Accelerating JavaScript Static Analysis via Dynamic Shortcuts

Joonyoung Park* Jihyeok Park*

Dongjun Youn Sukyoung Ryu

Korea Advanced Institute of Science and Technology

Aug 25/26 2021
FSE

*Both authors contributed equally to the paper

JavaScript

- The de-facto programming language for web applications
  - TIOBE index: the 7th popular language

Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
7	7		JavaScript	2.95%	+0.07%

  - Github: the most dominant language

Year: 2021 | Quarter: 2

# Ranking	Programming Language	Percentage (YoY Change)	YoY Trend
1	JavaScript	19.014% (+0.225%)	

JavaScript Static Analysis

- Semantics analyses based on abstract interpretation

  + Soundness: detecting all possible bugs

  + Termination

  - Effort for opaque code modeling

  - Precision (false positives)

  - Performance (analysis time)

Combined Approaches

- Combinding Static and Dynamic Analyses
  - Existing techniques

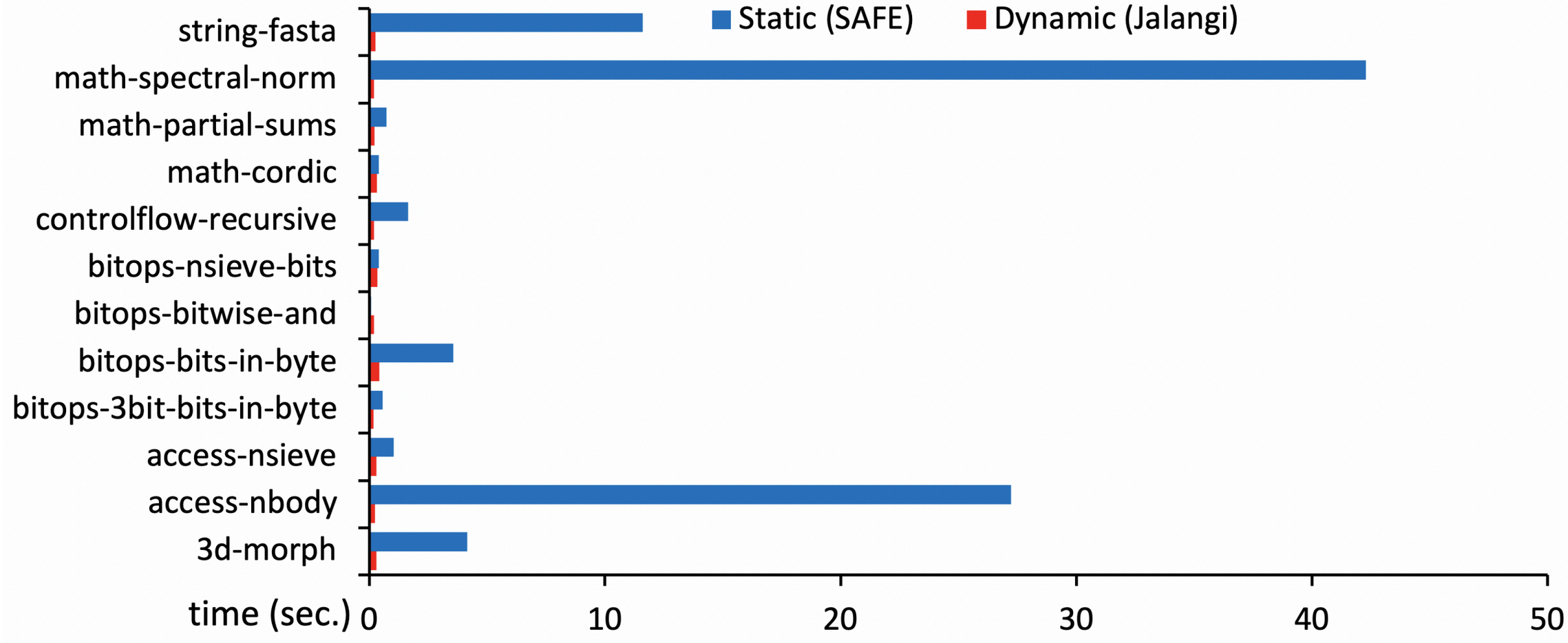
~~- Effort for opaque code modeling~~

~~- Precision~~

- Performance ← Our main goal

Motivational Experiments

- SunSpider benchmark results of static and dynamic analyses

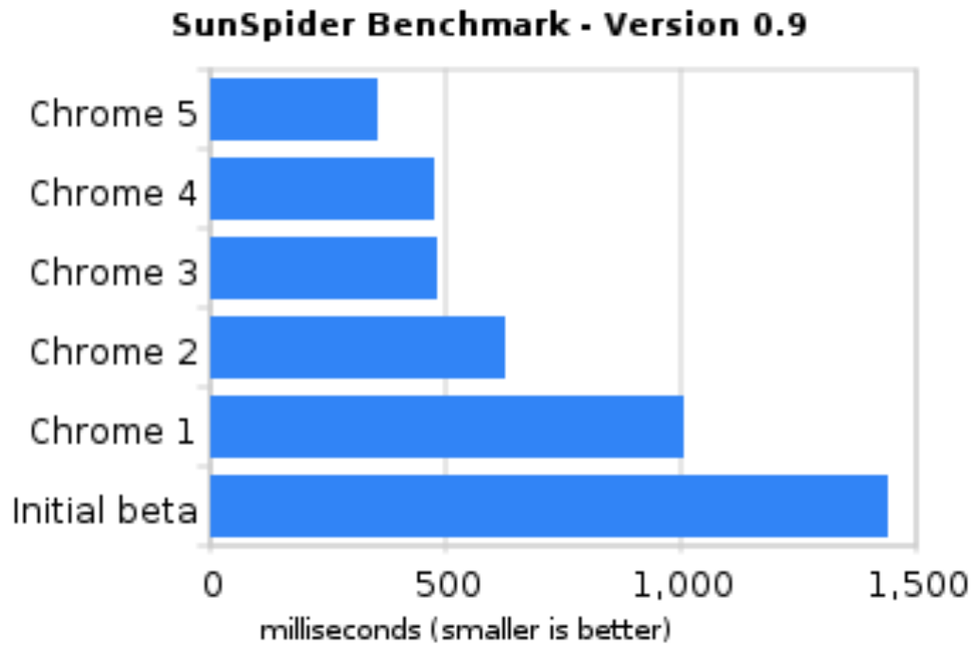


- Jalangi is 34.8x faster than SAFE on average.

  - Jalangi runs on the optimized JavaScript engine (V8)

Optimizations on JavaScript Engine

- History: browser wars
    - Competition of browser speed among vendors
- Complex and advanced techniques
    - Hidden class
    - JIT compilation
    - …



Maximing the Proportion of Dynamic Analysis

- Motivation
  - The dynamic analyzer is fast
  - The static analyzer is slow
- Goal
  - Using dynamic analysis as much as possible
- Problem
  - Which flows should be analyzed by the dynamic analysis?

Syntactic Approach?

- The syntactic approach for Java may not work well
- Lodash's concat function

```
1  function concat() {
2    var length = arguments.length;
3    if (!length) return [];
4    var array = arguments[0],
5        args  = Array(length - 1),
6        index = length;
7    while (index--)
8      args[index-1] = arguments[index];
9    return arrayPush(isArray(array) ?
10     copyArray(array) : [array],
11     baseFlatten(args, 1));
12 }
```

- Zoom (Alexa's the 7th top site)
  - Constants

```
13 function changeCountry(G) { ...
14   if (G.selectedVal === "US" && state) {
15     // deterministic arguments of `concat`
16     state.items = _.concat([["Other", "Other"]],
17       WebinarBase.questions.state.items);
18     state.selectedVal = _.head(_.head(C.items));
19   }
20 }
```

WebinarBase.questions.state.items = // 55 elements
[["AL","Alabama"], ..., ["WY", "Wyoming"]]

  - Data from server

```
22 function getData(e) {
23   var option = ... // option for server connection
24   post(option).then(function(e) {
25     if (e.total_records && e.total_records > 0) {
26       // non-deterministic arguments of `concat`
27       this.pastEvents =
28         _.concat(this.pastEvents, e.events);
29       this.total = e.total_records;
30     } else this.noPastData = !0
31   })
32 }
```

Our solution

- A trial (dynamic analysis) and error (static analysis) method
  - Detecting the use of any abstract value during the dynamic analysis
- Finding
  - Even though abstract values are present, concrete semantics can handle some instructions like assignment

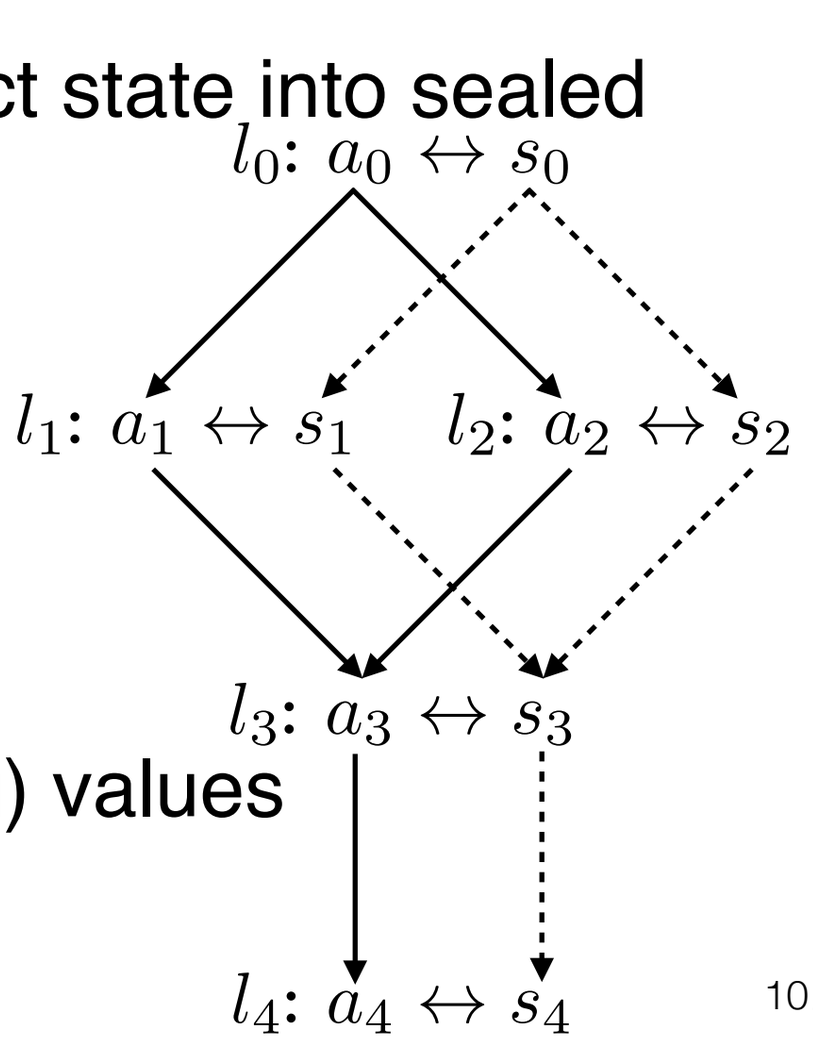
```
1 var v = ...; // an abstract value
2 var obj = { p1: v }, y = "p";
3 var x = obj[y + 1]; // so far so good
```

Dynamic Shortcut

- Converters ($\leftrightarrow$)
- Sealed state (i)
  - Sealing abstract values in an abstract state into sealed values

---▸

- Sealed execution ( )
  - Runs on concrete engines (fast!)
  - Detects the use of sealed (unsealing) values

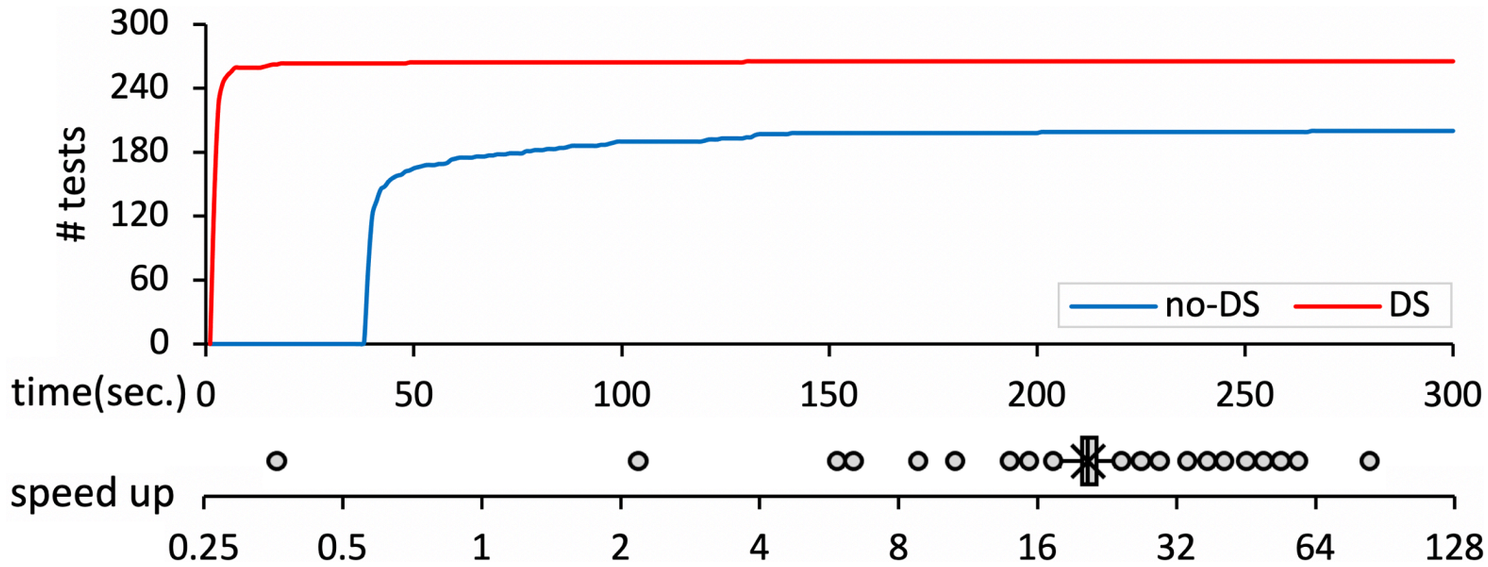


Evaluation

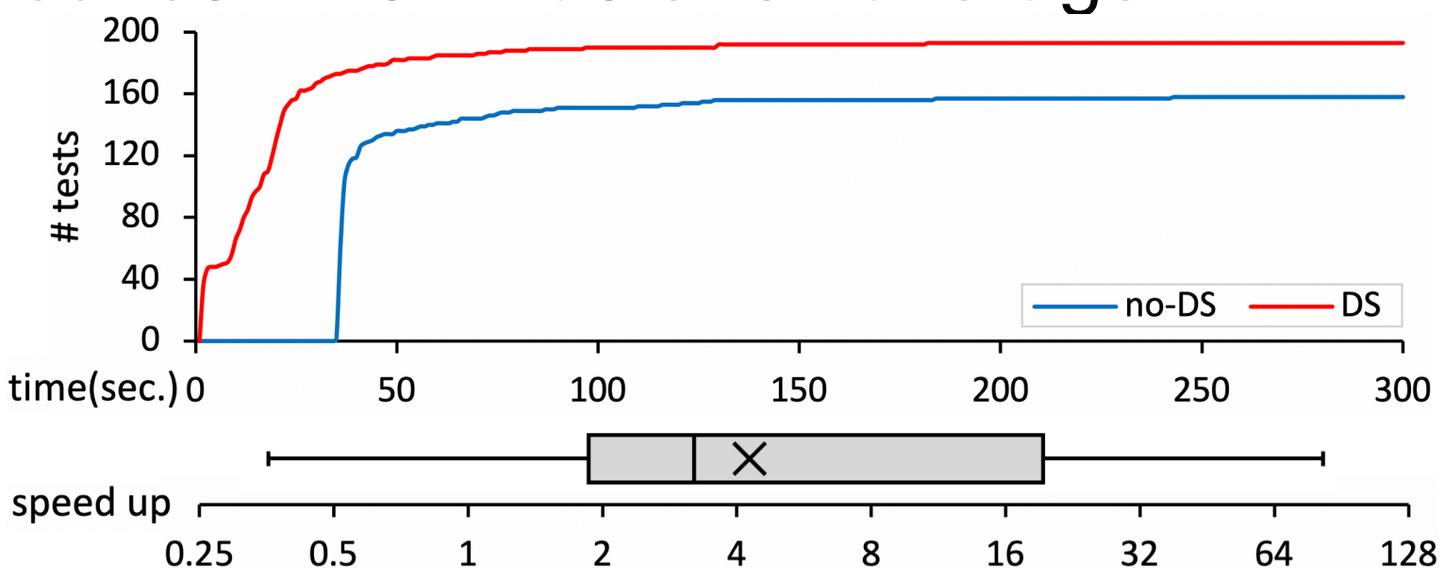
- Benchmark: Lodash v4 test set, 269 files
  - The original version
  - The abstracted version
    - A random replacement of a premitive literal to an abstract value
- Comparison between the original SAFE (no-DS) and $\mathrm{SAFE}_{\mathrm{DS}}$ (DS).
  - RQ1) Analysis performance: analysis time
  - RQ2) Precision: # failed assertions
  - RQ3) Efforts to model opaque code: # dynamically covered opaque functions

RQ1) Analysis Speed-up

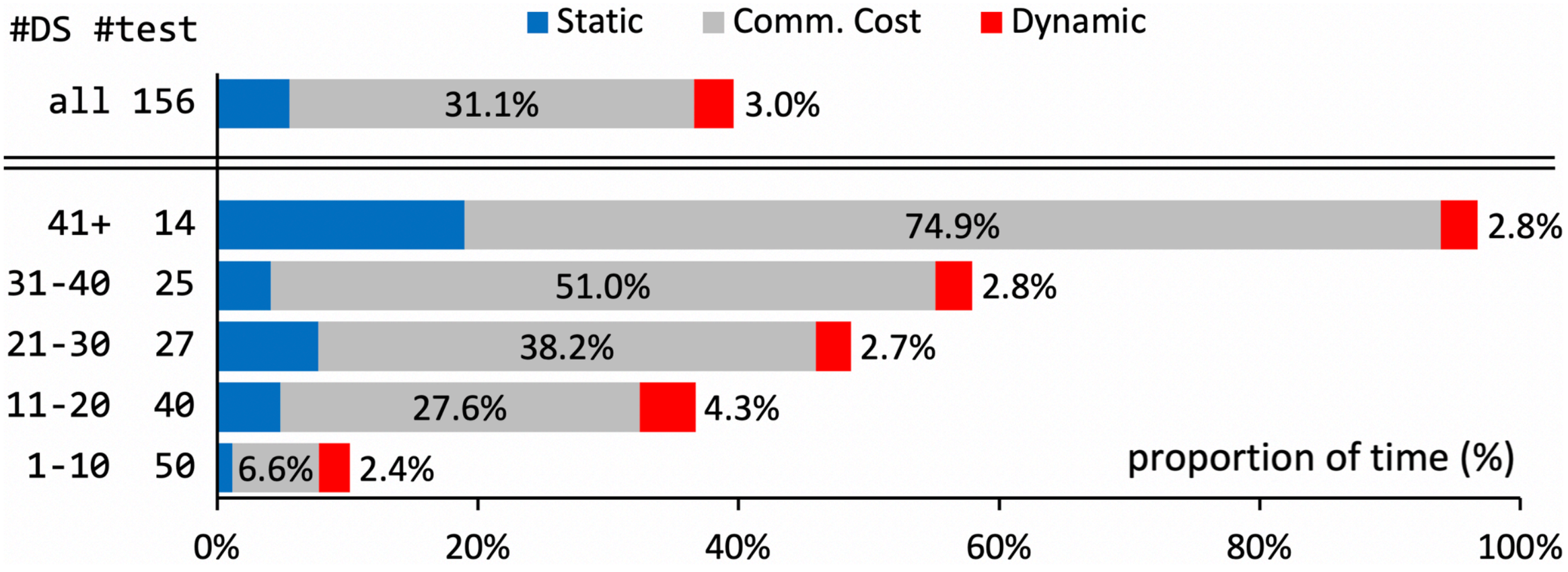
- Original test: 22.30x faster on average



- Abstracted test: 7.81x faster on average



Overhead Inspection



Summary

- The sound and most flexible combination of static and dynamic analyses

  + Better Performance

  + Efficient opaque code modeling

  + Higher Precision

  ~~- Sacrifice of soundness~~

  ~~- Syntactic limitations~~