>Tracy Logan</td>
<td>Update for Tracy's exhibition - Friends, Just a friendly ...</td>
<td>Jul 21</td>
</tr>
<tr>
<td>me, Yusef, Westley (6)</td>
<td>Los Angeles? - Hey guys, I'll be in LA between the we...</td>
<td>Jul 21</td>
</tr>
<tr>
<td>me, Dan (2)</td>
<td>WX-700 - Hello Dan, Indeed, the latest r...</td>
<td>Jul 21</td>
</tr>
<tr>
<td>Facebook</td>
<td>Paula Brisbane commented on your post. - f...</td>
<td>Jul 20</td>
</tr>
<tr>
<td>Sarah Richards</td>
<td>Following up - Hello Jeff, What are your thoughts on the...</td>
<td>Jul 20</td>
</tr>
<tr>
<td>Bernard, me (3)</td>
<td>Dentist recommendations - Thanks! On Jul 18, 2011, Je...</td>
<td>Jul 20</td>
</tr>
</tbody>
</table>

### Contacts

- Amy Villanova
  - "Thinking about going to the movies. Who's in?"
- Tracy Logan
  - "Still looking for a sublet in New York for July and August. Anyone have leads?"
2014

Asked whether Gmail would ever open up an API for developers to code add-ons and plug-ins with more official support and stability, Kowitz suggested it was unlikely. **Gmail is based on “hundreds of thousands” of lines of JavaScript**, Kowitz said, and it's a code base that “changes so quickly.” That said, Jackson said the Gmail team "loves" to see third-party functionality being developed, and the team tries to make developers aware of changes they know will affect those products.
2014

Asked whether Gmail would ever open up an API for developers to code add-ons and plug-ins with more official support and stability, Kowitz suggested it was unlikely. **Gmail is based on “hundreds of thousands” of lines of JavaScript**, Kowitz said, and it's a code base that “changes so quickly.”

That said, Jackson said the Gmail team "loves" to see third-party functionality being developed, and the team tries to make developers aware of changes they know will affect those products.
2014

JavaScript is Pervasive

<table>
<thead>
<tr>
<th>Count</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>599,605</td>
<td>C#</td>
</tr>
<tr>
<td>583,322</td>
<td>Java</td>
</tr>
<tr>
<td>555,164</td>
<td>JavaScript</td>
</tr>
<tr>
<td>532,851</td>
<td>PHP</td>
</tr>
<tr>
<td>273,086</td>
<td>Python</td>
</tr>
<tr>
<td>268,725</td>
<td>C++</td>
</tr>
<tr>
<td>129,483</td>
<td>C</td>
</tr>
<tr>
<td>97,154</td>
<td>Ruby</td>
</tr>
<tr>
<td>15,044</td>
<td>Haskell</td>
</tr>
<tr>
<td>2,175</td>
<td>OCaml</td>
</tr>
</tbody>
</table>

http://stackoverflow.com/tags
02-26-14
2014

JavaScript is Pervasive

Other Dynamic Languages Are, Too

<table>
<thead>
<tr>
<th>Count</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>599,605</td>
<td>C#</td>
</tr>
<tr>
<td>583,322</td>
<td>Java</td>
</tr>
<tr>
<td>555,164</td>
<td>JavaScript</td>
</tr>
<tr>
<td>532,851</td>
<td>PHP</td>
</tr>
<tr>
<td>273,086</td>
<td>Python</td>
</tr>
<tr>
<td>268,725</td>
<td>C++</td>
</tr>
<tr>
<td>129,483</td>
<td>C</td>
</tr>
<tr>
<td>97,154</td>
<td>Ruby</td>
</tr>
<tr>
<td>15,044</td>
<td>Haskell</td>
</tr>
<tr>
<td>2,175</td>
<td>OCaml</td>
</tr>
</tbody>
</table>

http://stackoverflow.com/tags
02-26-14
2014

JavaScript is Pervasive
2014

JavaScript is
Pervasive

Third-Party Ads
2014

JavaScript is Pervasive

Third-Party Ads

Banking  Shopping  News

Third-Party Apps
2014

Third-Party
JavaScript is
Pervasive
No longer just “scripting”

... an easy-to-use “scripting language” as a companion to Java
Significant Interest For:

1. Development Tools
2. Reliability / Security
3. Performance

But JS is hard to reason about!
My Ph.D.

Static Types for Dynamic Languages

JavaScript, Python, Ruby, PHP
Share Common Challenges

New Techniques Apply Broadly, Even to Statically Typed Languages
Static Types

Describe sets of run-time values

“booleans”

“objects with foo field”

“integers”

“urls that are safe to visit”

“non-null pointers”
Static Types

Prevent classes of run-time errors

“multiply int and bool”

“foo field not found”

“null pointer exception”

“redirect to evil.com”
\[ f(x) \]

\[ \text{"actual args"} \rightarrow \text{"expected args"} \]

\[ \text{Type}_1 \subseteq \text{Type}_2 \]

\[ \text{"function call produces no error"} \]
$f(x)$

```
| "actual args" | "expected args"
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type$_1$</td>
<td>⊆</td>
</tr>
<tr>
<td></td>
<td>Type$_2$</td>
</tr>
</tbody>
</table>
```

"function call may produce error"
For all programs $p$ in $\text{PROGRAMS}$.

If $p$ has a type $T$ in $\text{TYPES}$.

Then $p$ does not fail with any error $e$ in $\text{ERRORS}$. 
For all programs $p$ in \text{PROGRAMS}

If $p$ has a type $T$ in \text{TYPES}

Then $p$ does not fail with any error $e$ in \text{ERRORS}

GOAL: Reason About More Programs
For all programs \( p \) in \( \text{PROGRAMS} \),

If \( p \) has a type \( T \) in \( \text{TYPES} \),

Then \( p \) does not fail with any error \( e \) in \( \text{ERRORS} \).

**GOAL:** Reason About More Programs

**GOAL:** Eliminate More Errors
TYPES
Refinement Types

Types + Logic
Refinement Types

Types + Logic

"actual args" ⊆ "expected args"

\{ x \mid p \} ⊆ \{ x \mid q \}

p ⇒ q
Refinement Types

Types

\{ x \mid p \}\quad\text{Logic}\quad f \::\::\quad\{ y \mid q \}\n
Nested Refinements [POPL '12]
Key to Encode Dynamic Features
Expressiveness

Usability

Types + Logic

Dependent JavaScript (DJS)
[POPL ’12, OOPSLA ’12]

Verification
Hoare Logics
Model Checking
Abstract Interpretation
Theorem Proving
...

Expressiveness
Outline

Why is JavaScript Important?
Why is JavaScript Hard?
Static Types via Logic
Security via Logic
Looking Ahead

- Why JavaScript?
- Challenges
- Types via Logic
- Security via Logic
- Looking Ahead

- Programs
- Types
- Errors
Outline

Why is JavaScript Important?
Why is JavaScript Hard?
Static Types via Logic
Security via Logic
Looking Ahead
Why is JavaScript Hard?

Outline

Why is JavaScript Hard?

- Lightweight Reflection
- Dictionary Objects
- Additional Challenges
```javascript
function negate(x) {
    if (typeof x === "number") {
        // Code to negate number
    }
}
```

Q: What is my “type”? 

X
### Why JavaScript?

- **Security via Logic**

### Challenges

- **Types via Logic**

### Looking Ahead

---

#### All JavaScript Values

<table>
<thead>
<tr>
<th>“number”</th>
<th>“object”</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>{}</td>
</tr>
<tr>
<td>3.14</td>
<td>{ &quot;f&quot;:17 }</td>
</tr>
<tr>
<td>17</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>{ &quot;f&quot;:42, &quot;g&quot;:true }</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Q: What is my “type”?

- **“boolean”**
  - true
  - false

- **“number”**
  - 17
  - 42
  - ...

---

34
function negate(x) {
  if (typeof x === "number") {
    Q: What is my “type”?  
    A: Described by “tag”
  }
}
function negate(x) {
    if (typeof x === "number") {
        return 0 - x;
    } else {
        return !x;
    }
}

"expected args"

values with tag
"number" OR "boolean"
Outline

Why is JavaScript Hard?
  Lightweight Reflection
  Dictionary Objects
  Additional Challenges
obj.f means obj["f"]

Objects are "Dictionaries"
  a.k.a.
  "maps"
  "hash tables"
  "associative arrays"

Objects are that map string keys to values
```javascript
function dispatch(table, op, i, j) {
    method(table)
    method to invoke
}
```
function dispatch(table, op, i, j) {
    var fn = table[op];
    return fn(i, j);
}

var integerOps =
    {
        "+" : function (n, m) { ... },
        "-" : function (n, m) { ... }
    };

dispatch (integerOps, "*", 17, 42);
function dispatch(table, op, i, j) {
    var fn = table[op];
    return fn(i, j);
}
Outline

Why is JavaScript Hard?

- Lightweight Reflection
- Dictionary Objects
- Additional Challenges
Outline

Why is JavaScript Hard?

- Lightweight Reflection
- Dictionary Objects
- Extensible Objects
- Prototype Inheritance
- JavaScript Peculiarities

PROGRAMS
Outline

Why is JavaScript Hard?

- Lightweight Reflection
- Dictionary Objects
- Extensible Objects
- Prototype Inheritance
- JavaScript Peculiarities

Unsupported in Prior Work
Outline

Why is JavaScript Important?
Why is JavaScript Hard?
Static Types via Logic
Security via Logic
Looking Ahead
All JavaScript Values

```
<table>
<thead>
<tr>
<th>&quot;number&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
</tr>
<tr>
<td>3.14</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>42</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```

```
{},   {}  "f":17
...
{}  "f":42, "g":true
```

true    false
### All JavaScript Values

#### Numbers

<table>
<thead>
<tr>
<th>x</th>
<th>tag(x) = “number”</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>3.14</td>
</tr>
<tr>
<td>17</td>
<td>42</td>
</tr>
</tbody>
</table>

#### Booleans

<table>
<thead>
<tr>
<th>x</th>
<th>tag(x) = “boolean”</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

#### Objects

<table>
<thead>
<tr>
<th>x</th>
<th>tag(x) = “object”</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{} “f”:17</td>
</tr>
<tr>
<td>...</td>
<td>{} “f”:42, “g”:true</td>
</tr>
</tbody>
</table>
Why JavaScript?

Security via Logic

Looking Ahead

Types via Logic

---

All JavaScript Values

\[
\{ x \mid \text{integer}(x) \} \\
\{ x \mid x > 0 \} \\
17 \quad 42 \\
\{ \} \quad \{ \text{“f”}:17 \} \\
\quad \ldots \\
17 \quad 42 \quad 3.14 \\
\{ \text{“f”}:42, \text{“g”}:\text{true} \} \\
\]

true \quad false
Refinement Types

\[ \{ x \mid p \} \]

“set of values \( x \) s.t. formula \( p \) is true”
Why JavaScript? | Challenges | Types via Logic | Security via Logic | Looking Ahead

Type 1 \( \subseteq \) Type 2

f(x)

"actual args" \( \subseteq \) "expected args"
Why JavaScript?  

Challenges  

Types via Logic  

Security via Logic  

Looking Ahead

$\{ x \mid p \} \subseteq \{ x \mid q \}$

$f(x) \Rightarrow p \Rightarrow q$
Subtyping is Implication

Source of Expressive Power

\[
\{ x \mid p \} \subseteq \{ x \mid q \}
\]

\[ p \implies q \]
Outline

Static Types via Logic

- Lightweight Reflection
- Dictionary Objects
- Key Technical Innovations
- Evaluation
function negate(x) {
    if (typeof x == "number") {
        return 0 - x;
    } else {
        return !x;
    }
}

NumOrBool =

\{ v \mid \text{tag}(v) = \text{"number"} \lor \text{tag}(v) = \text{"boolean"} \}
//: negate :: (x:NumOrBool) → NumOrBool

function negate(x) {
  if (typeof x == "number") {
    return 0 - x;
  } else {
    return !x;
  }
}
function negate(x) {
    if (typeof x == "number") {
        return 0 - x;
    } else {
        return !x;
    }
}

//: negate :: (x:NumOrBool) \rightarrow NumOrBool

"expected args"

\{ x \mid \text{not}(\text{tag}(x) = "number") \}
\cap \{ x \mid \text{tag}(x) = "number" \lor \text{tag}(x) = "boolean" \}
//: negate :: (x:NumOrBool) → NumOrBool

function negate(x) {
    if (typeof x === "number") {
        return 0 - x;
    } else {
        return !x;
    }
}
## Logic Enables Types to Depend on Values

```javascript
//: negate :: (x:NumOrBool) → { y | tag(y) = tag(x) }

function negate(x) {
  if (typeof x == "number") {
    return 0 - x;
  } else {
    return !x;
  }
}
```
Outline

Static Types via Logic

- Lightweight Reflection
- Dictionary Objects
- Key Technical Innovations
- Evaluation
All JavaScript Values

\{ x | \text{tag}(x) = \text{“object”} \}

\{
\}

\{
\}

\{ “f”: 17 \}

\{ “f”: true \}

\{ “f”: 42, “g”: null \}

\ldots
<table>
<thead>
<tr>
<th>All JavaScript Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{}</code></td>
</tr>
<tr>
<td>`{ x</td>
</tr>
<tr>
<td>{“f”:true}</td>
</tr>
<tr>
<td>{“f”:17}</td>
</tr>
<tr>
<td>{“f”:42, “g”:null}</td>
</tr>
<tr>
<td>…</td>
</tr>
</tbody>
</table>
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

All JavaScript Values

{ x | tag(x) = "object" ∧ 
  has(x, "f") ∧ 
  tag(sel(x, "f")) = "number" }

{ "f": true }

{ "f": true }

{ "f": 17 }

{ "f": 42, "g": null }

...
function dispatch(table, op, i, j) {
    var fn = table[op];
    return fn(i, j);
}
function dispatch(table, op, i, j) {
  var fn = table[op];
  return fn(i, j);
}
var integerOps = { “+” : …, “-” : … };
dispatch (integerOps, “*”, 17, 42);

“actual args”

\[
\{ \text{op} \mid \text{op} = “*” \} \subseteq \{ \text{op} \mid \text{has}(\text{integerOps}, \text{op}) \} \\
\text{op} = “*” \quad \Rightarrow \quad \text{has}(\text{integerOps}, \text{op})
\]

Type Checking Prevents Run-Time Error
Outline

Static Types via Logic

Lightweight Reflection
Dictionary Objects
Key Technical Innovations
Evaluation
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior Refinement Types</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Dependent JavaScript (DJS)**

[POPL ’12, OOPSLA ’12]

- lightweight reflection ✓ ✓ ✓ ✓ ✓
- dictionary objects ✗ ✓ ✓ ✓ ✓ ✓
- extensible objects ✗ ✓ ✓ ✓ ✓ ✓
- prototype inheritance ✗ ✓ ✓ ✓ ✓ ✓
- JavaScript peculiarities ✗ ✓ ✓ ✓ ✓ ✓
Why JavaScript? | Challenges | Types via Logic | Security via Logic | Looking Ahead

- Prior Refinement Types
- Dependent JavaScript (DJS)
  [POPL ’12, OOPSLA ’12]

dictionary objects

Key Challenge:
Function Subtyping as Logical Implication
Why JavaScript?  Challenges  Types via Logic  Security via Logic  Looking Ahead

42 + negate(17)

\{'f | f :: (x:NumOrBool) \to \{ y | \text{tag}(y) = \text{tag}(x) \} \}

\subseteq

\{'f | f :: (x:Num) \to \text{Num} \}'

“expected function”
How to Prove

How to Encode Type System **Inside** Logic?

Prior Refinement Systems
**Disallow** Function Types Inside Logic!
Types via Logic

Types

\[ T ::= \{ x \mid p \} \]
\[ \mid T_1 \rightarrow T_2 \]

Refinement Logic

[SAT + Decidable Theories]

Satisfiability Modulo Theories (SMT)

\[ p \Rightarrow q \]

SMT Solver
(e.g. Z3)
Types

\[ T ::= \{ x \mid p \} \]
\[ \mid T_1 \rightarrow T_2 \]

Refinement Logic

\[ p ::= p \land q \mid p \lor q \mid \neg p \]
\[ \mid x = y \mid x > y \mid \ldots \]
\[ \mid \text{tag}(x) = \text{"number"} \mid \ldots \]
\[ \mid \text{sel}(x, k) = 17 \mid \ldots \]

- Boolean Connectives
- Equality
- Linear Arithmetic
- Uninterpreted Functions
- McCarthy Arrays
**Types**

\[
T ::= \{ x \mid p \} \\
\mid T_1 \rightarrow T_2
\]

**Refinement Logic**

\[
p ::= p \land q \mid p \lor q \mid \neg p \\
\mid x = y \mid x > y \mid \ldots \\
\mid \text{tag}(x) = "number" \mid \ldots \\
\mid \text{sel}(x, k) = 17 \mid \ldots
\]

**Enables Union Types and Lightweight Reflection**

\[
\text{NumOrBool} = \{ v \mid \text{tag}(v) = "number" \lor \text{tag}(v) = "boolean" \}
\]
### Types

\[
T ::= \{ x \mid p \} \quad \mid \quad T_1 \to T_2
\]

### Refinement Logic

\[
p ::= p \land q \quad \mid \quad p \lor q \quad \mid \quad \neg p
\]
\[
\mid \quad x = y \quad \mid \quad x > y \quad \mid \quad \ldots
\]
\[
\mid \quad \text{tag}(x) = \text{“number”} \quad \mid \quad \ldots
\]
\[
\mid \quad \text{sel}(x,k) = 17 \quad \mid \quad \ldots
\]

Enables **Dictionary Types** with Dynamic Keys ...

But **Disallows** Functions in Dictionaries!

\[
\{ d \mid \text{tag(\text{sel}(d,k))} = \text{“boolean”} \land \\
\text{tag(\text{sel}(d,“f”))} = \text{“function”} \land \\
\text{sel}(d,“f”) ???? \}
\]
Types via Logic

Security via Logic

Looking Ahead

Why JavaScript?

Challenges

Types

Refinement Logic

\( T ::= \{ x \mid p \} \)

\( p ::= p \land q \mid p \lor q \mid \neg p \)

\( \mid x = y \mid x > y \mid \ldots \)

\( \mid \text{tag}(x) = \text{"number"} \mid \ldots \)

\( \mid \text{sel}(x,k) = 17 \mid \ldots \)

\( \mid \text{sel}(x,k) :::: T_1 \rightarrow T_2 \mid \ldots \)

Goal: Refer to Type System Inside Logic
Without Giving up Decidability

Solution: Nested Refinements!
Types

\[ T ::= \{ x \mid p \} \]

Refinement Logic

\[ p ::= p \land q \mid p \lor q \mid \neg p \]
\[ \mid x = y \mid x > y \mid \ldots \]
\[ \mid \text{tag}(x) = \text{“number”} \mid \ldots \]
\[ \mid \text{sel}(x,k) = 17 \mid \ldots \]
\[ \mid \text{sel}(x,k) ::: T_1 \rightarrow T_2 \mid \ldots \]

Nested Refinements [POPL ’12]

1. Decidable Encoding
2. Precise Subtyping
3. Type Soundness Proof
Decidable Encoding

\[ f : (x: \text{NumOrBool}) \rightarrow \{ y | \text{tag}(y) = \text{tag}(x) \} \]

\[ \Rightarrow f : (x: \text{Num}) \rightarrow \text{Num} \]

Uninterpreted Predicate in Logic

Distinct Uninterpreted Constants in Logic…
Decidable Encoding

Uninterpreted Predicate in Logic

Distinct Uninterpreted Constants in Logic...

w/ Types Nested Inside

Why JavaScript? Challenges Types via Logic Security via Logic Looking Ahead
Decidable Encoding

\[ f :: (x:\text{NumOrBool}) \rightarrow \{ y | \text{tag}(y) = \text{tag}(x) \} \]

\[ \Rightarrow f :: (x:\text{Num}) \rightarrow \text{Num} \]

SMT Solver

\[ \times \text{ Trivially Decidable, but Imprecise} \]
Precise Subtyping

\[
\{ f \mid p \} \subseteq \{ f \mid f :: (x:T_1) \rightarrow T_2 \}
\]

Step 1: Find \((x:S_1) \rightarrow S_2\) such that

\[ p \Rightarrow f :: (x:S_1) \rightarrow S_2 \]

SMT Solver

Step 2: Compare \((x:S_1) \rightarrow S_2\) and \((x:T_1) \rightarrow T_2\)

“contra-variance”

\[ T_1 \subseteq S_1 \]

“co-variance”

\[ x:T_1 \land S_2 \subseteq T_2 \]
Precise Subtyping

\[ \{ f \mid f = \text{negate} \} \subseteq \{ f \mid f :: (x:\text{Num}) \rightarrow \text{Num} \} \]

Step 1:
\[ f = \text{negate} \]
\[ \Rightarrow f :: (x:\text{NumOrBool}) \rightarrow \{ y \mid \text{tag}(y) = \text{tag}(x) \} \]

Step 2:
Precise Subtyping

\[
\{ f \mid f = \text{negate} \} \subseteq \{ f \mid f :: (x:\text{Num}) \rightarrow \text{Num} \}
\]

Step 1:

\[
f = \text{negate} \\
\Rightarrow f :: (x:\text{NumOrBool}) \rightarrow \{ y | \text{tag}(y) = \text{tag}(x) \}
\]

Step 2:

\[
\text{Num} \subseteq \text{NumOrBool} \\
\text{x:Num} \land \{ y | \text{tag}(y) = \text{tag}(x) \} \subseteq \text{Num}
\]
Precise Subtyping

\[
\{ f \mid f = \text{negate} \} \subseteq \{ f \mid f :: (x:\text{Num}) \to \text{Num} \} 
\]

Step 1: Decidable Reasoning via SMT

Step 2: Precise Function Subtyping via Syntactic Techniques
Type Soundness Proof

Conventional Proofs
Require Logic to be an **Oracle** ...

... but Nesting Introduces **Circularity**
That Prevents Typical Inductive Proofs
Type Soundness Proof

Proof Technique:
**Stratify** into Infinite Hierarchy
Nested Refinements [POPL ’12]
Key to Encode Dictionary Objects
(Even in Statically Typed Languages!)
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightweight reflection</td>
<td>✓</td>
<td>✓ path-sensitivity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dictionary objects</td>
<td>✗</td>
<td>✓ nested refinements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extensible objects</td>
<td>✗</td>
<td>✓ flow-sensitivity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prototype inheritance</td>
<td>✗</td>
<td>✓ logical encoding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaScript peculiarities</td>
<td>✗</td>
<td>✓ logical encoding</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Dependent JavaScript (DJS)**

[POPL ’12, OOPSLA ’12]
Why JavaScript?     Challenges     Types via Logic     Security via Logic     Looking Ahead

JavaScript, Python, Ruby

- lightweight reflection
- dictionary objects
- extensible objects

JavaScript peculiarities

JavaScript

dependent JavaScript (DJS)
[POPL ’12, OOPSLA ’12]

- path-sensitivity
- nested refinements
- flow-sensitivity
- logical encoding
- logical encoding
**Statically Typed Languages**
(Java, C++, OCaml, Haskell, ...)

- lightweight reflection
- dictionary objects
- extensible objects
- prototype inheritance
- JavaScript peculiarities

**Dependent JavaScript (DJS)**
[POPL ’12, OOPSLA ’12]

- path-sensitivity
- nested refinements
- flow-sensitivity
- logical encoding
- logical encoding
Statically Typed Languages
(Java, C++, OCaml, Haskell, ...)

Dependent JavaScript (DJS)
[POPL ’12, OOPSLA ’12]

General Techniques Can Be Used to Eliminate More Errors
✓ path-sensitivity
✓ nested refinements
✓ flow-sensitivity

Why JavaScript?
Challenges
Types via Logic
Security via Logic
Looking Ahead
Outline

Static Types via Logic

- Lightweight Reflection
- Dictionary Objects
- Key Technical Innovations
- Evaluation
### Benchmarks

13 Excerpts from:
- *JavaScript, Good Parts*
- SunSpider Benchmark Suite
- Google Closure Library

#### LOC before/after

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>306</td>
<td>408</td>
</tr>
<tr>
<td>LOC</td>
<td></td>
<td>(+33%)</td>
</tr>
</tbody>
</table>

Chosen to Stretch Expressiveness Limits of DJS
<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>LOC before/after</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Why JavaScript?</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Challenges</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Types via Logic</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Security via Logic</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Looking Ahead</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Benchmarks</strong></td>
<td><strong>LOC before/after</strong></td>
</tr>
<tr>
<td>13 Excerpts from: <em>JavaScript, Good Parts</em> SunSpider Benchmark Suite Google Closure Library</td>
<td>306</td>
</tr>
<tr>
<td>9 Browser Extensions from: [Guha et al. Oakland ’11]</td>
<td>321</td>
</tr>
<tr>
<td>2 Examples from: Google Gadgets</td>
<td>1,003</td>
</tr>
<tr>
<td><strong>TOTALS</strong></td>
<td>1,630</td>
</tr>
</tbody>
</table>
Type Annotations

Relatively Simple **Syntactic Sugar** and **Type Inference** Have Helped Significantly

Opportunities for Improved Type Inference:

- Iterative Predicate Abstraction
- Bootstrap by **Running Test Cases** [STOP ‘11]
- Translating Programs from TypeScript
<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>LOC before/after</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 Excerpts from:</td>
<td>306 (306)</td>
<td>10 sec</td>
</tr>
<tr>
<td><em>JavaScript, Good Parts</em></td>
<td>408 (+33%)</td>
<td></td>
</tr>
<tr>
<td>SunSpider Benchmark Suite</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Google Closure Library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Browser Extensions from:</td>
<td>321 (321)</td>
<td>3 sec</td>
</tr>
<tr>
<td><a href="#">Guha et al. Oakland ’11</a></td>
<td>383 (+19%)</td>
<td></td>
</tr>
<tr>
<td>2 Examples from:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Google Gadgets</td>
<td>1,003 (1,003)</td>
<td>19 sec</td>
</tr>
<tr>
<td>TOTALS</td>
<td>1,630 (+12%)</td>
<td>32 sec</td>
</tr>
</tbody>
</table>
Performance

Relatively Simple Optimizations So Far:
  • Avoid SMT Solver When Possible
  • Reduce Precision for Common Patterns
  • Rely on Nested Refinements When Needed

| TOTALS   | 1,630 | 1,818 (+12%) | 32 sec |
## Outline

### Static Types via Logic

- Lightweight Reflection
- Dictionary Objects
- Key Technical Innovations
- Evaluation
Outline

Why is JavaScript Important?

Why is JavaScript Hard?

Static Types via Logic

Security via Logic

Looking Ahead
Outline

Security via Logic

Browser Extensions
Types for Fine-Grained Policies
... New Yorker article, “The Itch,” ...
ANNALS OF MEDICINE

THE ITCH

Its mysterious power may be a clue to a new theory about brains and bodies.

BY ATUL GAWANDE

JUNE 30, 2008

I t was still shocking to M. how much a few wrong turns could change your life. She had graduated from Boston College with a degree in psychology, married at twenty-five, and had two children, a son and a daughter. She and her family settled in a town on Massachusetts’ southern shore. She worked for thirteen years in health care, becoming the director...
It was still shocking to M. how much a few wrong turns could change your life. She had graduated from Boston College with a degree in psychology, married at twenty-five, and had two children, a son and a daughter. She and her family settled in a town on Massachusetts’ southern shore. She worked for thirteen years in health care, becoming the director of a residence program for men who’d suffered severe head injuries. But she and her husband began fighting. There were betrayals. By the time she was thirty-two, her marriage had disintegrated. In the divorce, she lost possession of their home, and, amid her financial and psychological struggles, she saw that she was losing her children, too. Within a few years, she was drinking. She began dating someone, and they drank together. After a while, he brought some drugs home, and she tried them. The drugs got harder. Eventually, they were doing heroin, which turned to be readily available from a street a block away from her apartment.

Scientists once saw itching as a form of pain. They now believe it to be a different order of sensation. Photograph by Gerald Slosa.
"I want every article in print mode"
Browser Extensions

Third-Party Developers

Customize Browser
for (var elt in doc.getElementsByTagName("a")) {
  var url = elt.getAttribute("href");
  if (url.match(/^newyorker.com/)) {
    elt.setAttribute("href", url + "?printable=true")
  }
}
Document (DOM)

host-site.com

newyorker.com?printable=true

Browser Extensions

PRINT NEW YORKER
Why JavaScript? | Challenges | Types via Logic | Security via Logic | Looking Ahead

Access All Resources

Policy Expressiveness
Access Some Resources

can access DOM
cannot access History
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

---

Policy Expressiveness

coarse-grained and uninformative!

- can access DOM
- cannot access History
Why JavaScript?

Security via Logic

Looking Ahead

Challenges

Types via Logic

Policy Expressiveness

coarse-grained and uninformative!

similar situation for other app platforms
Why JavaScript?  Challenges  Types via Logic  Security via Logic  Looking Ahead

GOAL

Fine-Grained Policies

Policy Expressiveness

App Store

Android

Apple
GOAL

Fine-Grained Policies

can write “a” elements but only “href” attribute when original “href” value starts with “newyorker.com” and is prefix of new value
Language-Based Security

If Language Could Enforce Some Security-Related Policies ...

Then **Reduce** Dependence On:

- Testing
- Code Reviews
- Dynamic Checking
For all programs $p$ in $\text{PROGRAMS}$

If $p$ has a type $T$ in $\text{TYPES}$

Then $p$ does not fail with any error $e$ in $\text{ERRORS}$
For all programs p in PROGRAMS

If p has a type T in TYPES

Then p does not fail with any error e in ERRORS

GOAL: Eliminate Some Security-Related Errors
But How?

**ERRORS**

GOAL: Eliminate Some Security-Related Errors
Subtyping is Implication
Source of Expressive Power

\[
\{ x \mid p \}\subseteq \{ x \mid q \} \Rightarrow p \implies q
\]
Outline

Security via Logic

- Browser Extensions
- Types for Fine-Grained Policies
Why JavaScript?

Security via Logic

Types via Logic

Looking Ahead

Static Verification (DJS)

Developer writes policy in logic!

Security Expert reviews only policy

User reads informative policy
Network Manager

request, ...

JS Engine
eval, ...

DOM
getAttr, setAttr, ...

request ::
url:Str
→ Str
<table>
<thead>
<tr>
<th>Network Manager</th>
<th>JS Engine</th>
<th>DOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>request, ...</td>
<td>eval, ...</td>
<td>getAttr, setAttr, ...</td>
</tr>
</tbody>
</table>

\[
\text{request} :: \{url:\text{Str} | \text{WhiteListed}(url)\} \\
\rightarrow \text{Str}
\]
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Manager request, ...</td>
<td>JS Engine eval, ...</td>
<td>DOM <code>getAttr, setAttr, ...</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
eval :: \quad s : \text{Str} \\
\quad \rightarrow \quad \text{Void}
\]
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Manager request, ...</td>
<td>JS Engine eval, ...</td>
<td>DOM getAttr, setAttr, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{eval} :: \{s:\text{Str} \mid \neg (\text{Tainted}(s))\} \\
\rightarrow \text{Void}
\]
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Manager request, ...</td>
<td>JS Engine eval, ...</td>
<td>DOM <code>getAttribute</code>, <code>setAttr</code>, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{getAttribute} ::
\]
\[
\text{elt:Element} \\
\rightarrow \text{attr:Str} \\
\rightarrow \text{Str}
\]
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Manager</td>
<td>JS Engine</td>
<td>DOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>request, …</td>
<td>eval, …</td>
<td>setAttr, …</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
getAttr ::
elt : Element
  \rightarrow \{ attr : Str | CanRead(elt, attr) \}
  \rightarrow Str
```
setAttr ::

elt:Element
→ attr:Str
→ y:Str
→ Void
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Manager request, ...</td>
<td>JS Engine eval, ...</td>
<td>DOM getAttr, setAttr, ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
setAttr ::
  elt:Element
→ attr:Str
→ {y:Str | CanWrite(elt, attr, y)}
→ Void
```
Refined APIs Mediate Access

Typecheck Extension with Policy
forall url.
    not (url = "evil.com")
⇒ WhiteListed (url)

var s = request("evil.com");

finite blacklist
Why JavaScript?  Challenges  Types via Logic  Security via Logic  Looking Ahead

//: url:Str
var s = request(url);

forall url.
not (url = “evil.com”) ⇒ WhiteListed (url)

might be blacklisted!
For all URL:

\[
\text{not (url = "evil.com")] \Rightarrow \text{WhiteListed (url)}
\]

If (`url` != "evil.com") {
  var `s` = request(`url`);
}

```javascript
if (url != "evil.com") {
  var s = request(url);
}
```
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

- forall url.
  - not (url = “evil.com”) ⇒ WhiteListed (url)

```javascript
if (url != “evil.com”) {
  var s = request(url);
  eval(s);
}
```

`s:Str` might be tainted!
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

### Security via Logic

#### Types via Logic

Looking Ahead

Challenges

if (url != "evil.com") {
    var s = request(url);
    s = sanitize(s);
    eval(s);
}

forall url.
    not (url = "evil.com")
⇒ WhiteListed (url)

STATICALLY VERIFIED!
WhiteListed ("ny.com")
WhiteListed ("golf.com")

\[
\text{forall url.} \quad \text{WhiteListed (url)} \Rightarrow \neg (\text{Tainted (request (url))})
\]

trust that whitelisted servers return *untainted* strings
WhiteListed ("ny.com")
WhiteListed ("golf.com")

forall url.
  WhiteListed (url)
  ⇒ not (Tainted (request (url)))

var s = request("ny.com");
eval(s);

no need to sanitize

STATICALLY VERIFIED!
forall e, oldUrl, newUrl.

ValueOf (e, “href”, oldUrl) ∧
StrPrefix ("newyorker.com", oldUrl) ∧
StrPrefix (oldUrl, newUrl)
⇒ CanWrite (e, “href”, newUrl)

for (var elt in doc.getEltsByTagName("a")) {
var url = elt.getAttr("href");
if (url.match(/^newyorker.com$/)) {
  elt.setAttr("href", url + "?printable=true");
} }

for (var elt in doc.getEltsByTagName("a")) {
var url = elt.getAttr("href");
if (url.match(/^newyorker.com$/)) {
  elt.setAttr("href", url + "?printable=true");
} }

for (var elt in doc.getEltsByTagName("a")) {
var url = elt.getAttr("href");
if (url.match(/^newyorker.com$/)) {
  elt.setAttr("href", url + "?printable=true");
} }
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

ValueOf (elt, "href", url) ∧
StrPrefix ("newyorker.com", url) ∧
StrPrefix (url, url + "?printable=true")
⇒ CanWrite (e, "href", url+"?printable=true")

for (var elt in doc.getEltsByTagName("a")) {
    var url = elt.getAttr("href")
    if (url.match(/^newyorker.com$/)) {
        elt.setAttr("href", url + "?printable=true");
    }
}
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

```
for (var elt in doc.getEltsByTagName("a")) {
    var url = elt.getAttr("href");
    if (url.match(/\^newyorker.com/)) {
        elt.setAttr("href", url + "?printable=true");
    }
}
```
Why JavaScript?

Security via Logic

Types via Logic

Looking Ahead

Challenges

ValueOf (elt, “href”, url) ∨
StrPrefix (“newyorker.com”, url) ∨
StrPrefix (url, url + “?printable=true”) ⇒ CanWrite (e, “href”, url+“?printable=true”)

for (var elt in doc.getEltsByTagName(“a”)) {
    var url = elt.getAttr(“href”);
    if (url.match(/^newyorker.com$/)) {
        elt.setAttr(“href”, url + “?printable=true”);
    }
}
why JavaScript? challenges  types via Logic  security via Logic  looking ahead

ValueOf (elt, “href”, url) ∧
StrPrefix (“newyorker.com”, url) ∧
StrPrefix (url, url + “?printable=true”) ⇒ CanWrite (e, “href”, url + “?printable=true”)

for (var elt in doc.getEltsByTagName(“a”)) {
  var url = elt.getAttr(“href”);
  if (url.match(^newyorker.com$/)) {
    elt.setAttr(“href”, url + “?printable=true”);
  }
}

statically verified!
ValueOf (elt, "href", url) ∧
StrPrefix ("newyorker.com", url) ∧
StrPrefix (url, url + "?printable=true")
⇒ CanWrite (e, "href", url+"?printable=true")

for (var elt in doc.getEltsByTagName("a")) {
    var url = elt.getAttr("href");
    if (url.match(/^newyorker.com$/)) {
        elt.setAttr("href", "evil.com");
    }
}
ValueOf (elt, "href", url) ∧
StrPrefix ("newyorker.com", url) ∧
StrPrefix (url, "evil.com")
⇒ CanWrite (elt, "href", "evil.com")

for (var elt in doc.getEltsByTagName("a")) {
    var url = elt.getAttr("href");
    if (url.match(/^newyorker.com$/)) {
        elt.setAttr("href", "evil.com");
    }
}
ValueOf (elt, “href”, url) ∧
StrPrefix (”newyorker.com”, url) ∧
StrPrefix (url, “evil.com”)
⇒ CanWrite (e, “href”, “evil.com”)

for (var elt in doc.getEltsByTagName(“a”)) {
    var url = elt.getAttr(“href”);
    if (url.match(/^newyorker.com$/)) {
        elt.setAttr(“href”, “evil.com”)
    }
}
Preliminary Benchmarks

Secure Extensions Written in OCaml

Types for Security
+ JavaScript Verification

= Fine-Grained Web Security

[ESOP ’10, Guha et al. ’11]
[POPL ’12, OOPSLA ’12]
[current]
Preliminary Benchmarks

Secure Extensions Written in OCaml

Ported to DJS (~400 LOC so far)

Well-typed programs don’t have run-time errors and conform to security policies
Outline

Security via Logic

Browser Extensions
Types for Fine-Grained Policies
Outline

Why is JavaScript Important?
Why is JavaScript Hard?
Static Types via Logic
Security via Logic
Looking Ahead

Why JavaScript? Challenges Types via Logic Security via Logic Looking Ahead

PROGRAMS TYPES ERRORS

149
Web Platform

Dependent JavaScript

Scripting Languages

Untrusted Code

Policy Specification
Developer must **write** policy

Security Expert
and
User must **read** policy
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
</table>

- Developer must **write** policy
  - Infer from traces
  - Incentives from platform

- Security Expert must **read** policy
  - Modeling tools to analyze policies

- User must **read** policy
  - Traces to demonstrate behavior
  - Community review-and-recommend services
Web Platform

- Dependent JavaScript
- Untrusted Code
- Scripting Languages
- Policy Specification
- Dynamic Enforcement
Why JavaScript?

Types via Logic

Security via Logic

Looking Ahead

Static Verification (DJS)

Developer

Security Expert

User
Impossible to **statically** analyze code you don’t have!
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

Static Verification (DJS)

Developer

Security Expert

User

Dynamic Enforcement
Why JavaScript?

Types via Logic

Security via Logic

Looking Ahead

... eval("...");

//: #assume

Old Types

New Types

Dynamic Enforcement

New Types
Limits of Static Reasoning

“actual”  “expected”

\[ p \implies \checkmark q \]  Definitely Safe

\[ p \implies \times q \]  Maybe Unsafe
   (But Maybe Safe...)
Limits of Static Reasoning

"actual"  "expected"

\[ p \implies \checkmark q \] Definitely Safe

\[ p \implies X q \] Maybe Unsafe

\[ p \implies \checkmark \neg q \] Definitely Unsafe
Limits of Static Reasoning

• Accept Program, Rather than Reject

• Rely on **Programmer-Truste**d or Dynamically-Checked Assumption

\[ p \implies \neg q \]

**Dynamic Enforcement**
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
</table>

### Web Platform

- **Untrusted Code**
- **Scripting Languages**
- **Dependent JavaScript**
- **Policy Specification**
- **Dynamic Enforcement**
- **Trust Base**
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

Developer

Static Verification (DJS)

Security Expert reviews only policy

User

Dynamic Enforcement
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

Static Verification (DJS)

trust

Security Expert reviews *only* policy

trust

SMT Solver
Eliminate from TCB via Formal Verification

Static Verification (DJS) → trust

Security Expert

trust → SMT Solver → Eliminate from TCB à la Proof-Carrying Code [PLDI ’10]
Tools for Web Development

- Generate Tests and Find Bugs for **Unannotated Programs**
- Support Refactoring and Code Completion for **Annotated Programs**
- Provide Analysis and Documentation Tools for Large, **Popular Libraries** like jQuery
- Web-Based Interfaces to **Collect Statistics** about Programming Patterns, Errors
Inferred Annotations are in Comments

Flow-Sensitive Type Information by Hovering over Program Point
Why JavaScript?

Challenges

Types via Logic

Security via Logic

Looking Ahead

Inferred Annotations are in Comments

Flow-Sensitive Type Information by Hovering over Program Point

Many Software Engineering Challenges!
## Tools for Education

**Learning Complex Languages is Difficult**

**Static Error Messages can be Mysterious**

**Generate Concrete Run-Time Traces?**

- Execution may be easier to understand
- Help motivate compile-time guarantees
Tools for Education

Learning in Large Classes is Difficult

Clickers and Peer Instruction Can Help

Analyze Live In-Class Performance?

• To guide subsequent topics during class
• To provide personalized feedback after class
• To suggest well-balanced study groups
<table>
<thead>
<tr>
<th>Why JavaScript?</th>
<th>Challenges</th>
<th>Types via Logic</th>
<th>Security via Logic</th>
<th>Looking Ahead</th>
</tr>
</thead>
</table>

### Why JavaScript?
- Dependent JavaScript
- Tool Support
- Scripting Languages

### Challenges
- Policy Specification
- Dynamic Enforcement
- Trust Base

### Types via Logic
- Untrusted Code

### Security via Logic

### Looking Ahead
- Web Platform
PL Techniques

Program Analysis for Security
- [PLDI ’09]
- [ESOP ’10]
- [PLDI ’10]
- [STOP ’11]
- [POPL ’12]
- [OOPSLA ’12]
- [ML ’13]

Dataflow Analysis for Race Detection
- [PLDI ’08]
Thanks!

Dynamic Languages are Pervasive

DJS: Precise Static Types via Logic

Towards Fine-Grained Web Security