# Type Refinement for Static Analysis of JavaScript

Vineeth Kashyap [2], **John Sarracino** [1], John Wagner [2],
Ben Wiedermann [1], and Ben Hardekopf [2]

[1]Harvey Mudd College

[2]University of California, Santa Barbara

December 21, 2013

# Conditionals lose precision

```
1
2 if ( object === undefined ) {
3   write( undefined );
4 } else {
5   write( object.foo );
6 }
```

# Conditionals lose precision

```
1 // suppose object ↦ {undefined, {foo : ..., ...}}
2 if ( object === undefined ) {
3   write( undefined );
4 } else {
5   write( object.foo );
6 }
```

# Conditionals lose precision

```
1 // suppose object ↦ {undefined, {foo : ..., ...}}
2 if ( object === undefined ) {
3   write( undefined ); // object ↦ {undefined, {foo : ..., ...}}
4 } else {
5   write( object.foo );
6 }
```

# Conditionals lose precision

```
1 // suppose object ↦ {undefined, {foo : ..., ...}}
2 if ( object === undefined ) {
3   write( undefined ); // object ↦ {undefined, {foo : ..., ...}}
4 } else {
5   write( object.foo ); // object ↦ {undefined, {foo : ..., ...}}
6 }
```

# Refinement regains precision

```
1 // suppose object ↦ {undefined, {foo : ..., ...}}
2 if ( object === undefined ) {
3   write( undefined ); // object ↦ {undefined, {foo : ..., ...}}
4 } else {
5   write( object.foo ); // object ↦ {undefined, {foo : ..., ...}}
6 }
```

# Refinement regains precision

```
1 // suppose object ↦ {undefined, {foo : ..., ...}}
2 if ( object === undefined ) {
3   write( undefined ); // object ↦ {undefined, {foo : ..., ...}}
4 } else {
5   write( object.foo ); // object ↦ {undefined, {foo : ..., ...}}
6 }
```

# Refinement regains precision

```
1 // suppose object ↦ {undefined, {foo : ..., ...}}
2 if ( object === undefined ) {
3   write( undefined ); // object ↦ {undefined, {foo : ..., ...}}
4 } else {
5   write( object.foo ); // object ↦ {undefined, {foo : ..., ...}}
6 }
```

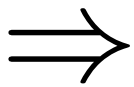
# A (relatively simple) implicit branch

```
1 f1.init();
2 f2.init();
```

# Our IR (notJS) makes implicit branches explicit

```
1 f1.init();

2 f2.init();
```



```
if f1 is null or undefined then
    TypeError
else
    if init is null or undefined then
        TypeError
    else
        if init is not an Object then
            TypeError
        else
            if init is not callable then
                TypeError
            else
                f1.init()
                if f2 is null or undefined then
                    TypeError
                else
                    if init is null or undefined then
                        TypeError
                    else
                        if init is not an Object then
                            TypeError
                        else
                            if init is not callable then
                                TypeError
                            else
                                f2.init()
                            end if
                        end if
                    end if
                end if
            end if
        end if
    end if
end if
```

# What branches to refine?

**Types of branches:**

- *Explicit* branches are visible in JavaScript source code (i.e., if $e_1$ $e_2$ else $e_3$).
- *Implicit* branches are generated by JavaScript semantics (e.g. TypeError generation).

# What branches to refine?

(previous work)

**Types of branches:**

- *Explicit* branches are visible in JavaScript source code (i.e., if $e_1$ $e_2$ else $e_3$).
- *Implicit* branches are generated by JavaScript semantics (e.g. TypeError generation).

*S. Tobin-Hochstadt and M. Felleisen. The design and implementation of typed scheme. In ACM SIGPLAN Symposium on Principles of Programming Languages (POPL), 2008.*

*S. Tobin-Hochstadt and M. Felleisen. Logical types for untyped languages. In ACM SIGPLAN International Conference on Functional programming (ICFP), 2010.*

*Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. Typing local control and state using flow analysis. In ESOP'11/ETAPS'11, 2011.*

# What branches to refine?

## (key insight)

**Types of branches:**

- *Explicit* branches are visible in JavaScript source code (i.e., if $e_1$ $e_2$ else $e_3$).
- *Implicit* branches are generated by JavaScript semantics (e.g. TypeError generation).

# Our contribution

**We introduce:**

- Several novel refinement heuristics over implicit branchpoints.
- An empirical evaluation of current work and our heuristics.
- Recommendations for future usage.

# Implemented refinement heuristics

```
1: if e then
2:     ...
3: else
4:     ...
5: end if
```

- **Type:** existing explicit `typeof` refinement heuristic.
- **Undef:** refines over field access implicit branch.
- **Func:** refines over function invocation implicit branch.
- **Prim:** refines over type conversion implicit branch.

# Implicit branch examples

## (object field access)

```
1 x.foo;
2       ...
```

1: **if** x is undefined or null **then**
2: TypeError
3: **else**
4: ...
5: **end if**

# Implicit branch examples

## (function invocation)

```
1 x();
2       ...
```

1: **if** x is callable **then**
2: invoke x
3: ...
4: **else**
5: TypeError
6: **end if**

# Implicit branch examples

## (type conversion)

```
1 var y = x + 3;
2       ...
```

1: **if** x is an object **then**
2: $\quad y \leftarrow x.valueOf() + 3$
3: $\quad$ ...
4: **else**
5: $\quad y \leftarrow x + 3$
6: $\quad$ ...
7: **end if**

# Precision evaluation

**Flow-, context-sensitive TypeError analysis:**

- without refinement
- with "typeof" refinement
- with "typeof", primitive refinement
- with "typeof", primitive, function, undef/null refinement

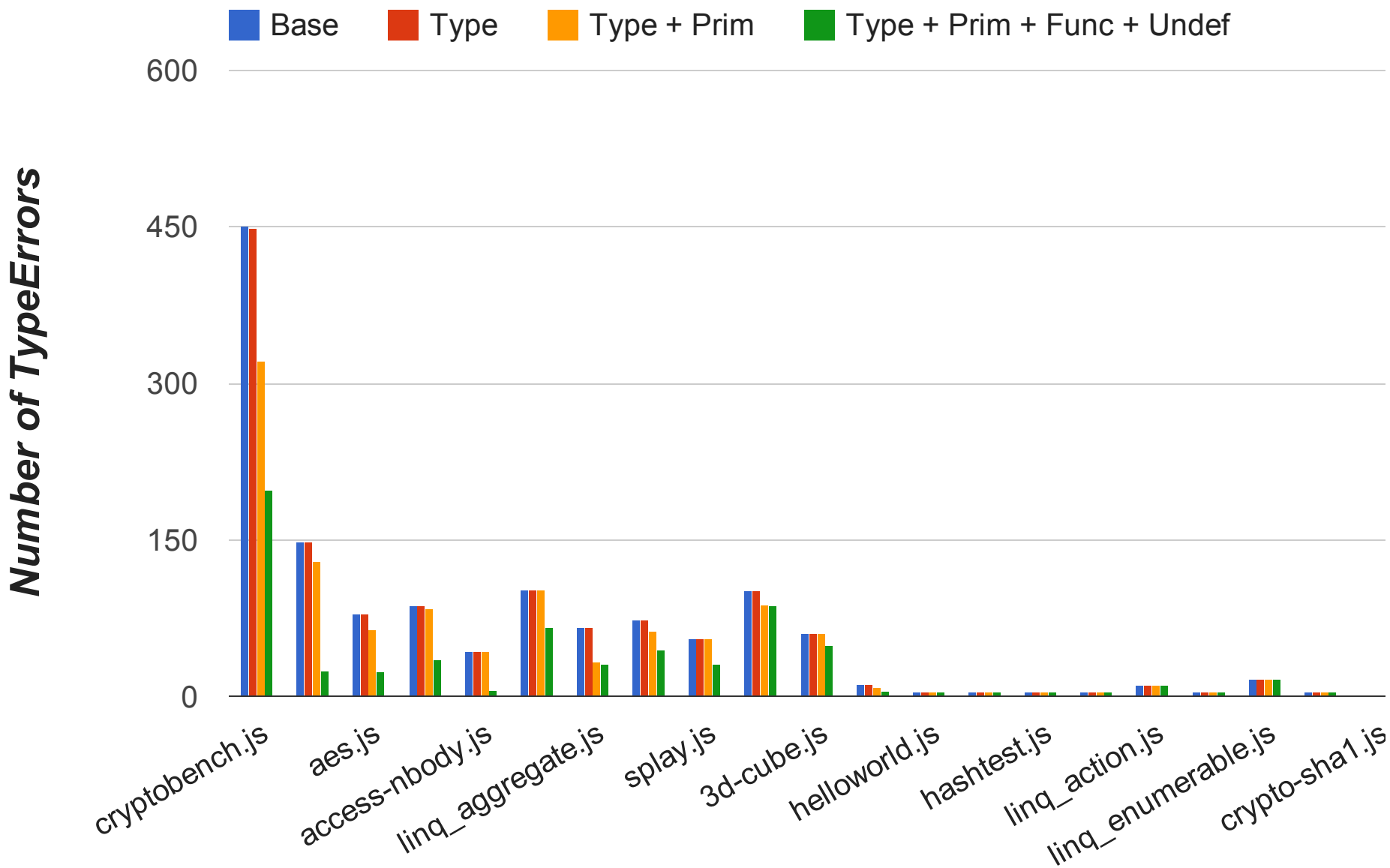
# Benchmark suites

**Data collected on:**

- SunSpider/Octane benchmark suites.
- Open source code from LINQ for JavaScript, Defensive JS.
- C/C++ code translated using Emscripten LLVM $\rightarrow$ JavaScript compiler.

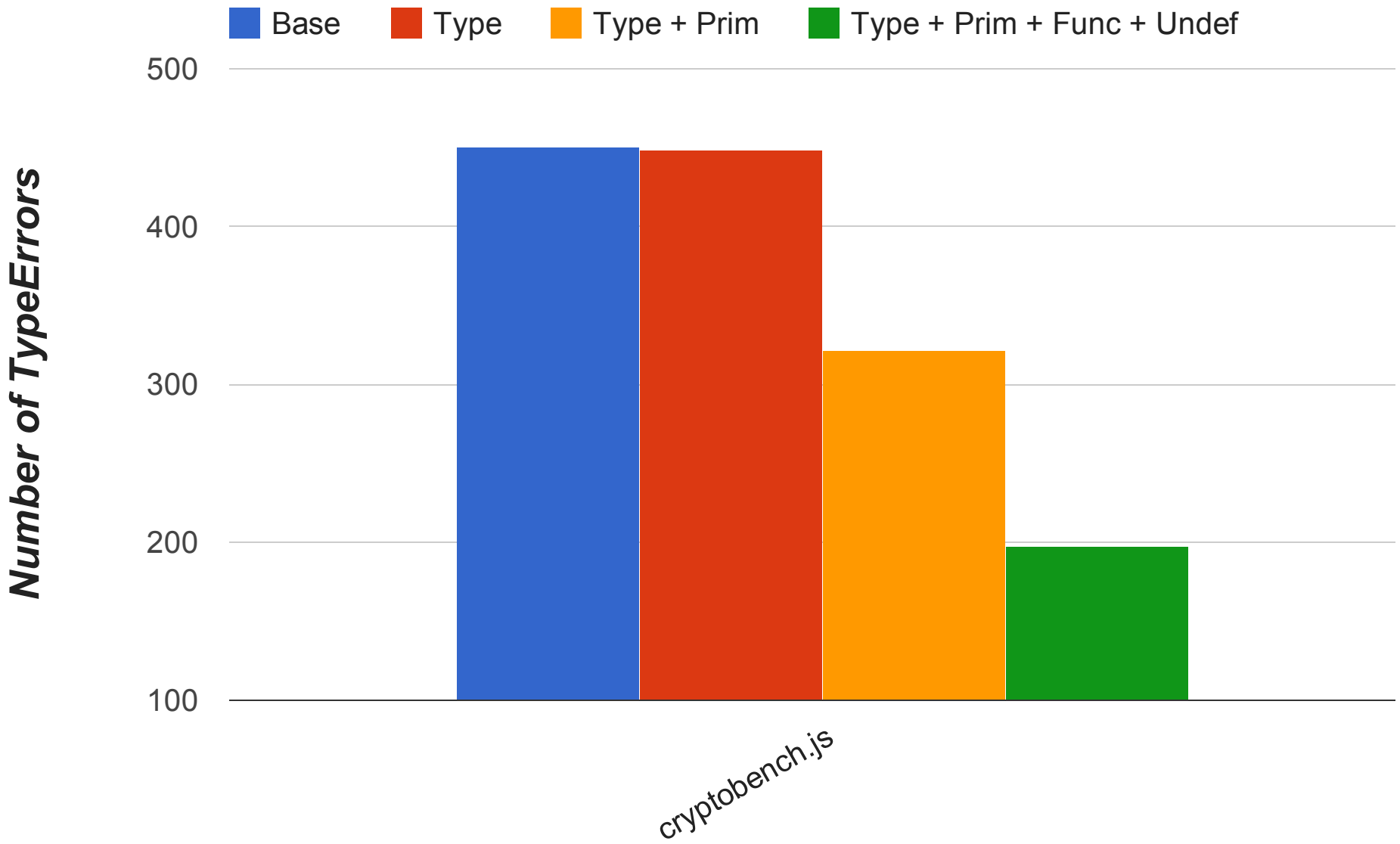
# Overall imprecision results

## (lower is better)



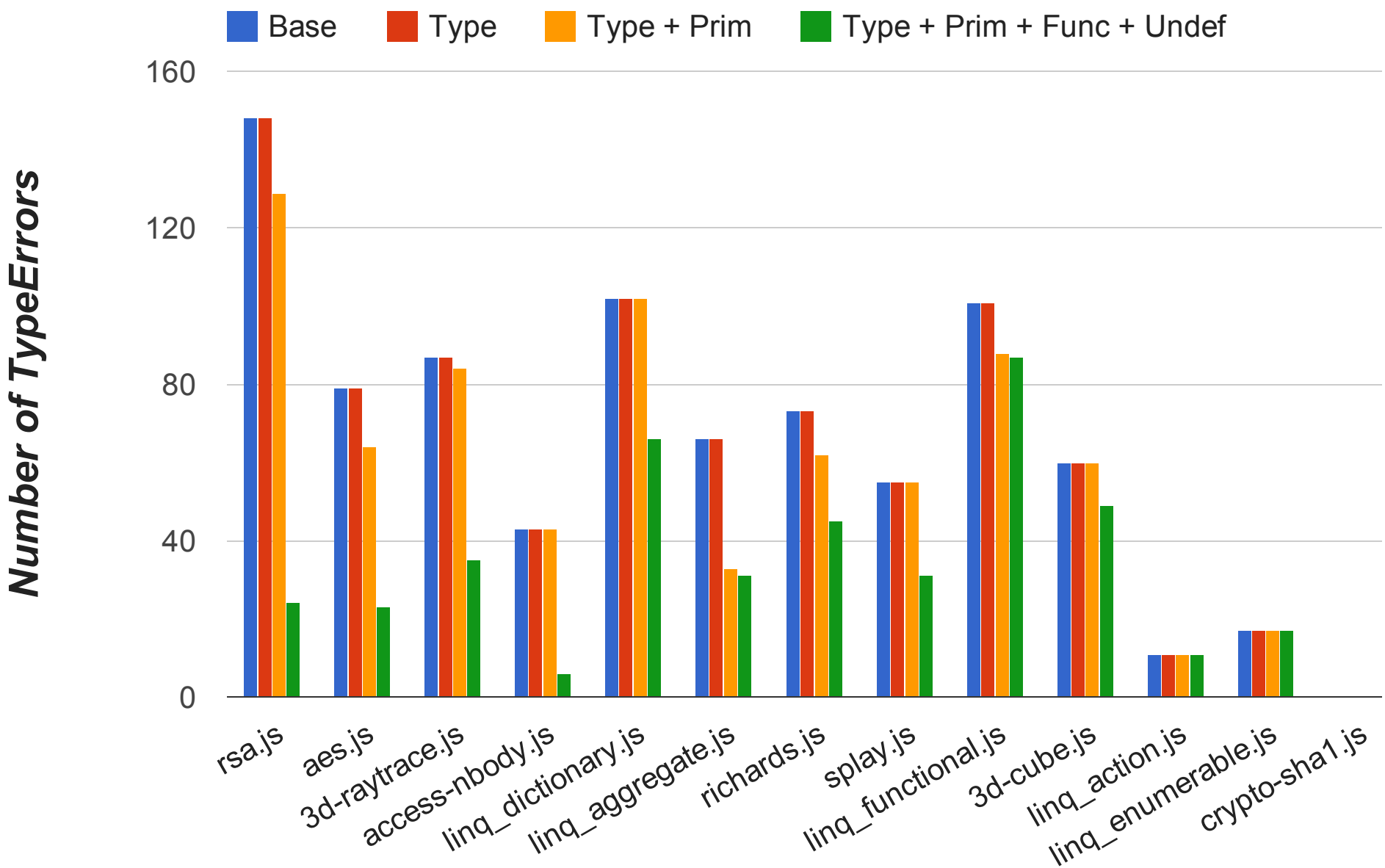
# Cryptobench imprecision results

## (lower is better)



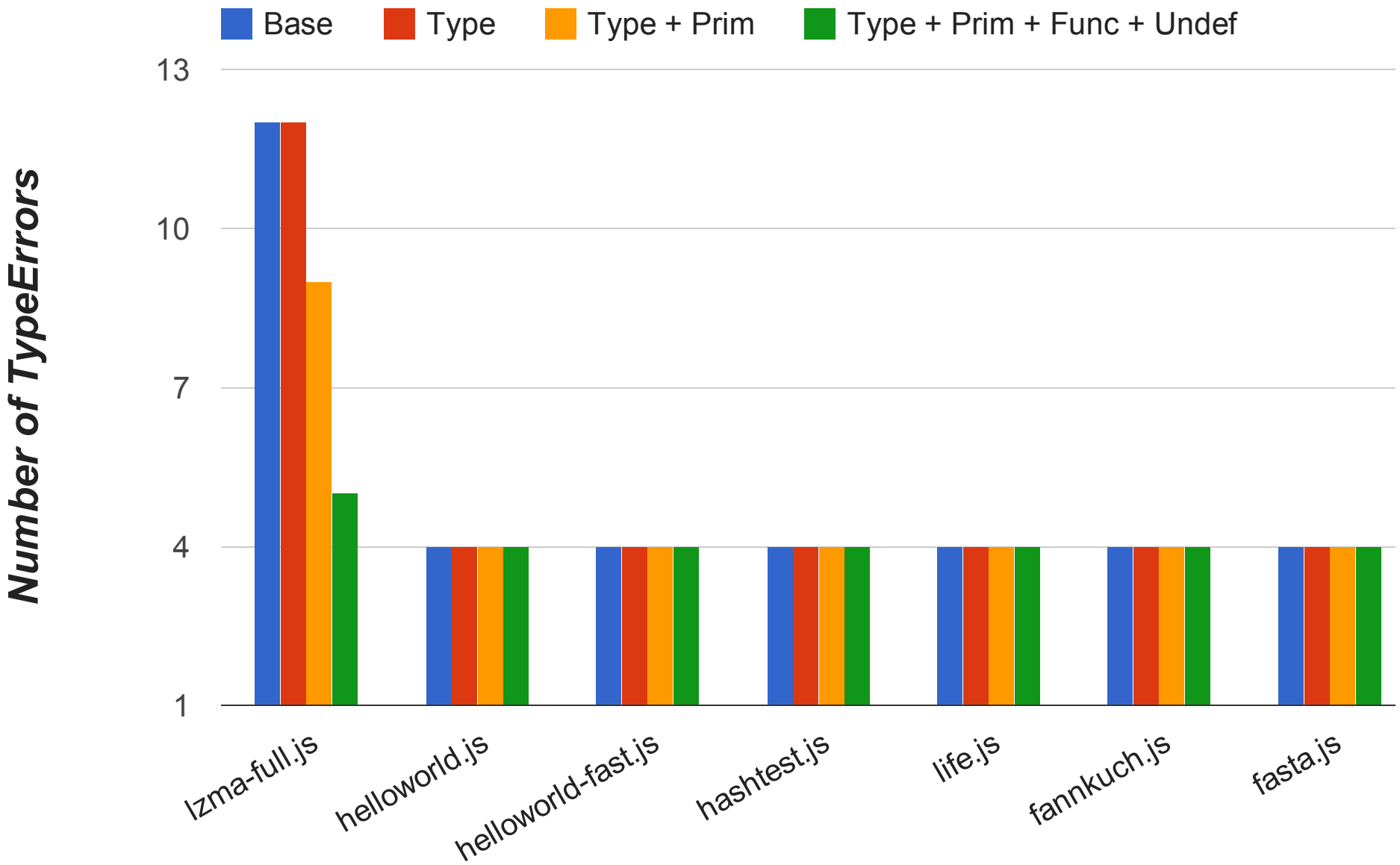
# Sunspider/Octane, open source imprecision results

**(lower is better)**



# Emscripten imprecision results

## (lower is better)



# Conclusion

## Refinement in JavaScript:

- Language semantics "hide" refinement opportunities
- Our work makes implicit branches syntactically explicit
- Refinement over semantic branches improves analysis precision


## Acknowledgements

- Programming Languages Lab at UCSB
- Thomas Ashmore, Jane Hoffswell, Ben Wiedermann, Ben Hardekopf, Vineeth Kashyap
- Funded by NSF CCF-1117165

# Refinement implementation matches intuition

If the analysis refines over $e$...

```
1: if e then
2:     ...
3: else
4:     ...
5: end if
```

# Refinement implementation matches intuition

If the analysis refines over $e$...

```
1: if e then
2:     ...                    ▷ then e is assumed to be true here...
3: else
4:     ...
5: end if
```

# Refinement implementation matches intuition

If the analysis refines over $e$...

```
1: if e then
2:     ...                          ▷ then e is assumed to be true here...
3: else
4:     ...                          ▷ ...and false here.
5: end if
```

# Refinement over implicit branches

## (object field access)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

```
1: if x is undefined or null then
2:     TypeError
3: else
4:     ...
5: end if
```

# Refinement over implicit branches

## (object field access)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

```
1: if x is undefined or null then
2:     TypeError                          ▷ x inferred to be null.
3: else
4:     ...
5: end if
```

# Refinement over implicit branches

## (object field access)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

1: **if** x is undefined or null **then**
2: TypeError ▷ $x$ inferred to be *null*.
3: **else**
4: ... ▷ $x$ inferred to be one of $2, FuncObject$.
5: **end if**

# Refinement over implicit branches

## (function invocation)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

```
1: if x is callable then
2:     invoke x
3:     ...
4: else
5:     TypeError
6: end if
```

# Refinement over implicit branches

## (function invocation)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

```
1: if x is callable then
2:     invoke x                          ▷ x inferred to be FuncObject
3:     ...
4: else
5:     TypeError
6: end if
```

# Refinement over implicit branches

## (function invocation)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

```
1: if x is callable then
2:     invoke x                          ▷ x inferred to be FuncObject
3:     ...
4: else
5:     TypeError                         ▷ x inferred to be one of null, 2.
6: end if
```

# Refinement over implicit branches

## (type conversion)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

1: **if** x is an object **then**
2: $y \leftarrow x.valueOf() + 3$
3: ...
4: **else**
5: $y \leftarrow x + 3$
6: ...
7: **end if**

# Refinement over implicit branches

## (type conversion)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

1: **if** x is an object **then**
2: $y \leftarrow x.valueOf() + 3$ ▷ $x$ inferred to be $FuncObject$.
3: ...
4: **else**
5: $y \leftarrow x + 3$
6: ...
7: **end if**

# Refinement over implicit branches

## (type conversion)

Suppose $x$ may be one of $\{null, 2, FuncObject\}$.

1: **if** x is an object **then**
2: $y \leftarrow x.valueOf() + 3$ ▷ $x$ inferred to be $FuncObject$.
3: ...
4: **else**
5: $y \leftarrow x + 3$ ▷ $x$ inferred to be one of $null, 2$.
6: ...
7: **end if**

# Our experimental hypothesis:

**Successful refinement requires:**

1. Imprecision in original analysis.
2. Analysis output heavily dependent on imprecision.
3. Refinement limits imprecision impact.

# JavaScript's default value is "undefined"

## (i.e., why an analysis might be imprecise)

```
1 object[fieldName];
```

1: **if** fieldName is within the inheritance chain of object **then**
2: return lookup of fieldName
3: **else**
4: return `undefined`
5: **end if**

# `Undefined` imprecisions induce possible TypeErrors

## (i.e., why analysis output depends on imprecision)

```
1
2 var f1 = new Foo();
3 var f2 = new Foo();
4 f1.init();
5 f2.init();
```

# `Undefined` imprecisions induce possible TypeErrors

## (i.e., why analysis output depends on imprecision)

```
1 // Foo.Prototype.init is one of {undefined, FuncObject}
2 var f1 = new Foo();
3 var f2 = new Foo();
4 f1.init();    TypeError
5 f2.init();    TypeError
```

# Our refinements restrict possible TypeErrors

## (i.e., why refinement affects output)

```
1 // Foo.Prototype.init is one of {undefined, FuncObject}
2 var f1 = new Foo();
3 var f2 = new Foo();
4 f1.init();    TypeError
5 f2.init();    safe!
```