TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript

Michael Pradel[1], Parker Schuh[2], Koushik Sen[2]

[1] TU Darmstadt, [2] UC Berkeley

Motivation

- JavaScript: Dynamic and permissive
- Problems remain unnoticed
- Purely static analysis is limited

Motivation

- JavaScript: Dynamic and permissive
- Problems remain unnoticed
- Purely static analysis is limited

This talk: Mostly dynamic analysis to find otherwise missed errors

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});                          // 23
addWrapped({v:20}, new Wrapper(3));          // 23
addWrapped({v:"18"}, new Wrapper(5));        // "185"
```

Incorrect behavior, but no obvious sign of misbehavior

Observations

1) Most code follows implicit type rules

- A single type per variable
- A single type per object property
- Functions have fixed signatures

2) Many bugs are violations of these rules

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

`x.v` has types `number` and `string`

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Returns both `number` and `string`

This Talk: TypeDevil

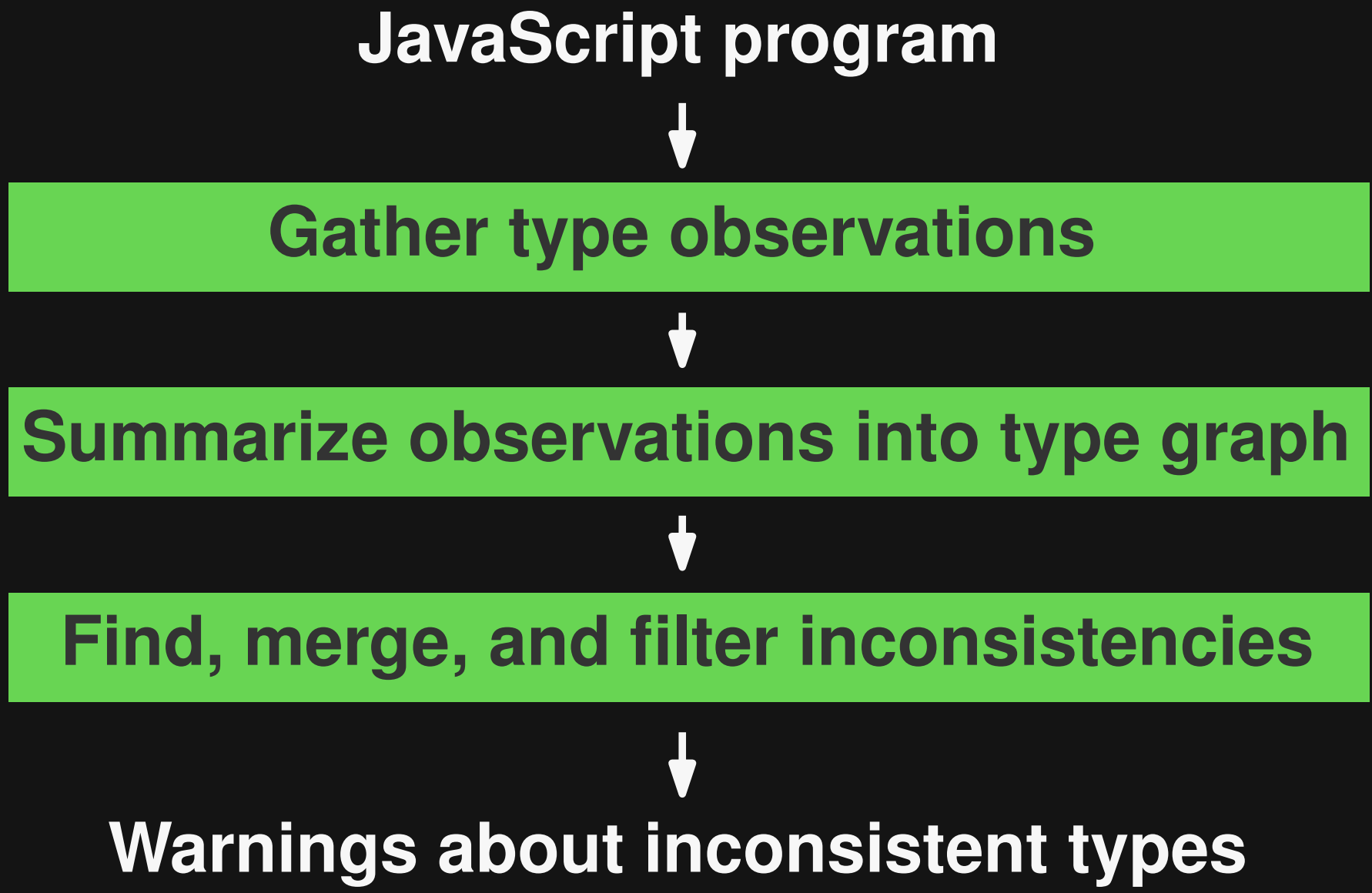
Find **inconsistent types** of

- local variables
- properties of objects
- function signatures

Challenges:

- No static type information
- No nominal types
- Intended polymorphism

Overview


JavaScript program
Gather type observations
Summarize observations into type graph
Find, merge, and filter inconsistencies
Warnings about inconsistent types

Types

Type = Primitive type or record type

Property names $\rightarrow$ Sets of types

Record types represent:

- Object types
- Array types
- Function types

  ”this” and ”return” as properties
- Frame types

  Local variables as properties

Gather Type Observations

One type per allocation site and function definition site

- Store unique name as shadow value

Gather type observations through dynamic analysis

- Observation = (base, property, type)

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

`object1` has property `v` of type `number`

Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

`addWrapped` has local variable `y` of type `undefined`

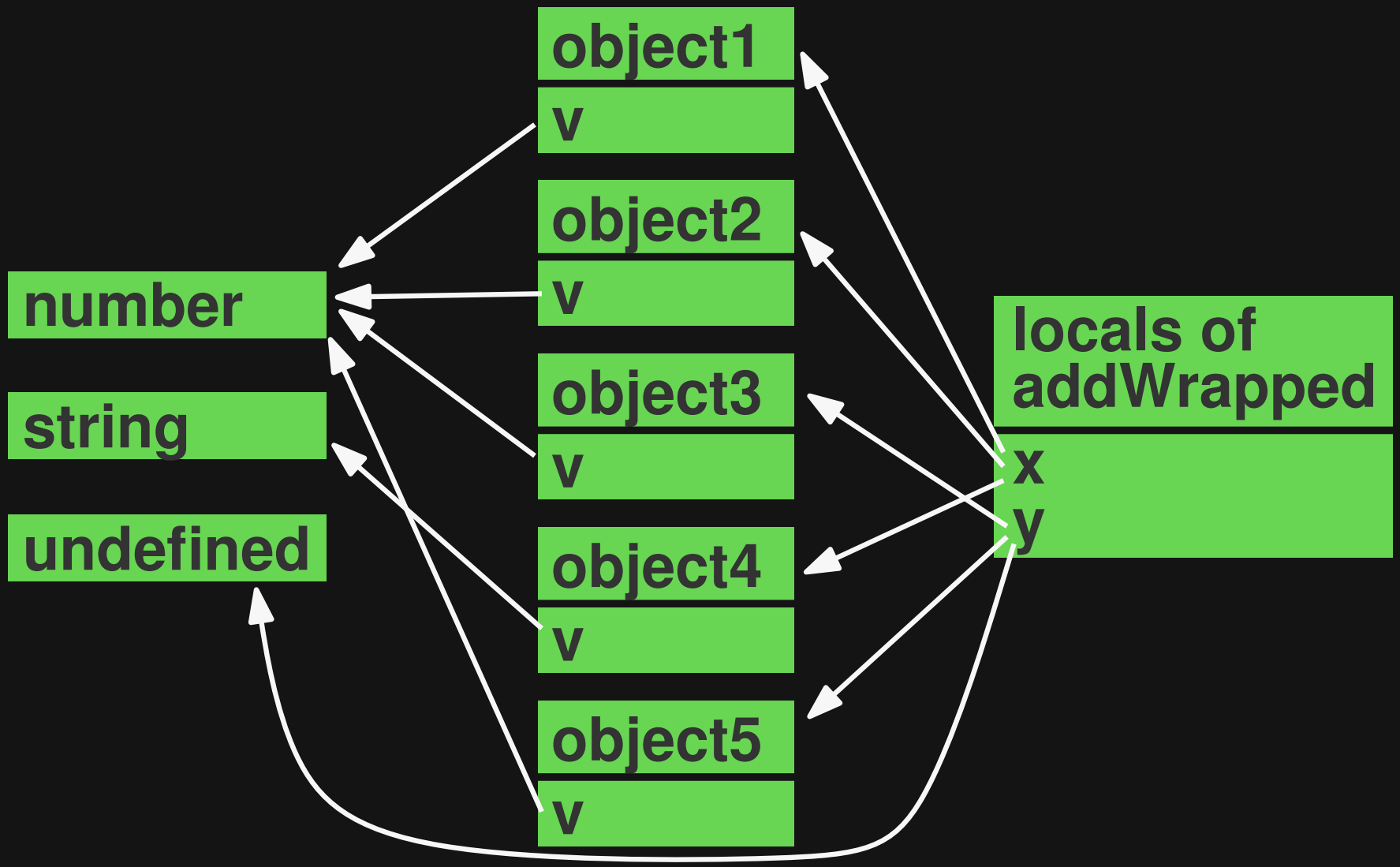
Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

function `addWrapped`
returns `number`

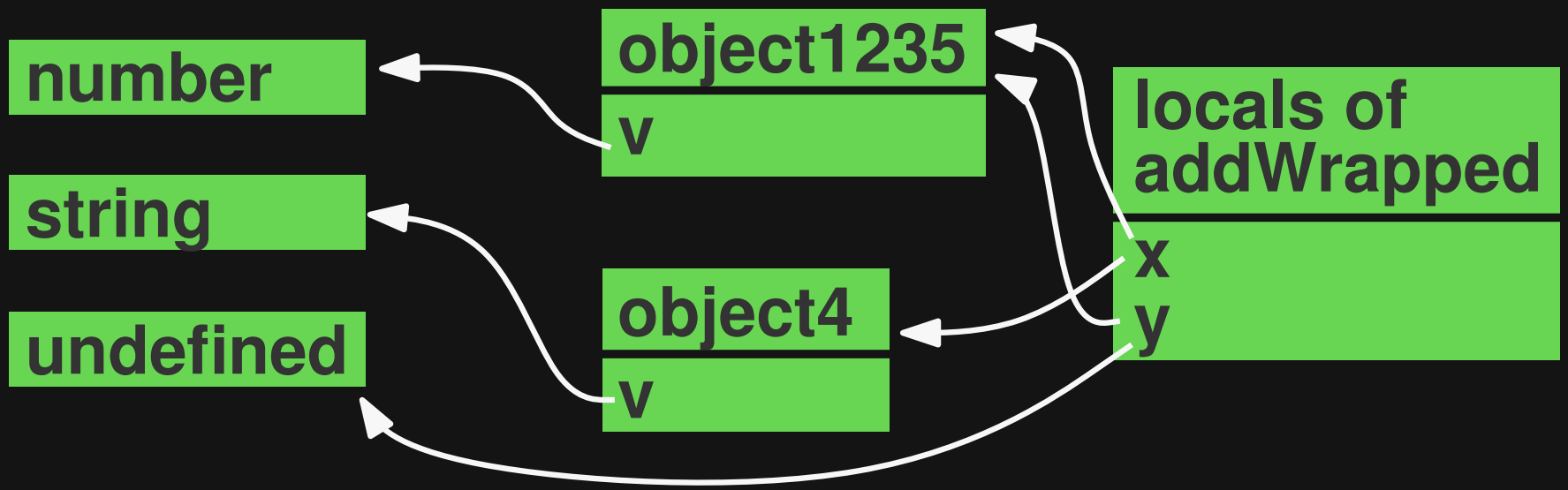
Type Graph

Nodes = type, edges = properties



Condensed Type Graph

Summarize graph by merging equivalent types

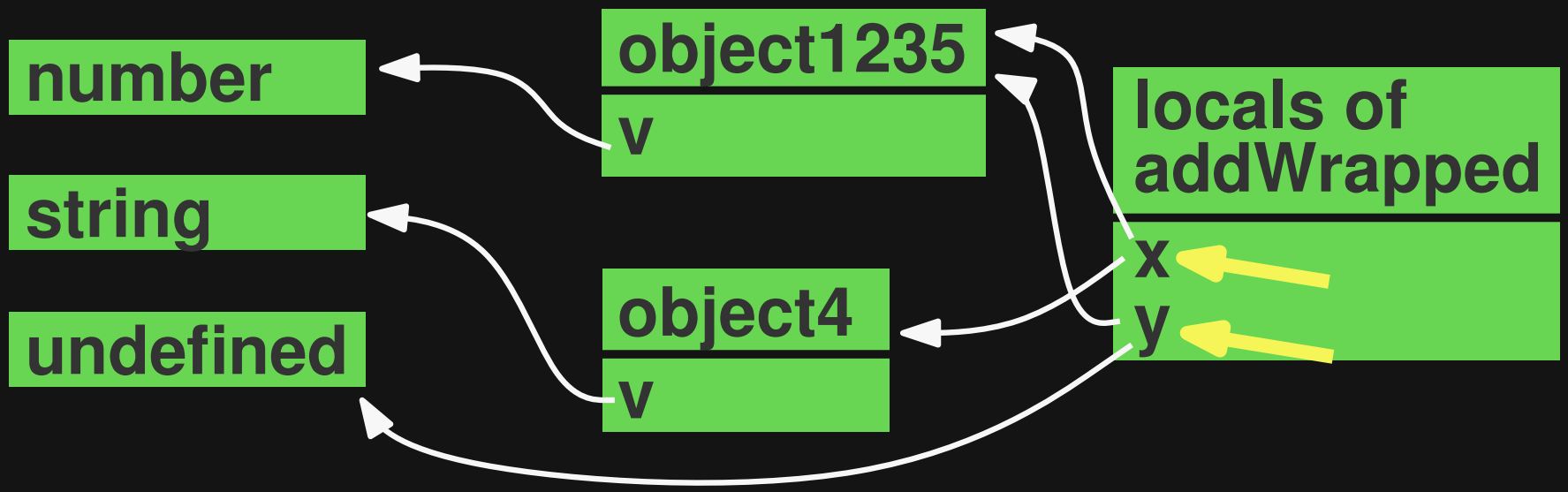


Reporting Inconsistencies

Report nodes with multiple outgoing edges for the same property

Reporting Inconsistencies

Report nodes with **multiple outgoing edges** for the same property



Inconsistencies: Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

Inconsistencies: Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

`x.v` has types `number` and `string`

Inconsistencies: Example

```
function addWrapped(x, y) {
  if (y) return x.v + y.v;
  else return x.v;
}
function Wrapper(v) {
  this.v = v;
}
addWrapped({v:23});
addWrapped({v:20}, new Wrapper(3));
addWrapped({v:"18"}, new Wrapper(5));
```

y has types object and undefined

Prune and Merge Warnings

Prune likely false positives:

- Via belief analysis
- By degree of inconsistency
- By size of type diff
- Structural subtypes
- `null`-related

Merge warnings with same root cause:

- By dataflow relations
- By type diff
- By array type

Prune and Merge Warnings

Prune likely false positives:

- Via belief analysis
- By degree of inconsistency
- By size of type diff
- Structural subtypes
- `null`-related

Merge warnings with same root cause:

- By dataflow relations
- By type diff
- By array type

Prune via Belief Analysis

Problem:

Intended polymorphism

Prune via Belief Analysis

Problem:

Intended polymorphism

```
function BigInteger(a, b, c) {
  if (a != null)
    if ('number' == typeof a)
      this.fromNumber(a, b, c);
    else if (b == null && 'string' != typeof a)
      this.fromString(a, 256);
    else
      this.fromString(a, b);
}
```

Code from Octane's crypto benchmark

Prune via Belief Analysis

Problem:

Intended polymorphism

Naive approach: Warnings about inconsistent argument types

```
function BigInteger(a, b, c) {
  if (a != null)
    if ('number' == typeof a)
      this.fromNumber(a, b, c);
    else if (b == null && 'string' != typeof a)
      this.fromString(a, 256);
    else
      this.fromString(a, b);
}
```

Code from Octane's crypto benchmark

Prune via Belief Analysis

Approach:

- Infer programmer beliefs from code
- Omit warnings about expected types

Prune via Belief Analysis

Approach:

- Infer programmer beliefs from code
- Omit warnings about expected types

```
function BigInteger(a, b, c) {
  if (a != null)
    if ('number' == typeof a)
      this.fromNumber(a, b, c);
    else if (b == null && 'string' != typeof a)
      this.fromString(a, 256);
    else
      this.fromString(a, b);
}
```

Code from Octane's crypto benchmark

Prune via Belief Analysis

Approach:

- Infer **programmer beliefs** from code
- Omit warnings about **expected types**

```
function BigInteger(a, b, c) {
  if (a != null)
    if ('number' == typeof a)
      this.fromNumber(a, b, c);
    else if (b == null && 'string' != typeof a)
      this.fromString(a, 256);
    else
      this.fromString(a, b);
}
```

a may be undefined or null

Code from Octane's crypto benchmark

Prune via Belief Analysis

Approach:

- Infer programmer beliefs from code
- Omit warnings about expected types

```
function BigInteger(a, b, c) {
  if (a != null)
    if ('number' == typeof a)
      this.fromNumber(a, b, c);
    else if (b == null && 'string' != typeof a)
      this.fromString(a, 256);
    else
      this.fromString(a, b);
}
```

a may be number

Code from Octane's crypto benchmark

Prune via Belief Analysis

Approach:

- Infer programmer beliefs from code
- Omit warnings about expected types

```
function BigInteger(a, b, c) {
  if (a != null)
    if ('number' == typeof a)
      this.fromNumber(a, b, c);
    else if (b == null && 'string' != typeof a)
      this.fromString(a, 256);
    else
      this.fromString(a, b);
}
```

Refined approach:

No warning

Code from Octane's crypto benchmark

Merge by Dataflow Relations

Problem: **Multiple references** may refer to a **single value**

Merge by Dataflow Relations

Problem: **Multiple references** may refer to a **single value**

```
function f(x) {
  return g(x);
}
function g(a) {
  return a;
}
f(23);
f({p:"abc"});
```

Merge by Dataflow Relations

Problem: **Multiple references** may refer to a **single value**

```
function f(x) {
  return g(x);
}
function g(a) {
  return a;
}
f(23);
f({p:"abc"});
```

Variable `x`

Merge by Dataflow Relations

Problem: **Multiple references** may refer to a **single value**

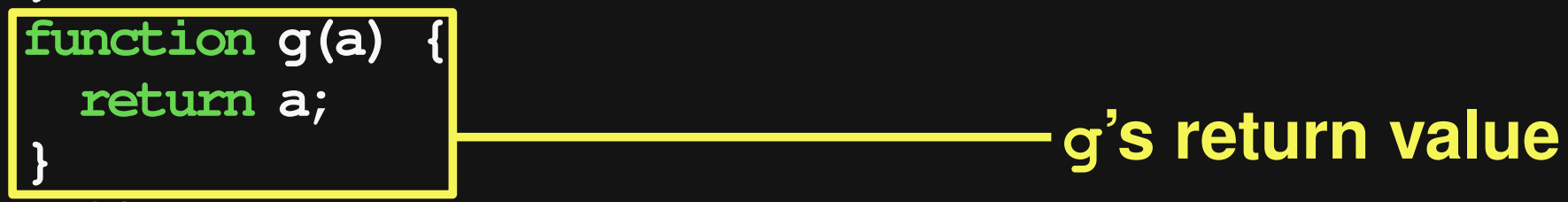
```
function f(x) {
  return g(x);
}
function g(a) {
  return a;
}
f(23);
f({p:"abc"});
```

Variable `a`

Merge by Dataflow Relations

Problem: Multiple references may refer to a single value

```
function f(x) {
  return g(x);
}
function g(a) {
  return a;
}
f(23);
f({p:"abc"});
```



Merge by Dataflow Relations

Problem: **Multiple references** may refer to a **single value**

```
function f(x) {
  return g(x);
}
function g(a) {
  return a;
}
f(23);
f({p:"abc"});
```

Merge by Dataflow Relations

Problem: **Multiple references** may refer to a **single value**

Merge by Dataflow Relations

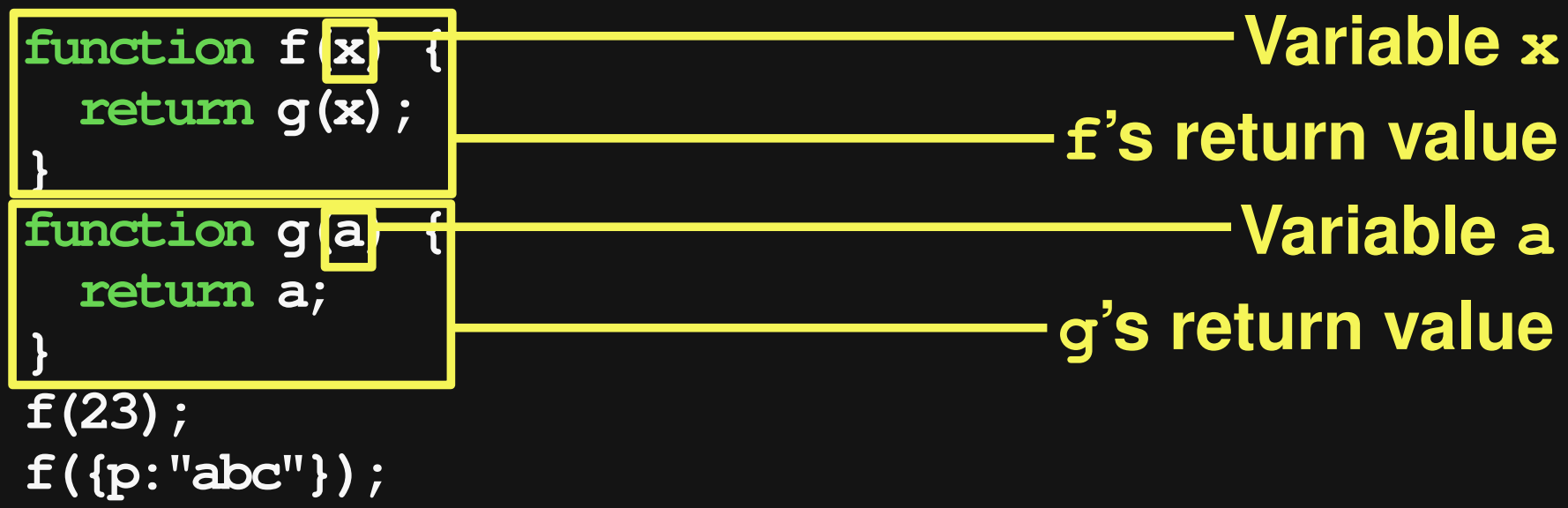
Approach:

- Approximate dataflow via call graph
- Merge warnings that may refer to the same value

Merge by Dataflow Relations

Approach:

- Approximate **dataflow** via call graph
- **Merge** warnings that may refer to the **same value**



Merge by Dataflow Relations

Approach:

- Approximate dataflow via call graph
- Merge warnings that may refer to the same value

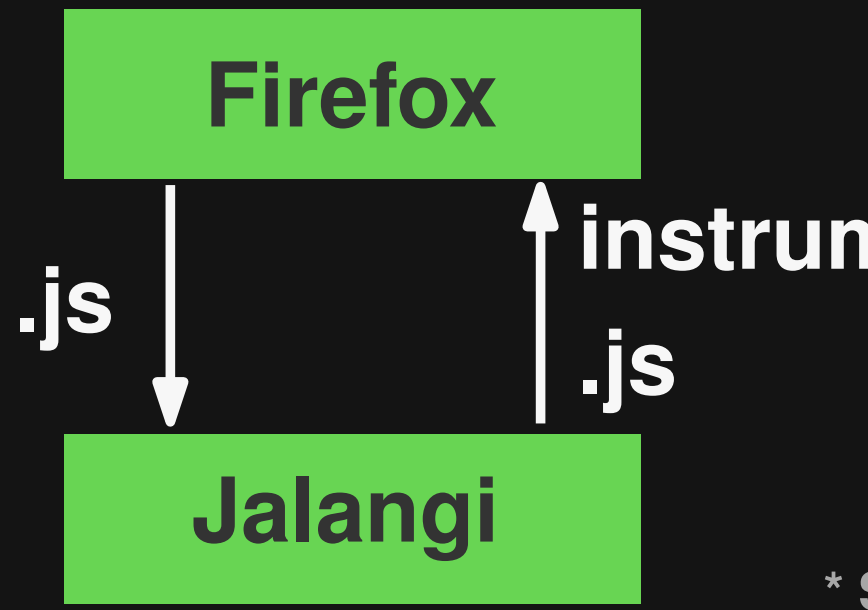
```
function f(x) {
  return g(x);
}
function g(a) {
  return a;
}
f(23);
f({p:"abc"});
```

Refined approach: 1 warning

Implementation

Instrumentation-based implementation on top of Jalangi *

- Hooks into execution
- Browser + node.js



* Sen et al., 2013

Evaluation

Setup:

- Sunspider, Octane, and 7 web apps

Main results:

- **Finds relevant problems:**
  33 warnings, 15 are relevant
- **Pruning and merging is crucial:**
  578 warnings $\rightarrow$ 33 warnings

Example

SunSpider's regexp-dna:

```
var dnaOutputStr;
for (i in seqs) {
  dnaOutputStr += seqs[i].source;
}
```

Example

SunSpider's regexp-dna:

```
var dnaOutputStr;
for (i in seqs) {
  dnaOutputStr += seqs[i].source;
}
```

`string` and `undefined`

Problem: Incorrect string value "undefinedGTAGG..."

Other Problems

Correctness problems

- Crash when dereferencing `undefined`

Performance problems

- Arrays with ”holes”

Dangerous coding practices

- Variable `string` sometimes holds a number

Related Work

Type inference and checking Thiemann 2005, Jensen 2009, Guha 2011, Heidegger 2010

→ **Many false positives**

Type inference by JIT compilers

Logozzo 2010, Hackett 2012, Rastogi 2012

→ **Speculative optimizations**

Type annotations TypeScript

→ **Manual effort**

Conclusion

Find **inconsistent types** in JavaScript:

- Observe types and summarize them into type graph
- Deal with intended polymorphism

Dynamic analysis:
Powerful alternative to static checking

Conclusion

Find **inconsistent types** in JavaScript:

- Observe types and summarize them into type graph
- Deal with intended polymorphism

Dynamic analysis:
Powerful alternative to static checking

Thanks!