Language-independent

# Semantics-Based Program Verifiers for All Languages

Andrei Stefanescu *Daejun Park*
Shijiao Yuwen Yilong Li Grigore Rosu

Nov 2, 2016 @ OOPSLA'16

# Problems with state-of-the-art verifiers

- Missing details of language behaviors
    - e.g., VCC’s false positives/negatives, undefinedness of SV-COMP benchmarks
- Fragmentation: specific to a fixed language

# Missing details of language behaviors

```
unsigned x = UINT_MAX;
unsigned y = x + 1;
_(assert y == 0)
```

VCC incorrectly reported an overflow error

# Missing details of language behaviors

```
int assign(int *p, int x)
  _(ensures *p == x)
  _(writes p)
{
  return (*p = x);
}

void main() {
  int r;
  assign(&r, 0) == assign(&r, 1);
  _(assert r == 1)
}
```

VCC incorrectly proved it, missing non-determinism

# Missing details of language behaviors

## 14% of SV-COMP's "Correct Programs" are Undefined!

Posted on September 18, 2016 by Grigore Rosu


Last April (2016), I gave a tutorial on K at ETAPS'16 in Eindhoven, Netherlands, where I also demonstrated RV-Match. During the week that I spent there, I heard several friends and colleagues who were involved with the Competition on Software Verification, SV-COMP, that some of the benchmark's *correct* programs appear to be undefined. What? So some of the assumed-correct C programs that are used to evaluate the best program verifiers in the world are actually *wrong* programs? Continue reading →

* Grigore Rosu, https://runtimeverification.com/blog/?p=200

# Problems with state-of-the-art verifiers

- Missing details of language behaviors
- Fragmentation: specific to a fixed language
  - e.g., KLEE (LLVM), JPF (JVM), Pex (.NET), CBMC (C), SAGE (x86), ...
  - Implemented similar heuristics/optimizations: duplicating efforts

# Our solution

Clear separation, yet smooth integration,

Between semantics reasoning and proof search,

Using language-independent logic & proof system

# Idea: separation of concerns

**Semantics Reasoning**

**Proof Search**

Language semantics:

- C (c11, gcc, clang, …)
- Java (6, 7, 8, …)
- JavaScript (ES5, ES6, …)
- …

Verification techniques:

- Deductive verification
- Model checking
- Abstract interpretation
- …

Defined/implemented once, and reused for all others

# Idea: separation of concerns

**Semantics Reasoning**

**Proof Search**

Language semantics:

- C (c11, gcc, clang, …)
- Java (6, 7, 8, …)
- JavaScript (ES5, ES6, …)
- …

Verification techniques:

- Deductive verification
- Model checking
- Abstract interpretation
- …

VCC

JPF

CBMC

Defined/implemented once, and reused for all others

# Language-independent verification framework

Semantics $\models$ Program & Properties

Language-independent uniform notation (logic)

Language-independent proof systems $\vdash$

Proof automation

- Provides a nice interface (logic) in which both language semantics and program properties can be described.
- Proof search in this logic becomes completely language-independent.

# Language-independent verification framework

Operational semantics
(C/Java/JavaScript semantics)

Reachability properties
(Functional correctness of heap manipulations)

Language-independent uniform notation
(Matching logic reachability)

Language-independent proof systems
(Matching logic reachability proof systems)

Proof automation
(Symbolic execution, SMT, Natural proofs, ...)

# Operational semantics

- Easy to define and understand than axiomatic semantics
  - Require little mathematical knowledge
  - Similar to implement language interpreter
- Executable, thus testable
  - Important when defining real large languages
- Shown to scale to defining full language semantics
  - C, Java, JavaScript, Python, PHP, ...

# Language-independent verification framework

Operational semantics
(C/Java/JavaScript semantics)

Reachability properties
(Functional correctness of heap manipulations)

Language-independent uniform notation
(Reachability logic)

Language-independent proof systems
(Reachability logic proof systems)

Proof automation
(Symbolic execution, SMT, Natural proofs, ...)

# Reachability logic

- Unifying logic in which both language semantics and program correctness properties can be specified.

reachability between "patterns"

$$\varphi \Rightarrow \varphi'$$

"pattern" formula
representing a set of program states

- Pattern formula is FOL without predicate symbols.
  - Similar to algebraic data types for pattern matching in functional languages such as OCaml and Haskell.

# Expressiveness: semantics

- In OCaml:

```
match e with
| ADD(x,y)                  => x + y
| SUB(x,y)                  => x - y
| MUL(x,y)                  => x * y
| DIV(x,y) when y != 0  => x / y
```

# Expressiveness: semantics

- In OCaml:

```
match e with
| ADD(x,y)                  => x + y
| SUB(x,y)                  => x - y
| MUL(x,y)                  => x * y
| DIV(x,y) when y != 0      => x / y
```

- In Reachability logic:

```
  ADD(x,y)                  => x + y
  SUB(x,y)                  => x - y
  MUL(x,y)                  => x * y
  DIV(x,y)  /\   y != 0     => x / y
```

# Expressiveness: properties

- In Hoare logic:

```
fun insert (v: elem, t: tree) return (t’: tree)
  @requires bst(t)
  @ensures  bst(t’)
       and  keys(t’) == keys(t) \union { v }
```

# Expressiveness: properties

- In Hoare logic:

```
fun insert (v: elem, t: tree) return (t’: tree)
  @requires bst(t)
  @ensures  bst(t’)
       and  keys(t’) == keys(t) \union { v }
```

- In Reachability logic:

```
  insert /\ bst(t)
=>
     .   /\ bst(t’)
         /\ keys(t’) == keys(t) \union { v }
```

# Expressiveness

- Reachability formula can specify:
  - Pre-/post-conditions
  - Safety properties by augmenting semantics
  - No liveness properties yet (ongoing work)
- Pattern formula can include:
  - Recursive predicates
  - Separation logic formula

# Language-independent verification framework

Operational semantics
(C/Java/JavaScript semantics)

Reachability properties
(Functional correctness of heap manipulations)

Language-independent uniform notation
(Reachability logic)

**Language-independent proof systems**
**(Reachability logic proof systems)**

Proof automation
(Symbolic execution, SMT, Natural proofs, ...)

# Proof system

Language-independent proof system for deriving sequents of the form:

STEP :

$$\frac{\models \varphi \rightarrow \bigvee_{\varphi_l \Rightarrow^{\exists} \varphi_r \,\in\, \mathcal{S}} \exists \mathit{FreeVars}(\varphi_l).\varphi_l \qquad \models ((\varphi \wedge \varphi_l) \neq \bot_{Cfg}) \wedge \varphi_r \rightarrow \varphi' \quad \text{for each } \varphi_l \Rightarrow^{\exists} \varphi_r \in \mathcal{S}}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^{\forall} \varphi'}$$

AXIOM :

$$\frac{\varphi \Rightarrow^{Q} \varphi' \in \mathcal{S} \cup \mathcal{A} \qquad \psi \text{ is FOL formula (logical frame)}}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \wedge \psi \Rightarrow^{Q} \varphi' \wedge \psi}$$

REFLEXIVITY :

$$\frac{\cdot}{\mathcal{S}, \mathcal{A} \vdash \varphi \Rightarrow^{Q} \varphi}$$

TRANSITIVITY :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^{Q} \varphi_2 \qquad \mathcal{S}, \mathcal{A} \cup C \vdash \varphi_2 \Rightarrow^{Q} \varphi_3}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^{Q} \varphi_3}$$

CONSEQUENCE :

$$\frac{\models \varphi_1 \rightarrow \varphi_1' \qquad \mathcal{S}, \mathcal{A} \vdash_C \varphi_1' \Rightarrow^{Q} \varphi_2' \qquad \models \varphi_2' \rightarrow \varphi_2}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^{Q} \varphi_2}$$

CASE ANALYSIS :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^{Q} \varphi \qquad \mathcal{S}, \mathcal{A} \vdash_C \varphi_2 \Rightarrow^{Q} \varphi}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow^{Q} \varphi}$$

ABSTRACTION :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^{Q} \varphi' \qquad X \cap \mathit{FreeVars}(\varphi') = \emptyset}{\mathcal{S}, \mathcal{A} \vdash_C \exists X\, \varphi \Rightarrow^{Q} \varphi'}$$

CIRCULARITY :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow^{Q} \varphi'\}} \varphi \Rightarrow^{Q} \varphi'}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^{Q} \varphi'}$$

# Proof system

Language-independent proof system for deriving sequents of the form:

semantics ↓ property ↓

$$\begin{array}{l} \varphi_1 \Rightarrow \varphi_1' \\ \varphi_2 \Rightarrow \varphi_2' \\ \varphi_3 \Rightarrow \varphi_3' \\ \vdots \end{array} \quad \vdash \quad \varphi \Rightarrow \varphi'$$

STEP :

$$\frac{\models \varphi \rightarrow \bigvee_{\varphi_l \Rightarrow^\exists \varphi_r \,\in\, \mathcal{S}} \exists \mathit{FreeVars}(\varphi_l).\varphi_l \qquad \models ((\varphi \wedge \varphi_l) \neq \bot_{Cfg}) \wedge \varphi_r \rightarrow \varphi' \quad \text{for each } \varphi_l \Rightarrow^\exists \varphi_r \in \mathcal{S}}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^\forall \varphi'}$$

AXIOM :

$$\frac{\varphi \Rightarrow^Q \varphi' \in \mathcal{S} \cup \mathcal{A} \qquad \psi \text{ is FOL formula (logical frame)}}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \wedge \psi \Rightarrow^Q \varphi' \wedge \psi}$$

REFLEXIVITY :

$$\frac{\cdot}{\mathcal{S}, \mathcal{A} \vdash \varphi \Rightarrow^Q \varphi}$$

TRANSITIVITY :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi_2 \qquad \mathcal{S}, \mathcal{A} \cup C \vdash \varphi_2 \Rightarrow^Q \varphi_3}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi_3}$$

CONSEQUENCE :

$$\frac{\models \varphi_1 \rightarrow \varphi_1' \qquad \mathcal{S}, \mathcal{A} \vdash_C \varphi_1' \Rightarrow^Q \varphi_2' \qquad \models \varphi_2' \rightarrow \varphi_2}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi_2}$$

CASE ANALYSIS :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi \qquad \mathcal{S}, \mathcal{A} \vdash_C \varphi_2 \Rightarrow^Q \varphi}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow^Q \varphi}$$

ABSTRACTION :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^Q \varphi' \qquad X \cap \mathit{FreeVars}(\varphi') = \emptyset}{\mathcal{S}, \mathcal{A} \vdash_C \exists X\, \varphi \Rightarrow^Q \varphi'}$$

CIRCULARITY :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow^Q \varphi'\}} \varphi \Rightarrow^Q \varphi'}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^Q \varphi'}$$

# Proof system

Language-independent proof system for deriving sequents of the form:

semantics ↓ property ↓

$$\begin{array}{l} \varphi_1 \Rightarrow \varphi_1' \\ \varphi_2 \Rightarrow \varphi_2' \\ \varphi_3 \Rightarrow \varphi_3' \\ \vdots \end{array} \quad \vdash \quad \varphi \Rightarrow \varphi'$$

```
ADD(x,y) => x + y
SUB(x,y) => x - y
MUL(x,y) => x * y
         ⋮
```

$\vdash$

```
  insert /\ bst(t)
=>
     .    /\ bst(t')
          /\ keys(t') == keys(t) \union { v }
```

STEP :

$$\frac{\models \varphi \to \bigvee_{\varphi_l \Rightarrow^{\exists} \varphi_r \,\in\, \mathcal{S}} \exists FreeVars(\varphi_l).\varphi_l \qquad \models ((\varphi \wedge \varphi_l) \neq \bot_{Cfg}) \wedge \varphi_r \to \varphi' \quad \text{for each } \varphi_l \Rightarrow^{\exists} \varphi_r \in \mathcal{S}}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^{\forall} \varphi'}$$

AXIOM :

$$\frac{\varphi \Rightarrow^{Q} \varphi' \in \mathcal{S} \cup \mathcal{A} \qquad \psi \text{ is FOL formula (logical frame)}}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \wedge \psi \Rightarrow^{Q} \varphi' \wedge \psi}$$

REFLEXIVITY :

$$\frac{\cdot}{\mathcal{S}, \mathcal{A} \vdash \varphi \Rightarrow^{Q} \varphi}$$

TRANSITIVITY :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^{Q} \varphi_2 \qquad \mathcal{S}, \mathcal{A} \cup C \vdash \varphi_2 \Rightarrow^{Q} \varphi_3}{\mathcal{S}, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^{Q} \varphi_3}$$

CONSEQUENCE :

$$\models \varphi_1 \to \varphi_1' \qquad \mathcal{S}, \mathcal{A} \vdash_C \varphi_1' \Rightarrow^{Q} \varphi_2' \qquad \models \varphi_2' \to \varphi_2$$

CIRCULARITY :

$$\frac{\mathcal{S}, \mathcal{A} \vdash_{C \cup \{\varphi \Rightarrow^{Q} \varphi'\}} \varphi \Rightarrow^{Q} \varphi'}{\mathcal{S}, \mathcal{A} \vdash_C \varphi \Rightarrow^{Q} \varphi'}$$

# Language-independent verification framework

Operational semantics
(C/Java/JavaScript semantics)

Reachability properties
(Functional correctness of heap manipulations)

Language-independent uniform notation
(Reachability logic)

Language-independent proof systems
(Reachability logic proof systems)

Proof automation
(Symbolic execution, SMT, Natural proofs, ...)

# Proof automation

- Deductive verification
- Symbolic execution for reachability space search
- Domain reasoning (e.g., integers, bit-vectors, floats, set, sequences, ...) using SMT
- Natural proofs technique for quantifier instantiation for recursive heap predicates (e.g., list, tree, ...)

# Language-independent verification framework

Operational semantics
(C/Java/JavaScript
semantics)

Reachability properties
(Functional correctness of
heap manipulations)

Language-independent uniform notation
(Reachability logic)

Language-independent proof systems
(Reachability logic proof systems)

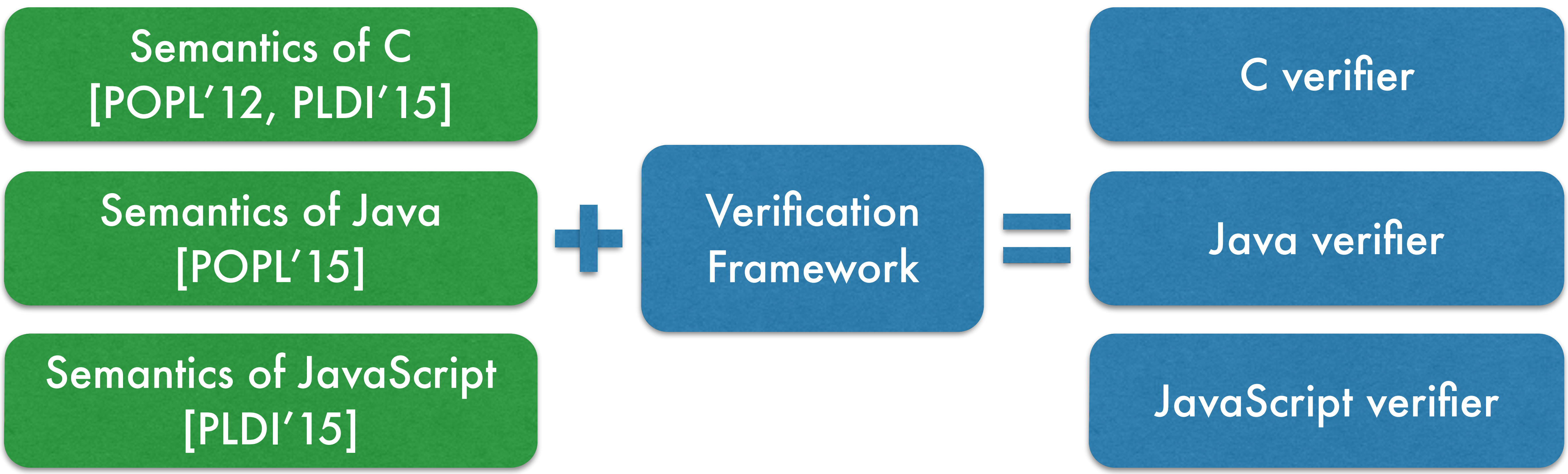
Proof automation
(Symbolic execution, SMT, Natural proofs, ...)

Does it really work?

- Q1: How easy to instantiate the framework?
- Q2: Is performance OK?

# Evaluation

- Instantiated framework by plugging-in three language semantics.




- Verified challenging heap-manipulating programs implementing the same algorithms in all three languages.

# Efforts

| | C | JAVA | JAVASCRIPT |
|---|---|---|---|
| Language-specific effort (days) | 7 | 4 | 5 |
| Semantics changes size (#rules) | 63 | 38 | 12 |
| Semantics changes size (LOC) | 468 | 95 | 49 |

instantiating framework
(additional effort)

Instantiating efforts include:

- Fixing bugs of semantics
- Specifying heap abstractions (e.g., lists and trees)

# Efforts

| | C | JAVA | JAVASCRIPT |
|---|---|---|---|
| Semantics development (months) | 40 | 20 | 4 |
| Semantics size (#rules) | 2,572 | 1,587 | 1,378 |
| Semantics size (LOC) | 17,791 | 13,417 | 6,821 |

defining semantics (already given)

| | | | |
|---|---|---|---|
| Language-specific effort (days) | 7 | 4 | 5 |
| Semantics changes size (#rules) | 63 | 38 | 12 |
| Semantics changes size (LOC) | 468 | 95 | 49 |

instantiating framework (additional effort)

Instantiating efforts include:

- Fixing bugs of semantics
- Specifying heap abstractions (e.g., lists and trees)

# Experiments

Time (secs)

| Programs | C | Java | JS |
|---|---|---|---|
| BST find | 14.0 | 4.7 | 6.3 |
| BST insert | 30.2 | 8.6 | 8.2 |
| BST delete | 71.7 | 24.9 | 21.2 |
| AVL find | 13.0 | 4.9 | 6.4 |
| AVL insert | 281.3 | 105.2 | 135.0 |
| AVL delete | 633.7 | 271.6 | 239.6 |
| RBT find | 14.5 | 5.0 | 6.8 |
| RBT insert | 903.8 | 115.6 | 114.5 |
| RBT delete | 1,902.1 | 171.2 | 183.6 |

| Programs | C | Java | JS |
|---|---|---|---|
| Treap find | 14.4 | 4.9 | 6.5 |
| Treap insert | 67.7 | 23.1 | 18.9 |
| Treap delete | 90.4 | 28.4 | 33.2 |
| List reverse | 11.4 | 4.1 | 5.5 |
| List append | 14.8 | 7.3 | 5.3 |
| Bubble sort | 66.4 | 38.8 | 31.3 |
| Insertion sort | 61.9 | 31.1 | 44.8 |
| Quick sort | 79.2 | 47.1 | 48.1 |
| Merge sort | 170.6 | 87.0 | 66.0 |

| | | | |
|---|---|---|---|
| Total | 4,441.1 | 983.5 | 981.2 |
| Average | 246.7 | 54.6 | 54.5 |

# Experiments

Time (secs)

| Programs | C | Java | JS |
|---|---|---|---|
| BST find | 14.0 | 4.7 | 6.3 |
| BST insert | 30.2 | 8.6 | 8.2 |
| RBT delete | 1,902.1 | 171.2 | 183.6 |

| Programs | C | Java | JS |
|---|---|---|---|
| Treap find | 14.4 | 4.9 | 6.5 |
| Treap insert | 67.7 | 23.1 | 18.9 |
| Merge sort | 170.6 | 87.0 | 66.0 |

| | | | |
|---|---|---|---|
| Total | 4,441.1 | 983.5 | 981.2 |
| Average | 246.7 | 54.6 | 54.5 |

Full functional correctness:

```
  insert /\ bst(t)
=>
     .   /\ bst(t')
         /\ keys(t') == keys(t) \union { v }
```

# Experiments

Time (secs)

| Programs | C | Java | JS |
|---|---|---|---|
| BST find | 14.0 | 4.7 | 6.3 |
| BST insert | 30.2 | 8.6 | 8.2 |
| BST delete | 71.7 | 24.9 | 21.2 |
| AVL find | 13.0 | 4.9 | 6.4 |
| AVL insert | $\boxed{281.3}$ | 105.2 | 135.0 |

| Programs | C | Java | JS |
|---|---|---|---|
| Treap find | 14.4 | 4.9 | 6.5 |
| Treap insert | 67.7 | 23.1 | 18.9 |
| Treap delete | 90.4 | 28.4 | 33.2 |
| List reverse | 11.4 | 4.1 | 5.5 |
| List append | 14.8 | 7.3 | 5.3 |
| Total | 4,441.1 | 983.5 | 981.2 |
| Average | 246.7 | 54.6 | 54.5 |

Performance is comparable to a state-of-the-art verifier for C, VCDryad [PLDI'14], based on a separation logic extension of VCC:
e.g., AVL insert : 260s vs 280s (ours)

# Idea: separation of concerns

Semantics Reasoning

Proof Search

Language semantics:

- C (c11, gcc, clang, …)
- Java (6, 7, 8, …)
- JavaScript (ES5, ES6, …)
- …

VCC

JPF

CBMC

Verification techniques:

- Deductive verification
- Model checking
- Abstract interpretation
- …

Defined/implemente

https://github.com/kframework/k

# Language-independent verification framework

Semantics $\models$ Program & Properties

Language-independent uniform notation (logic)

Language-independent proof systems $\vdash$

Proof automation

- Provides a nice interface (logic) in which both language semantics and program properties can be described.

comes completely language-

# Evaluation

- Instantiated framework by plugging-in three language semantics.

Semantics of C [POPL'12, PLDI'15]

Semantics of Java [POPL'15]

Semantics of JavaScript [PLDI'15]

+ Verification Framework =

C verifier

Java verifier

JavaScript verifier

- Verified challenging heap-manipulating programs implementing the same algorithms in all three languages.

# Experiments

Time (secs)

| Programs | C | Java | JS |
|---|---|---|---|
| BST find | 14.0 | 4.7 | 6.3 |
| BST insert | 30.2 | 8.6 | 8.2 |
| BST delete | 71.7 | 24.9 | 21.2 |
| AVL find | 13.0 | 4.9 | 6.4 |
| AVL insert | 281.3 | 105.2 | 135.0 |
| AVL delete | 633.7 | 271.6 | 239.6 |
| RBT find | 14.5 | 5.0 | 6.8 |
| RBT insert | 903.8 | 115.6 | 114.5 |
| RBT delete | 1,902.1 | 171.2 | 183.6 |

| Programs | C | Java | JS |
|---|---|---|---|
| Treap find | 14.4 | 4.9 | 6.5 |
| Treap insert | 67.7 | 23.1 | 18.9 |
| Treap delete | 90.4 | 28.4 | 33.2 |
| List reverse | 11.4 | 4.1 | 5.5 |
| List append | 14.8 | 7.3 | 5.3 |
| Bubble sort | 66.4 | 38.8 | 31.3 |
| Insertion sort | 61.9 | 31.1 | 44.8 |
| Quick sort | 79.2 | 47.1 | 48.1 |
| Merge sort | 170.6 | 87.0 | 66.0 |
| Total | 4,441.1 | 983.5 | 981.2 |
| Average | 246.7 | 54.6 | 54.5 |