Interpretation-based Violation Witness Validation for C code

4th International Workshop on CPAchecker

Philipp Berger, Jan Svejda and Joost-Pieter Katoen
Witnesses

- similar to certificates in complexity theory \(\Rightarrow\) they should ease verification

- two types are currently in use: violation and correctness witnesses

- violation witnesses (can) restrict the state-space, (can) provide resolution of non-determinism and (can) guide the search towards an error location
Motivation

Situation: testing model checkers on complex and large industry-sourced C code

- Conflicting results - now what?
  - Back then only two violation witness verifiers existed
  - We observed: if the tools can not verify the property alone, it can not verify the witness
  - Very time and memory intensive

- Scenario: property violation detected by CBMC, correctness established by CPAchecker
  - Issue: witnesses by CBMC often not accepted by CPAchecker nor Ultimate (format)

- Scenario: CPAchecker and Ultimate Automizer do not agree
  - Issue: When Ultimate and CPAchecker do not agree - do I trust their witness verification?

- Some of the tools output a significant amount of unknown results
The NITWIT Validator

iNterpretation-based vIolaTion WITness Validator

It:
- is a new execution-based validator,
- explores only a single path through the program,
- is fast and memory efficient and
- is applicable to a large variety of C programs and witnesses from different verifiers.
The NITWIT Validator

Interpretation-based violation witness validator

It:

- is a new execution-based validator,

- explores only a single path through the program,

- is fast and memory efficient and

- is applicable to a large variety of C programs and witnesses from different verifiers.
The NITWIT Validator - Implementation A

- The interpreter explores a path while simultaneously feeding control flow locations and variable values into an internal representation of the witness automaton.
- The witness automaton in turn resolves nondeterministic variables if appropriate assumptions in the witness are available.
The NITWIT Validator - Implementation B

Checking assumptions: Easy with interpreters!

1. Back up the stack and heap,

2. execute the state-space guard in the context of the program,

3. check whether non-deterministic variables were resolved and

4. restore the stack and heap.
Extending PicoC

Most of the work consisted in extending PicoC for our purposes and in adding new features to support the required parts of C99:

- callbacks to witness automaton at important program locations,
- ND types + functions,
- goto constructs,
- function pointers,
- distinguishing double/float types + operations,
- long longs, fixed many bugs in integer arithmetic,
- const types (partially), proper parsing of numerical constants,
- variable shadowing,
- struct initialization,
- bit fields
- …
Results on the SV-COMP ’19 dataset
 (including other validators)
NITWIT - A Violation Witness Validator for C code
berger@cs.rwth-aachen.de
berger@cs.rwth-aachen.de
(a) CPAChecker

(b) Ultimate Automizer
(c) CPA-witness2test

(d) FShell-witness2test
<table>
<thead>
<tr>
<th>Succ. validations per verifier</th>
<th>CPAchecker</th>
<th>Ult. Auto.</th>
<th>CPA-w2test</th>
<th>FShell-w2t</th>
<th>Nitwit</th>
<th>Virt. best</th>
<th>In total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBMC</td>
<td>458</td>
<td>323</td>
<td>486</td>
<td>537</td>
<td>577</td>
<td>714</td>
<td>804</td>
</tr>
<tr>
<td>CBMC-Path</td>
<td>110</td>
<td>125</td>
<td>227</td>
<td>240</td>
<td>263</td>
<td>323</td>
<td>331</td>
</tr>
<tr>
<td>CPA-Seq</td>
<td><strong>1038</strong></td>
<td>692</td>
<td>853</td>
<td>431</td>
<td>1008</td>
<td>1109</td>
<td>1123</td>
</tr>
<tr>
<td>DIVINE-SMT</td>
<td><strong>185</strong></td>
<td>15</td>
<td>0</td>
<td>121</td>
<td>84</td>
<td>192</td>
<td>205</td>
</tr>
<tr>
<td>DIVINE-explicit</td>
<td>120</td>
<td>21</td>
<td>97</td>
<td><strong>127</strong></td>
<td>105</td>
<td>141</td>
<td>145</td>
</tr>
<tr>
<td>DepthK</td>
<td>493</td>
<td>80</td>
<td>434</td>
<td><strong>617</strong></td>
<td>614</td>
<td>755</td>
<td>778</td>
</tr>
<tr>
<td>ESBMC-kind</td>
<td>599</td>
<td>76</td>
<td>579</td>
<td><strong>810</strong></td>
<td>736</td>
<td>950</td>
<td>970</td>
</tr>
<tr>
<td>Map2Check</td>
<td>75</td>
<td>32</td>
<td>100</td>
<td><strong>149</strong></td>
<td>115</td>
<td>153</td>
<td>160</td>
</tr>
<tr>
<td>PeSCo</td>
<td><strong>1011</strong></td>
<td>655</td>
<td>785</td>
<td>532</td>
<td>941</td>
<td>1045</td>
<td>1062</td>
</tr>
<tr>
<td>Pinaka</td>
<td>594</td>
<td>420</td>
<td>421</td>
<td><strong>630</strong></td>
<td>622</td>
<td>659</td>
<td>670</td>
</tr>
<tr>
<td>PredatorHP</td>
<td><strong>57</strong></td>
<td>38</td>
<td>13</td>
<td>46</td>
<td>22</td>
<td>68</td>
<td>71</td>
</tr>
<tr>
<td>SMACK</td>
<td><strong>423</strong></td>
<td>13</td>
<td>375</td>
<td>353</td>
<td>395</td>
<td>521</td>
<td>622</td>
</tr>
<tr>
<td>Skink</td>
<td>86</td>
<td>35</td>
<td>92</td>
<td><strong>96</strong></td>
<td>82</td>
<td>104</td>
<td>105</td>
</tr>
<tr>
<td>Symbiotic</td>
<td><strong>801</strong></td>
<td>13</td>
<td>398</td>
<td>455</td>
<td>538</td>
<td>877</td>
<td>1099</td>
</tr>
<tr>
<td>UAutomizer</td>
<td>267</td>
<td><strong>419</strong></td>
<td>214</td>
<td>185</td>
<td>145</td>
<td>471</td>
<td>483</td>
</tr>
<tr>
<td>UKojak</td>
<td>264</td>
<td><strong>276</strong></td>
<td>42</td>
<td>158</td>
<td>132</td>
<td>321</td>
<td>324</td>
</tr>
<tr>
<td>UTaipan</td>
<td>260</td>
<td><strong>380</strong></td>
<td>183</td>
<td>183</td>
<td>141</td>
<td>427</td>
<td>438</td>
</tr>
<tr>
<td>VIAP</td>
<td>44</td>
<td>32</td>
<td>43</td>
<td><strong>82</strong></td>
<td>52</td>
<td>100</td>
<td>112</td>
</tr>
<tr>
<td>VeriAbs</td>
<td>549</td>
<td>295</td>
<td>399</td>
<td>1000</td>
<td><strong>1070</strong></td>
<td>1220</td>
<td>1274</td>
</tr>
<tr>
<td>VeriFuzz</td>
<td>1040</td>
<td>44</td>
<td>877</td>
<td>1085</td>
<td><strong>1202</strong></td>
<td>1262</td>
<td>1271</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8474</strong></td>
<td><strong>3984</strong></td>
<td><strong>6618</strong></td>
<td><strong>7837</strong></td>
<td><strong>8844</strong></td>
<td>11412</td>
<td>12047</td>
</tr>
</tbody>
</table>
Result summary

A dataset of 12 047 witnesses in total (all C programs from the ReachSafety category)

- 8844 witnesses validated by us
- Time (only validated): on average 0.79s, median 0.017s, std. deviation 6.04s. In total about 7000s.
- Time (all): on average 1.69s, median 0.017s, std. deviation 11.12. In total about 20 000s.
- Memory (all): on average 21.6 MB, median 18.8 MB, std. deviation: 47.0 MB.
Our contribution

- The NITWIT Validator is open-source and free to use:

  https://github.com/moves-rwth/nitwit-validator

- It is fast, memory efficient and, compared to SV-COMP ’19, finds the most violations for witnesses from the ReachSafety category.

- It was developed independently of existing model checkers.

- It borrows C semantics from your favorite compiler!

- It supports 32-bit and 64-bit programs.

- Users have some more compilation options for control of specific validation details.

- **Planned feature:** witness refinement