TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript

Michael Pradel, Parker Schuh, Koushik Sen
University of California, 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
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

```javascript
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

```javascript
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

$x.v$ has types number and string

```javascript
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

```javascript
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
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));

Goal: Program analysis to find inconsistent types of variables, properties, and functions
TypeDevil: 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 → Sets of types

**Record types** represent:

- Object types
- Array types
- Function types
  - "this" and "return" as properties
- Frame types
  - Local variables as properties
Type Observations

Type observation = (base, property, type)

Gather observations through dynamic analysis *

- Store type names of objects as shadow values
- Add type observations to global set

* Based on Jalangi (Sen et al., 2013)
Gathering Type Observations

Object literal

- Store type name as shadow
- One observation per property
Gathering Type Observations

Object literal

- Store type name as shadow
- One observation per property

```javascript
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));
```
Gathering Type Observations

Function call

- Observations for "this" and "return"
Gathering Type Observations

Function call

- Observations for "this" and "return"

```javascript
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));
```
Gathering Type Observations

Variable read/write

- Observation for type of local variable
  (property of frame type)
Gathering Type Observations

Variable read/write

- Observation for type of local variable (property of frame type)

```javascript
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));
```
TypeDevil: Overview

1. JavaScript program
2. Gather type observations
3. Summarize observations into type graph
4. Find, merge, and filter inconsistencies
5. Warnings about inconsistent types
Type Graph

Nodes = type, edges = properties
Condensed Type Graph

Summarize graph by merging equivalent types
TypeDevil: Overview

JavaScript program

Gather type observations

Summarize observations into type graph

Find, merge, and filter inconsistencies

Warnings about inconsistent 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

```javascript
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(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

```javascript
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
Problem:
Intended polymorphism

```javascript
function BigInteger(a, b, c) {
    if (a != null)
        if ('number' == typeof a)
            this.fromCharCode(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
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);
}
```

Naive approach:
Warnings about inconsistent argument types

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**

```javascript
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**

```javascript
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**

```javascript
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**

```javascript
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);
}
```

$b$ 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**

```javascript
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 string*

Code from Octane’s crypto benchmark
Prune via Belief Analysis

Approach:

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

```javascript
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

```javascript
function f(x) {
    return g(x);
}
function g(a) {
    return a;
}
f(23);
f({p:"abc"});
```
Problem: **Multiple references may refer to a single value**

```javascript
function f(x) {
    return g(x);
}
function g(a) {
    return a;
}
f(23);
f({p:"abc"});
```
Problem: **Multiple references may refer to a single value**

```javascript
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**

```javascript
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**

```javascript
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

Naive approach: 4 warnings
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

```javascript
function f(x) {
  return g(x);
}

function g(a) {
  return a;
}

f(23);
f({p:"abc"});
```
Merge by Dataflow Relations

Approach:

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

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

Refined approach: 1 warning
TypeDevil: Overview

JavaScript program

Gather type observations

Summarize observations into type graph

Find, merge, and filter inconsistencies

Warnings about inconsistent types
Evaluation

Setup:
- Sunspider, Octane, and 7 web apps

Main results:
- Finds relevant problems:
  33 warnings, 15 are relevant
- Merging and pruning is crucial:
  578 warnings $\rightarrow$ 33 warnings
Examples

SunSpider’s regexp-dna:

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

SunSpider’s regexp-dna:

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

Problem: Incorrect string value
"undefinedGTAGG..."
Examples (2)

SunSpider’s date-format-xparb:

```javascript
String.leftPad = function (val, size, ch) {
    var result = new String(val);
    if (ch == null)
        ch = " ";
    while (result.length < size) {
        result = ch + result;
    }
    return result;
}
```
String.leftPad = function (val, size, ch) {
    var result = new String(val);
    if (ch == null)
        ch = " ";
    while (result.length < size) {
        result = ch + result;
    }
    return result;
}

Problem: string and String behave differently

SunSpider’s date-format-xparb:

Returns string and object
Examples (3)

Octane’s GB emulator:

```javascript
function GameBoyCanvas() {
    this.width = 160;
    this.height = 144;
}

function initNewCanvas() {
    gb.canvas.width = gb.canvas.clientWidth;
    gb.canvas.height = gb.canvas.clientHeight;
}
```
Octane’s GB emulator:

```javascript
function GameBoyCanvas() {
    this.width = 160;
    this.height = 144;
}

function initNewCanvas() {
    gb.canvas.width = gb.canvas.clientWidth;
    gb.canvas.height = gb.canvas.clientHeight;
}
```

Problem: Crash when changing the emulator settings
Conclusion

Find **inconsistent types** in JavaScript:
- Observe types and summarize them into type graph
- Deal with intended polymorphism

**Dynamic analysis:**
Powerful approach to find bugs early

https://github.com/Berkeley-Correctness-Group/Jalangi-Berkeley
TypeDevil:
Dynamic Type Inconsistency Analysis for JavaScript

Thank you!

Michael Pradel, Parker Schuh, Koushik Sen
University of California, Berkeley