Testing versus static analysis of maximum stack size

Mahdi Eslamimehr
Jens Palsberg
Computer Science Department
Dept. Computer Science
University of California, Los Angeles
COMPSAC’13
Talk Outline

• **Introduction**
  – Problem Definition and Previous Work
  – Classical Software Testing VS. Static Analysis
  – Event-Driven Software Testing and Challenges
  – Quick Review of Directed Testing

• **Introducing 7 Testing Approaches**
  – Example
  – VICE: Algorithms and tools

• **Experiment Results**

• **Conclusion**
  – *Event-Based Directed Testing* could compete with Static Analysis
Maximum Stack Size

• Stack size in resource-constrained device.
  – Underestimation causes stack overflow.
  – Overestimation is waste of expensive memory.

• **Question**: what is the exact maximum stack size across all inputs?

  1. Program $P, K$ is the maximum size of all executions of $P$, if $P$’s stack size never grows beyond $K$.

  2. There is a possible schedule of events and an execution of the program $P$ such that the stack size becomes $K$. 
Maximum Stack Size Analysis

- **Testing**
  - Random Algorithms:
    - [Brylow *ICSE’01*]
  - Genetic Algorithm:
    - [Regehr *EMSOFT’05*]

- **Static Analysis**
  - [Regehr *EMSOFT’03*], [Chatterjee *SAS’03*], [Regehr *SIGARCH’04*], [Alur *STOC’04*]
Motivating Experiments

Testing VS. Static Analysis

Maximum stack size

Branch coverage
High Angle - Max Stack Size

Tested ≤ Real ≤ Static
High Angle- Branch Coverage
Maximum Stack Size Analysis

• Questions:
  
  – how good is the state-of-the-art static analysis of maximum stack size?
  
  • Big gap between the estimates: testing approach achieves 67% of static maximum stack size in average.

  – Is the gap mostly due to weak testing or overly conservative static analysis?
  
  • better testing is possible, however the static analysis is near optimal.
Event-Driven Software (EDS) & Testing Challenges

• EDS:
  – Takes internal/external events (commands, messages) as input (from operating environment or other applications)
  – Process events and changes its state
    • Sometimes outputs an event sequence.
  Examples: Sensor Networks, Web Servers, GUIs.
Testing Event Driven Software

• Classical software:
  – tester only devices a suite of single inputs.

• Event-Driven software (with real-time behavior):
  – tester must device a suite of event sequences.
    – In each sequence: # of events, types of events, values associated with the events e.g. registers’ value, and timing of events.

• **Challenge**: Quickly generate a small number of challenging event sequences
Evolution of Sequential Software Testing

30-35% Random Testing

60-75% Genetic Algorithm

90-100% Directed Testing
Directed Testing

- Generate concrete inputs one by one
  - each input leads program along a different path

- On each input execute program both concretely and symbolically
  - concrete execution guides the symbolic execution
  - concrete execution enables symbolic execution to overcome incompleteness of theorem prover
  - symbolic execution helps to generate concrete input for next execution
    - increases coverage

Input = Random

While all paths covered?

Yes

End

No

Concolically execute and collect path constraints

Solving constraints and generate inputs
Seven Testing Approaches

- Categorized on how generating *event*

  *(event name, event value, wait time)*

<table>
<thead>
<tr>
<th>Approach #</th>
<th>event names</th>
<th>event values</th>
<th>wait times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sample</td>
<td>Sample</td>
<td>Sample</td>
</tr>
<tr>
<td>2</td>
<td>GA</td>
<td>GA</td>
<td>Sample</td>
</tr>
<tr>
<td>3</td>
<td>All</td>
<td>Sample</td>
<td>All</td>
</tr>
<tr>
<td>4</td>
<td>Sample</td>
<td>DT</td>
<td>All</td>
</tr>
<tr>
<td><strong>VICE: 5</strong></td>
<td><strong>SA-Tree</strong></td>
<td><strong>DT</strong></td>
<td><strong>Sample</strong></td>
</tr>
<tr>
<td>6</td>
<td>All</td>
<td>DT</td>
<td>Sample</td>
</tr>
<tr>
<td>Dtall:7</td>
<td>All</td>
<td>DT</td>
<td>all</td>
</tr>
</tbody>
</table>
program Sample {
  entrypoint main = TestMe.main;
  entrypoint timer_comp = testMe.m_intr;
}

component TestMe {
  field sending:bool = false;
  method main(x:int):void {
    computeValue();
    transmitValue(x);
  }
  method computeValue():void { ... }
  method transmitValue(a:int):void {
    local buffer:int, b:int;
    b = rand(100);
    local bufferSize:int = (a+b) * 256;
    if (atomic_swap(sending,true)) return;
    if (a > 2000) {
      buffer = checks(a,b);
      sending = false;
      return;
    }
    m_intr(y:int):void {
      transmitValue(y);
    }
    method checks(s:int, t:int):int {
      if (s==5000) {
        t=square(s);
        if (s<-5) return square(-s);
      }
      return 1;
    }
    method square(root:int):int { ... }
    method rand(seed:int):int { ... }
    method atomic_swap(cur:bool,status:bool):bool {
      ... }
  }
  method m_intr(y:int):void {
    transmitValue(y);
  }
  method checks(s:int, t:int):int {
    if (s==5000) {
      t=square(s);
      if (s<-5) return square(-s);
    }
    return 0;
  }
  method square(root:int):int { ... }
  method rand(seed:int):int { ... }
  method atomic_swap(cur:bool,status:bool):bool {
    ... }
}

method main(x:int):void {
  ... }

method computeValue():void { ... }

method transmitValue(a:int):void {
  local buffer:int, b:int;
  b = rand(100);
  local bufferSize:int = (a+b) * 256;
  if (atomic_swap(sending,true)) return;
  if (a > 2000) {
    buffer = checks(a,b);
    sending = false;
    return;
  }
  m_intr(y:int):void {
    transmitValue(y);
  }
  method checks(s:int, t:int):int {
    if (s==5000) {
      t=square(s);
      if (s<-5) return square(-s);
    }
    return 1;
  }
  method square(root:int):int { ... }
  method rand(seed:int):int { ... }
  method atomic_swap(cur:bool,status:bool):bool {
    ... }
}
Example

• Round 1:

[(main, 673), (m_intr, -8634756), (main, -991), (m_intr, 34)]

Constraints: y=a, a>2000, x=a, a>2000

Max Stack= two event handlers → m_intr → transmitValue → atomic_swap.

Branch Coverage(round1): 25%
Example

• Round 1:
[(main, 673), (m_intr, -8634756), (main, -991), (m_intr, 34)]
Constraints: y=a, a>2000, x=a, a>2000
Max Stack= two event handlers → m_intr → transmitValue → atomic_swap.
Branch Coverage(round1): 25%

• Round 2:
[(main, 2833), (m_intr, 4756), (m_intr, 77733), (main, 6500)]
New constraints: y = x = a = s = 5000
Max Stack= two event handlers → transmitValue → checks
Branch Coverage(round 1,2) = 50%
Example

• Round 1:
  \[(\text{main,673}), (\text{m}_\text{intr}, -8634756), (\text{main}, -991), (\text{m}_\text{intr}, 34)\]
  Constraints: \(y = a, a > 2000, x = a, a > 2000\)
  Max Stack= two event handlers \(\rightarrow\) \(m\_\text{intr} \rightarrow\) \(\text{transmitValue}\) \(\rightarrow\) \(\text{atomic\_swap}\).
  Branch Coverage(round1): 25%

• Round 2:
  \[(\text{main,2833}), (\text{m}_\text{intr,4756}), (\text{m}_\text{intr, 77733}), (\text{main, 6500})\]
  New constraints: \(y = x = a = s = 5000\)
  Max Stack= two event handlers \(\rightarrow\) \(\text{transmitValue}\) \(\rightarrow\) \(\text{checks}\)
  Branch Coverage(round 1, 2)= 50%

• Round 3:
  \[(\text{main,5000}), (\text{m}_\text{intr,5000}); (\text{main,5000}), (\text{m}_\text{intr,5000})\]
  New Constraints: \(s = \text{root} ^ s < -5\)
  Max Stack= two event handlers \(\rightarrow\) \(\text{transmitValue}\) \(\rightarrow\) \(\text{checks}\) \(\rightarrow\) \(\text{square}\)
  Branch Coverage(round 1,2,3)= 75%
Example

• Round 1:
  
  $$[(\text{main,673}), (\text{m\_intr, -8634756}), (\text{main, -991}), (\text{m\_intr, 34})]$$

  Constraints: y = a, a > 2000, x = a, a > 2000

  Max Stack = two event handlers → m\_intr → transmitValue → atomic\_swap.

  Branch Coverage(round1): 25%

• Round 2:
  
  $$[(\text{main,2833}), (\text{m\_intr,4756}), (\text{m\_intr, 77733}), (\text{main, 6500})]$$

  New constraints: y = x = a = s = 5000

  Max Stack = two event handlers → transmitValue → checks

  Branch Coverage(round 1, 2) = 50%

• Round 3:
  
  $$[(\text{main,5000}), (\text{m\_intr,5000}); (\text{main,5000}), (\text{m\_intr,5000})]$$

  New Constraints: s = root ^ s < -5

  Max Stack = two event handlers → transmitValue → checks → square

  Branch Coverage(round 1, 2, 3) = 75%

• Round 4 and 5:
  
  Constraints: Unsolvable both s == 5000 and s < -5

  Max Stack: Already Reached!

  Branch Coverage: No improvement
Event Based Directed Testing (EBDT)

- **Types:**
  - `VirgilProgram` = http://compilers.cs.ucla.edu/virgil
  - `machineCode` = AVR assembly code
  - `eventSequence` = (identifier × int × int)list
  - `constraint` = a Virgil arithmetic or logical expression
  - `nameSequence` = (identifier)list
  - `prefixTree` = a prefix-tree of elements of `nameSequence`
Tools

- **concolic:**
  \((\text{VirgilProgram} \times \text{eventSequence}) \rightarrow (\text{constraints} \times \text{float})\)

- **compiler:**
  \(\text{VirgilProgram} \rightarrow \text{machineCode}\)

- **avrora:**
  \(\text{machineCode} \times \text{eventSequence} \rightarrow \text{int}\)

- **SA-Tree-Gen:**
  \(\text{VirgilProgram} \rightarrow \text{prefixTree}\)

- **random:**
  - \(\text{nameSequence} \times \text{int} \rightarrow \text{event-sequence}\)

- **generator:**
  \(\text{nameSequence} \times \text{int} \times \text{constraint} \rightarrow \text{eventSequence}\)
VICE Illustration
Algorithm

```java
int VICE(VirgilProgram p, int waitTime) {
    prefixTree tree = SA-Tree-Gen(p)
    machineCode code = compiler(p)
    int maxStack = 0
    for each nameSequence ns ∈ tree do {
        int noChange = 0
        float branchCoverage = 0
        eventSequence seq = random(ns, waitTime)
        while (noChange < 2) {
            int ms = avrora(code, seq)
            (constraint × float) (c, bc) = concolic(p, seq)
            seq = generator(ns, waitTime, c)
            if ((ms > maxStack) ∨ (bc > branchCoverage))
                then { maxStack = ms; branchCoverage = bc;
                        noChange = 0 }
            else { noChange = noChange + 1 }
        }
    }
    return maxStack
}
```
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC(Virgil)</th>
<th>LOC(C)</th>
<th>#Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestCon1</td>
<td>329</td>
<td>461</td>
<td>4</td>
</tr>
<tr>
<td>TestCon2</td>
<td>347</td>
<td>528</td>
<td>3</td>
</tr>
<tr>
<td>StackTest1</td>
<td>293</td>
<td>513</td>
<td>2</td>
</tr>
<tr>
<td>StackTest2</td>
<td>251</td>
<td>483</td>
<td>2</td>
</tr>
<tr>
<td>TestUSART</td>
<td>1226</td>
<td>1737</td>
<td>5</td>
</tr>
<tr>
<td>TestSPI</td>
<td>859</td>
<td>1109</td>
<td>3</td>
</tr>
<tr>
<td>TestADC</td>
<td>605</td>
<td>1055</td>
<td>4</td>
</tr>
</tbody>
</table>
Experiment Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestCon1</td>
<td>318</td>
<td>441</td>
<td>417</td>
<td>455</td>
<td>505</td>
<td>506</td>
<td>511</td>
<td>516</td>
</tr>
<tr>
<td>TestCon2</td>
<td>366</td>
<td>612</td>
<td>798</td>
<td>703</td>
<td>846</td>
<td>866</td>
<td>882</td>
<td>894</td>
</tr>
<tr>
<td>StackTest1</td>
<td>421</td>
<td>353</td>
<td>318</td>
<td>619</td>
<td>703</td>
<td>749</td>
<td>958</td>
<td>979</td>
</tr>
<tr>
<td>StackTest2</td>
<td>353</td>
<td>324</td>
<td>390</td>
<td>420</td>
<td>564</td>
<td>564</td>
<td>564</td>
<td>566</td>
</tr>
<tr>
<td>TestUSART</td>
<td>459</td>
<td>481</td>
<td>472</td>
<td>525</td>
<td>664</td>
<td>664</td>
<td>664</td>
<td>665</td>
</tr>
<tr>
<td>TestSPI</td>
<td>490</td>
<td>350</td>
<td>481</td>
<td>490</td>
<td>518</td>
<td>522</td>
<td>529</td>
<td>533</td>
</tr>
<tr>
<td>TestADC</td>
<td>247</td>
<td>306</td>
<td>283</td>
<td>302</td>
<td>306</td>
<td>306</td>
<td>308</td>
<td>310</td>
</tr>
<tr>
<td>% of SA</td>
<td>62</td>
<td>67</td>
<td>71</td>
<td>81</td>
<td>94</td>
<td>95</td>
<td>99</td>
<td>100</td>
</tr>
</tbody>
</table>

Maximum stack sizes in bytes. The last line gives a geometric mean.
Experiment Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestCon1</td>
<td>23</td>
<td>56</td>
<td>61</td>
<td>72</td>
<td>92</td>
<td>93</td>
<td>94</td>
</tr>
<tr>
<td>TestCon2</td>
<td>21</td>
<td>60</td>
<td>78</td>
<td>78</td>
<td>89</td>
<td>89</td>
<td>90</td>
</tr>
<tr>
<td>StackTest1</td>
<td>26</td>
<td>40</td>
<td>73</td>
<td>80</td>
<td>64</td>
<td>71</td>
<td>73</td>
</tr>
<tr>
<td>StackTest2</td>
<td>20</td>
<td>43</td>
<td>69</td>
<td>81</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>TestUSART</td>
<td>23</td>
<td>58</td>
<td>66</td>
<td>85</td>
<td>96</td>
<td>96</td>
<td>96</td>
</tr>
<tr>
<td>TestSPI</td>
<td>32</td>
<td>56</td>
<td>71</td>
<td>75</td>
<td>67</td>
<td>73</td>
<td>75</td>
</tr>
<tr>
<td>TestADC</td>
<td>22</td>
<td>62</td>
<td>69</td>
<td>78</td>
<td>98</td>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>% of (7)</td>
<td>24</td>
<td>53</td>
<td>69</td>
<td>78</td>
<td>85</td>
<td>88</td>
<td>89</td>
</tr>
</tbody>
</table>

Branch coverage in percent. The last line gives a geometric mean.
Experiment Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestCon1</td>
<td>7.2</td>
<td>8.83</td>
<td>281</td>
<td>38</td>
<td>1.53</td>
<td>16</td>
<td>439</td>
<td>0.10</td>
</tr>
<tr>
<td>TestCon2</td>
<td>10.1</td>
<td>3.11</td>
<td>173</td>
<td>29</td>
<td>2.48</td>
<td>45</td>
<td>381</td>
<td>0.12</td>
</tr>
<tr>
<td>StackTest1</td>
<td>12.9</td>
<td>2.55</td>
<td>179</td>
<td>23</td>
<td>0.56</td>
<td>26</td>
<td>307</td>
<td>0.05</td>
</tr>
<tr>
<td>StackTest2</td>
<td>2.9</td>
<td>2.29</td>
<td>165</td>
<td>72</td>
<td>4.13</td>
<td>56</td>
<td>266</td>
<td>0.05</td>
</tr>
<tr>
<td>TestUSART</td>
<td>7.4</td>
<td>3.05</td>
<td>204</td>
<td>43</td>
<td>1.18</td>
<td>16</td>
<td>452</td>
<td>0.32</td>
</tr>
<tr>
<td>TestSPI</td>
<td>4.0</td>
<td>3.11</td>
<td>197</td>
<td>26</td>
<td>0.79</td>
<td>9</td>
<td>393</td>
<td>0.15</td>
</tr>
<tr>
<td>TestADC</td>
<td>3.0</td>
<td>1.37</td>
<td>289</td>
<td>33</td>
<td>0.45</td>
<td>6</td>
<td>444</td>
<td>0.13</td>
</tr>
<tr>
<td>% of (7)</td>
<td>1.6</td>
<td>0.4</td>
<td>55</td>
<td>9</td>
<td>0.3</td>
<td>5</td>
<td>100</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Execution time in minutes. The last line gives a geometric mean.
Experiment Results

Comparison of seven testing approaches
High Angle- Max Stack Size

Tested ≤ Real ≤ Static
High Angle - Branch Coverage

Random

GA

Reachable

All
High Angle- Branch Coverage

Random

GA

VICE

Reachable

All
Conclusion

• Static Analysis reaches to its maturity leve
• DTAll almost matches to Static Analysis
• VICE is very close and two order of magnitude faster than DTAll
• The key to improve testing is applying
  – Directed Testing
  – SA-Tree
• VICE could quickly generates challenging event sequence and can provide higher branch coverage than other state-of-art testing techniques