Static Analysis with Demand-Driven Value Refinement


Benno Stein
University of Colorado Boulder


Benjamin Barslev Nielsen
Aarhus University


Bor-Yuh Evan Chang
University of Colorado Boulder


Anders Møller
Aarhus University

CUPLV

Problem

Even a minor precision loss in whole-program JavaScript static analysis can incur a huge slowdown as a result of *dynamic property access*.

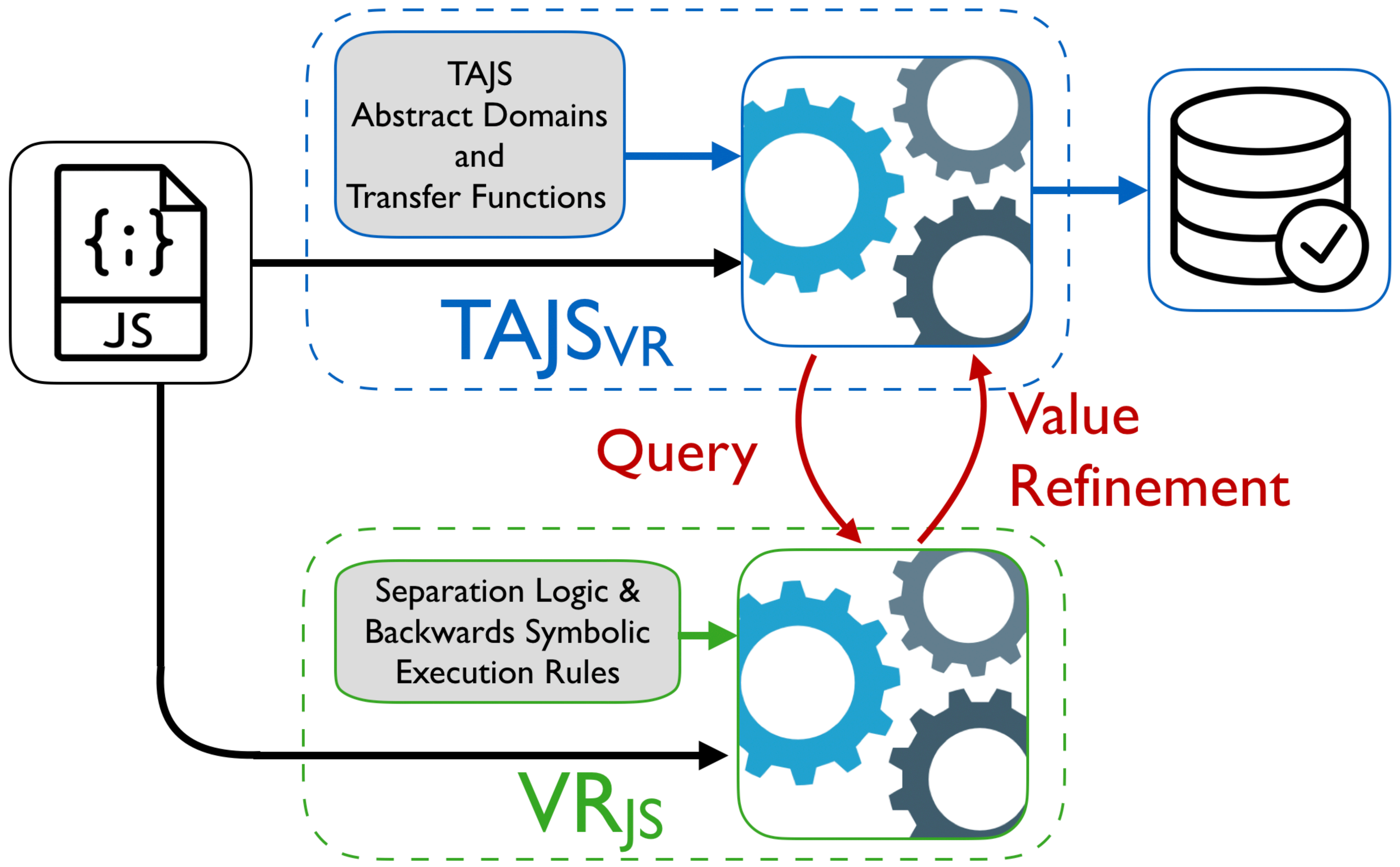
```
// library code
var src = {
 foo: function f1(){…},
 bar: function f2(){…} };
var lib = {};
for (var p in src)
  { lib[p] = src[p]; }
// client code
lib.foo();
```

Imprecise handling of dynamic object manipulation...

... leads to spurious call edges — analysis can't resolve the target of this call!

Our Approach

We augment a whole-program dataflow analysis ($TAJS_{VR}$) with a ***value refiner*** (VR_{JS}): a very precise and targeted analysis that non-monotonically refines the base analysis abstract state on the fly, increasing precision at crucial locations like the write to `lib[p]` above.



Queries are issued based on *semantic* heuristics in $TAJS_{VR}$, so value refinement provides precision where it's needed and doesn't incur additional cost elsewhere, unlike existing techniques that rely on brittle *syntactic* patterns.

Refinement Analysis

Value refinement answers queries of the form "which values can this memory location hold at this abstract state?", providing relational precision to a non-relational underlying analysis.

Our value refiner VR_{JS} is a separation logic-based backwards abstract interpreter that soundly over-approximates possible dataflow to a queried location.

ASSUME

$$\frac{\hat{h} = x \mapsto \hat{x}}{\langle \hat{h} \wedge \hat{x} \wedge \pi \rangle \ \texttt{assume}\ x \ \langle \hat{h} \wedge \pi \rangle}$$

Accumulate path conditions on backwards search

CONSEQUENCE

$$\frac{\varphi_2' \Rightarrow \varphi_2 \qquad \langle \varphi_2' \rangle\ s\ \langle \varphi_1' \rangle \qquad \varphi_1 \Rightarrow \varphi_1'}{\langle \varphi_2 \rangle\ s\ \langle \varphi_1 \rangle}$$

Inverse of standard consequence rule!

WRITEPROP

$$\frac{\hat{h} = x \mapsto \hat{x} * y \mapsto \hat{y} * z \mapsto \hat{z}}{\langle (\hat{h} \wedge \pi)[\hat{z}/\hat{z}'] \rangle\ x[y] = z\ \langle \hat{h} * \hat{x}[\hat{y}] \mapsto \hat{z}' \wedge \pi \rangle}$$

Transfer postcondition constraints on *l*-val to precondition constraints on *r*-val

The analysis is defined in terms of ***refutation-sound*** triples of the form $\langle \varphi \rangle\ s\ \langle \varphi' \rangle$ that hold if any concrete run through s that *ends* in φ' must have *started* in φ, over-approximating the backwards semantics.

Abstract

Static analysis tools for JavaScript must strike a delicate balance, achieving the level of precision required by the most complex features of target programs without incurring prohibitively high analysis time. For example, reasoning about dynamic property accesses sometimes requires precise relational information connecting the object, the dynamically-computed property name, and the property value. Even a minor precision loss at such critical program locations can result in a proliferation of spurious dataflow that renders the analysis results useless.

We present a technique by which a conventional non-relational static dataflow analysis can be combined soundly with a value refinement mechanism to increase precision on demand at critical locations. Crucially, our technique is able to incorporate relational information from the value refinement mechanism into the non-relational domain of the dataflow analysis.

We demonstrate the feasibility of this approach by extending an existing JavaScript static analysis with a demand-driven value refinement mechanism that relies on backwards abstract interpretation. Our evaluation finds that precise analysis of widely used JavaScript utility libraries depends heavily on the precision at a small number of critical locations that can be identified heuristically, and that backwards abstract interpretation is an effective mechanism to provide that precision on demand.

Evaluation

We evaluate the demand-driven value refinement technique by implementing a JavaScript type analysis $TAJS_{VR}$ and comparing it against two state-of-the-art JS analysis tools:

Low overhead on programs that are analyzable without value refinement

Enables analysis of **large examples** from previous works' test corpora

Analyze full test suites of **popular libraries** — both of which were out of the reach of state-of-the-art JavaScript analyzers

Type Analyzer for JavaScript (without value refinement)

Extension to SAFE that targets dynamic property access with syntactic patterns

Our tool: TAJS with demand-driven value refinement

		TAJS		CompAbs		$TAJS_{VR}$	
Benchmark		Success (%)	Time (s)	Success (%)	Time (s)	Success (%)	Time (s)
JQuery	(71 tests)	7%	14.4	0%	-	7%	17.2
JSAI tests	(29 tests)	86%	12.3	34%	32.4	86%	14.3
Prototype	(6 tests)	0%	-	33%	23.1	83%	97.7
Scriptaculous	(1 test)	0%	-	100%	62.0	100%	236.9
Underscore	(182 tests)	0%	-	0%	-	95%	2.9
Lodash3	(176 tests)	0%	-	0%	-	98%	5.5
Lodash4	(306 tests)	0%	-	0%	-	87%	24.7

This work was supported by: ERC#647544, NSF CCF-1619282 & CCF-1055066, and DARPA FA8750-14-2-0263