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

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

VCC incorrectly reported an overflow error
Missing details of language behaviors

```c
1  int assign(int *p, int x)
2    _(ensures *p == x)
3    _(writes p)
4  {
5    return (*p = x);
6  }
7
8  void main() {
9    int r;
10   assign(&r, 0) == assign(&r, 1);
11   _(assert r == 1)
12  }
```

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

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

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
• 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.

  \[ \varphi \implies \varphi' \]

  reachability between "patterns"

  "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:

```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:

```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:

```plaintext
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:

```plaintext
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:

```plaintext
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:

```plaintext
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:
\[ \vdash \varphi \rightarrow \forall \varphi' \in S \exists \text{FreeVars}(\varphi) \cdot \varphi' \]
\[ \vdash ((\varphi \land \varphi') \neq \bot_{\text{cfg}}) \land \varphi \rightarrow \varphi' \quad \text{for each } \varphi \Rightarrow \exists \varphi' \in S \]
\[ S, \mathcal{A} \vdash \varphi \Rightarrow \varphi' \]

Axiom:
\[ \varphi \Rightarrow^0 \varphi' \in S \cup \mathcal{A} \quad \psi \text{ is FOL formula (logical frame)} \]
\[ S, \mathcal{A} \vdash \varphi \land \psi \Rightarrow^0 \varphi' \land \psi \]

Reflexivity:
\[ S, \mathcal{A} \vdash \varphi \Rightarrow^0 \varphi \]

Transitivity:
\[ S, \mathcal{A} \vdash \varphi_1 \Rightarrow^0 \varphi_2 \quad S, \mathcal{A} \cup C \vdash \varphi_2 \Rightarrow^0 \varphi_3 \]
\[ S, \mathcal{A} \vdash \varphi_1 \Rightarrow^0 \varphi_3 \]

Consequence:
\[ \vdash \varphi_1 \rightarrow \varphi'_1 \quad S, \mathcal{A} \vdash \varphi'_1 \Rightarrow^0 \varphi'_2 \quad \vdash \varphi'_2 \rightarrow \varphi_2 \]
\[ S, \mathcal{A} \vdash \varphi_1 \Rightarrow^0 \varphi_2 \]

Case Analysis:
\[ S, \mathcal{A} \vdash \varphi_1 \Rightarrow^0 \varphi \quad S, \mathcal{A} \vdash \varphi_2 \Rightarrow^0 \varphi \]
\[ S, \mathcal{A} \vdash \varphi_1 \lor \varphi_2 \Rightarrow^0 \varphi \]

Abstraction:
\[ S, \mathcal{A} \vdash \varphi \Rightarrow^0 \varphi' \quad X \cap \text{FreeVars}(\varphi') = \emptyset \]
\[ S, \mathcal{A} \vdash \exists X \varphi \Rightarrow^0 \varphi' \]

Circularity:
\[ S, \mathcal{A} \vdash \cup_{\varphi' \Rightarrow^0 \varphi} \varphi \Rightarrow^0 \varphi' \]
\[ S, \mathcal{A} \vdash \varphi \Rightarrow^0 \varphi' \]
Proof system

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

\[
\begin{align*}
\varphi_1 & \Rightarrow \varphi_1' \\
\varphi_2 & \Rightarrow \varphi_2' \\
\varphi_3 & \Rightarrow \varphi_3' \\
& \vdots
\end{align*}
\]

Semantics

Property

Step:

\[
\frac{
\vdash \varphi \Rightarrow \bigvee \varphi_i \Rightarrow \exists \varphi_r \in S \exists \text{FreeVars}(\varphi_i) \varphi_r
\quad
\vdash ((\varphi \land \varphi_i) \neq \bot_{\text{cfg}}) \land \varphi_r \Rightarrow \varphi'
\quad
\text{for each } \varphi_i \Rightarrow \exists \varphi_r \in S
\quad
}{S, \mathcal{A} \vdash_C \varphi \Rightarrow \varphi'}
\]

Axiom:

\[
\varphi \Rightarrow^Q \varphi' \in S \cup \mathcal{A} \quad \psi \text{ is FOL formula (logical frame)}
\]

Reflexivity:

\[
S, \mathcal{A} \vdash \varphi \Rightarrow^Q \varphi
\]

Transitivity:

\[
S, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi_2 \quad S, \mathcal{A} \cup \mathcal{C} \vdash \varphi_2 \Rightarrow^Q \varphi_3
\]

\[
S, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi_3
\]

Consequence:

\[
\frac{
\vdash \varphi_1 \Rightarrow \varphi_1' \\
S, \mathcal{A} \vdash_C \varphi_1' \Rightarrow^Q \varphi_2' \\
\vdash \varphi_2' \Rightarrow \varphi_2
\quad
}{S, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi_2}
\]

Case Analysis:

\[
S, \mathcal{A} \vdash_C \varphi_1 \Rightarrow^Q \varphi \\
S, \mathcal{A} \vdash_C \varphi_2 \Rightarrow^Q \varphi
\]

\[
S, \mathcal{A} \vdash_C \varphi_1 \lor \varphi_2 \Rightarrow^Q \varphi
\]

Abstraction:

\[
S, \mathcal{A} \vdash_C \varphi \Rightarrow^Q \varphi' \\
X \cap \text{FreeVars}(\varphi') = \emptyset
\]

\[
S, \mathcal{A} \vdash_C \exists X \varphi \Rightarrow^Q \varphi'
\]

Circularity:

\[
S, \mathcal{A} \vdash_C \exists \varphi \Rightarrow^Q \varphi
\]

\[
S, \mathcal{A} \vdash_C \varphi \Rightarrow^Q \varphi'
\]
Language-independent proof system

for deriving sequents of the form:

\[ \phi_1 \Rightarrow \phi'_1 \]
\[ \phi_2 \Rightarrow \phi'_2 \quad \vdash \quad \phi \Rightarrow \phi' \]
\[ \phi_3 \Rightarrow \phi'_3 \]
\[ \vdots \]

Step:
\[ \models \varphi \rightarrow \exists \forall \varphi_r \in S \exists \text{FreeVars}(\varphi_l). \varphi_l \]
\[ \models ((\varphi \land \varphi_l) \neq \perp) \land \varphi_r \rightarrow \varphi' \quad \text{for each } \varphi_l \Rightarrow \exists \varphi_r \in S \]
\[ S, A \vdash_C \varphi \Rightarrow \varphi' \]

Axiom:
\[ \varphi \Rightarrow^0 \varphi' \in S \cup A \quad \psi \text{ is FOL formula (logical frame)} \]
\[ S, A \vdash_C \varphi \land \psi \Rightarrow^0 \varphi' \land \psi \]

Reflexivity:
\[ S, A \vdash \varphi \Rightarrow^0 \varphi \]

Transitivity:
\[ S, A \vdash_C \varphi_1 \Rightarrow^0 \varphi_2 \quad S, A \cup C \vdash \varphi_2 \Rightarrow^0 \varphi_3 \]
\[ S, A \vdash_C \varphi_1 \Rightarrow^0 \varphi_3 \]

Consequence:
\[ \models \varphi_1 \rightarrow \varphi', \quad S, A \vdash_C \varphi' \Rightarrow^0 \varphi \quad \models \varphi' \rightarrow \varphi_2 \]

Circularity:
\[ S, A \vdash_{C \cup \varphi \Rightarrow^0 \varphi'} \varphi \Rightarrow^0 \varphi' \]
\[ S, A \vdash_C \varphi \Rightarrow^0 \varphi' \]
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.

Semantics of C
[POPL’12, PLDI’15] + Verification Framework = C verifier

Semantics of Java
[POPL’15] + Verification Framework = Java verifier

Semantics of JavaScript
[PLDI’15] + Verification Framework = JavaScript verifier

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

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>JAVA</th>
<th>JAVA\textsc{Script}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language-specific effort (days)</td>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Semantics changes size (#rules)</td>
<td>63</td>
<td>38</td>
<td>12</td>
</tr>
<tr>
<td>Semantics changes size (LOC)</td>
<td>468</td>
<td>95</td>
<td>49</td>
</tr>
</tbody>
</table>

Instantiating efforts include:

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

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>JAVA</th>
<th>JAVASCRIPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantics development (months)</td>
<td>40</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>Semantics size (#rules)</td>
<td>2,572</td>
<td>1,587</td>
<td>1,378</td>
</tr>
<tr>
<td>Semantics size (LOC)</td>
<td>17,791</td>
<td>13,417</td>
<td>6,821</td>
</tr>
<tr>
<td>Language-specific effort (days)</td>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Semantics changes size (#rules)</td>
<td>63</td>
<td>38</td>
<td>12</td>
</tr>
<tr>
<td>Semantics changes size (LOC)</td>
<td>468</td>
<td>95</td>
<td>49</td>
</tr>
</tbody>
</table>

### Instantiating efforts include:
- Fixing bugs of semantics
- Specifying heap abstractions (e.g., lists and trees)
## Experiments

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BST find</td>
<td>14.0</td>
<td>4.7</td>
<td>6.3</td>
</tr>
<tr>
<td>BST insert</td>
<td>30.2</td>
<td>8.6</td>
<td>8.2</td>
</tr>
<tr>
<td>BST delete</td>
<td>71.7</td>
<td>24.9</td>
<td>21.2</td>
</tr>
<tr>
<td>AVL find</td>
<td>13.0</td>
<td>4.9</td>
<td>6.4</td>
</tr>
<tr>
<td>AVL insert</td>
<td>281.3</td>
<td>105.2</td>
<td>135.0</td>
</tr>
<tr>
<td>AVL delete</td>
<td>633.7</td>
<td>271.6</td>
<td>239.6</td>
</tr>
<tr>
<td>RBT find</td>
<td>14.5</td>
<td>5.0</td>
<td>6.8</td>
</tr>
<tr>
<td>RBT insert</td>
<td>903.8</td>
<td>115.6</td>
<td>114.5</td>
</tr>
<tr>
<td>RBT delete</td>
<td>1,902.1</td>
<td>171.2</td>
<td>183.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treap find</td>
<td>14.4</td>
<td>4.9</td>
<td>6.5</td>
</tr>
<tr>
<td>Treap insert</td>
<td>67.7</td>
<td>23.1</td>
<td>18.9</td>
</tr>
<tr>
<td>Treap delete</td>
<td>90.4</td>
<td>28.4</td>
<td>33.2</td>
</tr>
<tr>
<td>List reverse</td>
<td>11.4</td>
<td>4.1</td>
<td>5.5</td>
</tr>
<tr>
<td>List append</td>
<td>14.8</td>
<td>7.3</td>
<td>5.3</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>66.4</td>
<td>38.8</td>
<td>31.3</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>61.9</td>
<td>31.1</td>
<td>44.8</td>
</tr>
<tr>
<td>Quick sort</td>
<td>79.2</td>
<td>47.1</td>
<td>48.1</td>
</tr>
<tr>
<td>Merge sort</td>
<td>170.6</td>
<td>87.0</td>
<td>66.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>4,441.1</td>
<td>983.5</td>
<td>981.2</td>
</tr>
<tr>
<td>Average</td>
<td>246.7</td>
<td>54.6</td>
<td>54.5</td>
</tr>
</tbody>
</table>
# Experiments

## Time (secs)

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BST find</td>
<td>14.0</td>
<td>4.7</td>
<td>6.3</td>
</tr>
<tr>
<td>BST insert</td>
<td>30.2</td>
<td>8.6</td>
<td>8.2</td>
</tr>
<tr>
<td>Treap find</td>
<td>14.4</td>
<td>4.9</td>
<td>6.5</td>
</tr>
<tr>
<td>Treap insert</td>
<td>67.7</td>
<td>23.1</td>
<td>18.9</td>
</tr>
<tr>
<td>RBT delete</td>
<td>1,902.1</td>
<td>171.2</td>
<td>183.6</td>
</tr>
<tr>
<td>Merge sort</td>
<td>170.6</td>
<td>87.0</td>
<td>66.0</td>
</tr>
</tbody>
</table>

**Total**

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4,441.1</td>
<td>983.5</td>
<td>981.2</td>
</tr>
</tbody>
</table>

**Average**

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>246.7</td>
<td>54.6</td>
<td>54.5</td>
</tr>
</tbody>
</table>

---

**Full functional correctness:**

\[
\text{insert} \setminus \text{bst}(t) \\
\Rightarrow \\
\setminus \text{bst}(t') \setminus \text{keys}(t') = \text{keys}(t) \cup \{ v \} \]
## Experiments

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BST find</td>
<td>14.0</td>
<td>4.7</td>
<td>6.3</td>
</tr>
<tr>
<td>BST insert</td>
<td>30.2</td>
<td>8.6</td>
<td>8.2</td>
</tr>
<tr>
<td>BST delete</td>
<td>71.7</td>
<td>24.9</td>
<td>21.2</td>
</tr>
<tr>
<td>AVL find</td>
<td>13.0</td>
<td>4.9</td>
<td>6.4</td>
</tr>
<tr>
<td>AVL insert</td>
<td>281.3</td>
<td>105.2</td>
<td>135.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treap find</td>
<td>14.4</td>
<td>4.9</td>
<td>6.5</td>
</tr>
<tr>
<td>Treap insert</td>
<td>67.7</td>
<td>23.1</td>
<td>18.9</td>
</tr>
<tr>
<td>Treap delete</td>
<td>90.4</td>
<td>28.4</td>
<td>33.2</td>
</tr>
<tr>
<td>List reverse</td>
<td>11.4</td>
<td>4.1</td>
<td>5.5</td>
</tr>
<tr>
<td>List append</td>
<td>14.8</td>
<td>7.3</td>
<td>5.3</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>4,441.1</td>
<td>983.5</td>
<td>981.2</td>
</tr>
<tr>
<td>Average</td>
<td>246.7</td>
<td>54.6</td>
<td>54.5</td>
</tr>
</tbody>
</table>
Idea: separation of concerns

Language semantics:
- C (c11, gcc, clang, ...)
- Java (6, 7, 8, ...)
- JavaScript (ES5, ES6, ...)
- ...

Verification techniques:
- Deductive verification
- Model checking
- Abstract interpretation
- ...

Defined/implemented: https://github.com/KFramework/K

Evaluation

- Instantiated framework by plugging-in three language semantics.

Semantics of C [POPL’12, PLDI’15] + Verification Framework = C verifier

Semantics of Java [POPL’15] + Verification Framework = Java verifier


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

Language-independent verification framework

- Provides a nice interface (logic) in which both language semantics and program properties can be described.
- Verifying and Reasoning about Programs—mechanical verification of the generated program verifiers can check automatically the specifications as reachability rules between matching logic patterns, and uses the sound and relatively complete reachability logic proof system to prove the specifications using deductive verification.

Language-independent uniform notation (logic)

Language-independent proof systems

Proof automation

- Verification techniques:
  - Deductive verification
  - Model checking
  - Abstract interpretation

Experiments

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BST find</td>
<td>14.0</td>
<td>4.7</td>
<td>6.3</td>
</tr>
<tr>
<td>BST insert</td>
<td>30.2</td>
<td>8.6</td>
<td>8.2</td>
</tr>
<tr>
<td>BST delete</td>
<td>71.7</td>
<td>24.9</td>
<td>21.2</td>
</tr>
<tr>
<td>AVL find</td>
<td>13.0</td>
<td>4.9</td>
<td>6.4</td>
</tr>
<tr>
<td>AVL insert</td>
<td>281.3</td>
<td>105.2</td>
<td>135.0</td>
</tr>
<tr>
<td>AVL delete</td>
<td>633.7</td>
<td>271.6</td>
<td>239.6</td>
</tr>
<tr>
<td>RBT find</td>
<td>14.5</td>
<td>5.0</td>
<td>6.8</td>
</tr>
<tr>
<td>RBT insert</td>
<td>903.8</td>
<td>115.6</td>
<td>114.5</td>
</tr>
<tr>
<td>RBT delete</td>
<td>1,902.1</td>
<td>171.2</td>
<td>183.6</td>
</tr>
</tbody>
</table>

Time (secs)

<table>
<thead>
<tr>
<th>Programs</th>
<th>C</th>
<th>Java</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treap find</td>
<td>14.4</td>
<td>4.9</td>
<td>6.5</td>
</tr>
<tr>
<td>Treap insert</td>
<td>67.7</td>
<td>23.1</td>
<td>18.9</td>
</tr>
<tr>
<td>Treap delete</td>
<td>90.4</td>
<td>28.4</td>
<td>33.2</td>
</tr>
<tr>
<td>List reverse</td>
<td>11.4</td>
<td>4.1</td>
<td>5.5</td>
</tr>
<tr>
<td>List append</td>
<td>14.8</td>
<td>7.3</td>
<td>5.3</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>66.4</td>
<td>38.8</td>
<td>31.3</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>61.9</td>
<td>31.1</td>
<td>44.8</td>
</tr>
<tr>
<td>Quick sort</td>
<td>79.2</td>
<td>47.1</td>
<td>48.1</td>
</tr>
<tr>
<td>Merge sort</td>
<td>170.6</td>
<td>87.0</td>
<td>66.0</td>
</tr>
</tbody>
</table>

Total 4,441.1 983.5 981.2
Average 246.7 54.6 54.5