Goal-Directed Abstract Interpretation for JavaScript


Benno Stein


Bor-Yuh Evan Chang

CUPLV
Programming Languages and Verification
University of Colorado Boulder

Problem: Local loss of precision in whole-program JavaScript static analysis can incur a massive slowdown as a result of dynamic dispatch.

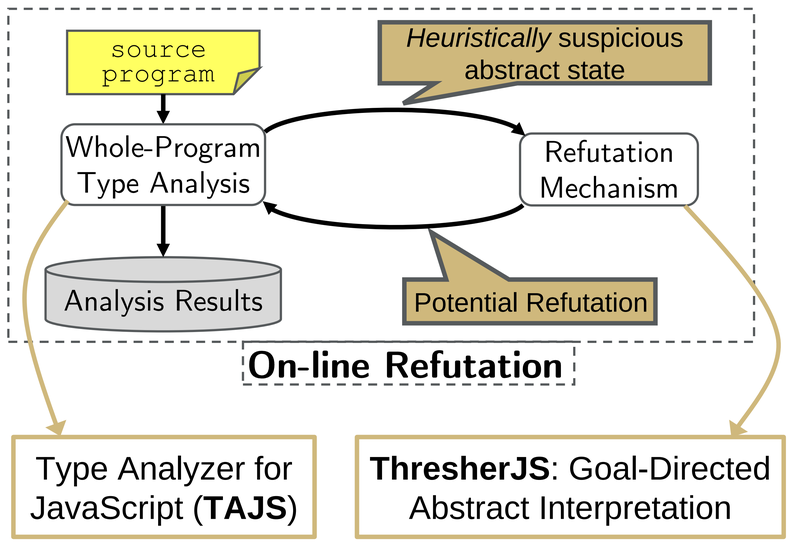
```
var o = {foo : function(){…},
         bar : function(){…},
         baz : function()
               {… ; throw 'error';} };
var prop = someFunction();
o[prop]();
```

Spurious call edges!

Over-approximation in someFunction causes analysis to infer `T` as `prop`'s value

Example: Over-approximation of `prop`'s value leads to spurious data-flow at the `o[prop]` call site.

Our Solution: Provide the whole-program analysis with a mechanism to refine its abstract state at particularly problematic locations.



In collaboration with Aarhus University, we are building an instantiation of this system for JavaScript type analysis (**TAJS-Thresher**).

Goal-Directed Abstract Interpretation

ThresherJS attempts to refute queries by soundly exploring the state-space backwards from a given location and abstract state. If all backwards paths reach a contradiction, the query is *refuted*; if some path reaches an entrypoint without contradiction, the query is *not* refuted – there *may* exist a concrete witness.

Example: Dynamic property read transfer function. Informally, the precondition state asserts that a prototype lookup on y with respect to property z yields some object x' whose z field satisfies any precondition constraints on x.

$$\frac{\hat{y}, \hat{z}, \hat{x}' \text{ fresh } \in \varphi}{\left\{ \varphi \cup \{\pi(\hat{y}, \hat{z}, \emptyset, \emptyset) = \hat{x}', y \mapsto \hat{y}, z \mapsto \hat{z}\} \cup \bigcup_i \hat{x}'[\hat{z}] \mapsto \hat{x}_i \right\} \; x = y[z] \; \left\{ \varphi \wedge \bigwedge_i (x \mapsto \hat{x}_i) \right\}}$$

Constraint Language

$$\begin{array}{lll} \text{constraints} & c & ::= x \mapsto \hat{x} \\ & & \mid \hat{x}_1[\hat{x}_2] \mapsto \hat{x}_3 \\ & & \mid e : \hat{v} \\ & & \mid \pi(\hat{x}_1, \hat{x}_2, u, pg) = \hat{x}_3 \\ \text{abstract states} & \varphi \in \hat{\text{State}} & ::= c \wedge \varphi \mid \text{true} \end{array}$$

Type Constraint

Prototype Constraint

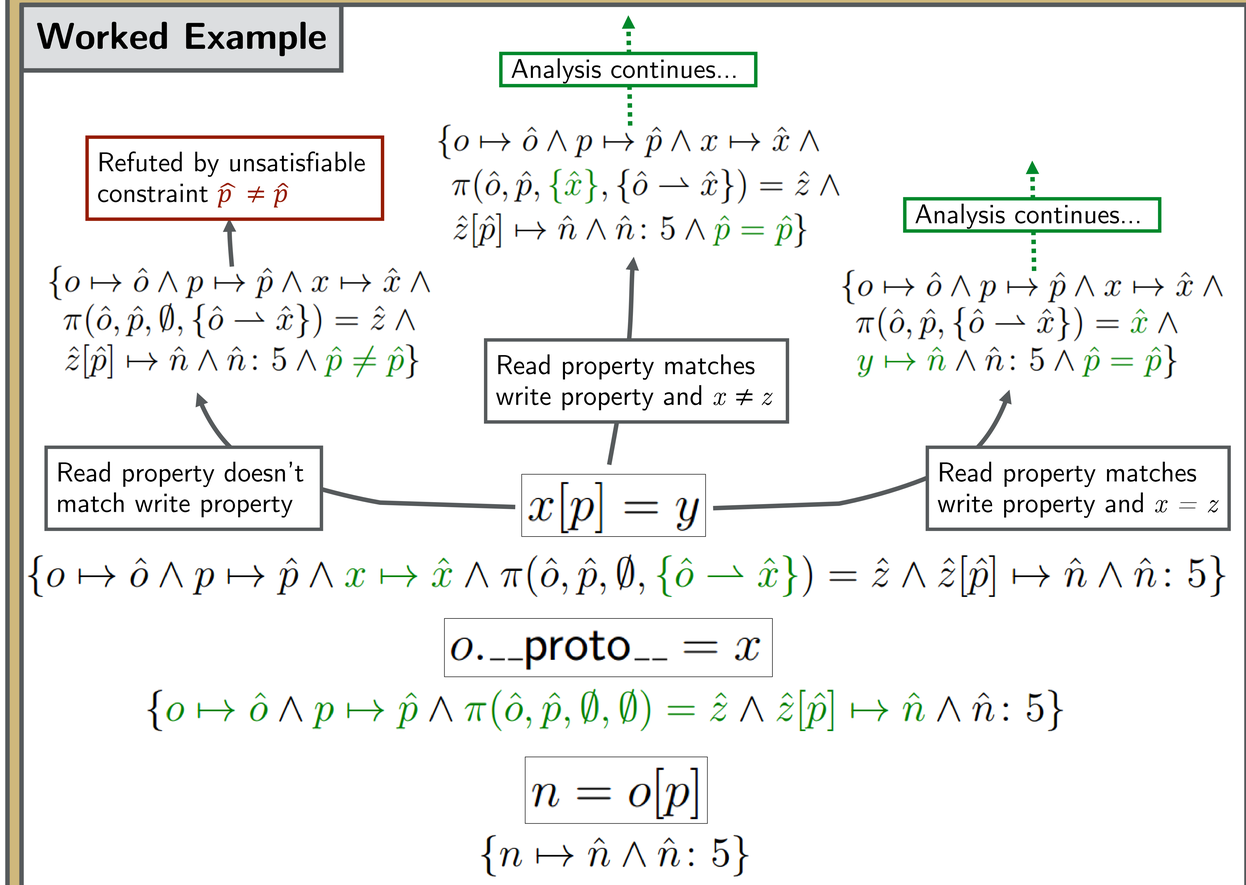
Concretization

$\gamma : \hat{\text{State}} \to \mathcal{P}(\text{State} \times \text{Valua})$

$$\gamma(e : \hat{v}) = \{((l,h,p),\eta) \mid \text{eval}(e,\eta) \sqcap_{\hat{\text{Val}}} \hat{v} \neq \bot\}$$

$$\gamma(\pi(\hat{x}_1, \hat{x}_2, u, pg) = \hat{x}_3) = \left\{ ((l,h,p),\eta) \;\middle|\; \begin{array}{c} \exists \text{ path } w \text{ from } \eta\hat{x}_1 \text{ to } \eta\hat{x}_3 \text{ in} \\ (p \cup \{\eta\hat{x} \rightharpoonup \eta\hat{y} \mid \hat{x} \rightharpoonup \hat{y} \in pg\}). \\ (\forall \hat{x} \in u \,.\, \eta\hat{x} \notin w) \quad \wedge \\ (\forall v \in w \,.\, (a, \eta\hat{x}_2 \in \text{dom}(h) \Rightarrow a = \eta\hat{x}_3) \end{array} \right\}$$

Worked Example



Abstract

JavaScript is notoriously difficult to analyze due to its rampant use of standard dynamic features (e.g. duck typing, dynamic dispatch, first-class functions, and run-time string evaluation), as well as its idiosyncratic approach to scoping (scope object chains) and inheritance (prototyping). Therefore, despite its near-universal adoption as a client-side scripting language and its increasing use in server-side and mobile applications, JavaScript is rarely analyzed in practice and can be quite buggy, unreliable, and unsafe.

We present a novel technique to improve precision and efficiency of JavaScript analysis by combining a forwards whole-program type analysis with a goal-directed backwards abstract interpretation refutation mechanism. The backwards abstract interpretation can operate either as a standalone tool to refute false alarms that arise from over-approximation in the forwards analysis, or on-line, refuting spurious data-flow on demand at critical points during the whole-program forwards analysis.

References: **[1]** S.H. Jensen, A. Møller, and P. Thiemann. *Type Analysis for JavaScript.* In *SAS*, 2009. **[2]** S. Blackshear. *Flexible Goal-Directed Abstraction.* PhD thesis, University of Colorado Boulder, 2015. **[3]** S. Blackshear, B.-Y. E. Chang, and M. Sridharan. *Thresher: Precise refutations for heap reachability.* In *PLDI*, 2013.