Efficient Decision Ordering Techniques for SAT-based Test Generation

Mingsong Chen, Xiaoke Qin and Prabhat Mishra

Computer and Information Science and Engineering
University of Florida, USA

March 10, 2010
Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion
Functional validation is a major challenge

- Majority of the SOC fails due to logic errors

Simulation using directed tests is promising
Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion
Test Generation using Model Checking

- **Model Checking**
  - Design is modeled temporal specification, e.g., SMV
  - Desired behaviors in temporal logic properties
  - Property holds, or fails with a counterexample

**Problem:** Test generation is very costly or not possible in many scenarios in the presence of complex SoCs and/or complex properties.

**Approach:** Exploit learning to reduce complexity
- Reduction of TG time & memory requirements
- Enables test generation in complex scenarios
SAT-based Bound Model Checking

- For every finite model and a LTL property $\phi$ there exists $k$ such that: $M \models_k \phi \Rightarrow M \models \phi$

- Test generation needs to consider safety properties

- The safety property $P$ is valid up to cycle $k$ iff $\Omega(k)$ is not satisfiable.

$$\Omega(k) = I(S_0) \land \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \land \bigvee_{i=0}^{k} \neg P(s_i)$$

- If $\Omega(k)$ is satisfiable, then we can get an assignment which can be translated to a test.
DPLL Algorithm

while (1) {
    run_periodic_function();
    if (decide_next_branch()) {
        while (Implication == CONFLICT) {
            blevel = Conflict Backtrack();
            if (blevel < 0)
                return UNSAT;
        }
    } else return SAT;
}

BCP = Implication Number + Conflict Backtrack

Boolean Constraint Propagation (BCP) consumes up to 80% of the time and resources during SAT solving.
The minimal bound is $k$:

$$\text{Save: } \Delta p_1^2 + \Delta p_1^3 + \ldots + \Delta p_1^{k-1} + \ldots + \Delta p_1^k$$

O. Strichman. *Pruning Techniques for the SAT-Based Bounded Model Checking Problems.* CHARME, 2001
Same Design, Different Properties

Benefit:
Original: Red + Blue + Green
Now: Red + (Blue - Δblue) + (Green - Δgreen)
Save: Δblue + Δgreen

Δblue

Δgreen

Promising Observations

- Similar properties have the similar counter-examples (variable assignments).
  - Such important information can be reused.

- Current decision ordering techniques focus on the SAT problem instead of the real design.
  - For example, VSDIS, for each literal $lit$ has a score
  - Initialization
    - $score(lit) = \text{literal count of } lit \text{ in CNF clauses}$
  - Periodical update (not include initialization)
    - $score(lit) = score(lit)/2 + lit\_in\_conflict(lit)$
Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion
Two Similar SAT Problems

SAT 1

SAT 2

Ordering: a, a’, b, b’, c, c’

Ordering: a, a’, b, b’, c, c’

Without Learning, 7 conflicts in SAT2.
**Learning: Bit Value Ordering**

**SAT 1**

- Bit value: $a=1, b=0, c=0$

**Ordering:** $a, a', b, b', c, c'$

**SAT 2**

**Ordering:** $a, a', b', b, c', c$

*With bit value learning, 4 conflicts in SAT2.*
Learning: Variable Ordering

SAT 1

Bit value: $a=1, b=0, c=0$
Variable order: $b > c > a$

Ordering: $a, a', b, b', c, c'$

SAT 2

Ordering: $b', b, c', c, a, a'$

With bit value + variable order learning, 1 conflict in SAT2.
Test Generation Using Our Method

Inputs: a) Formal model, $D$

   b) A cluster of properties $P$ with satisfiable bounds

1. Initialize $varStat$

2. Select the base property $p_1$, and generate $CNF_1$

3. $(assignment_1, test_1) = SAT(CNF_1)$

4. $Test-suite = \{test_1\}$

5. for $i$ is from 2 to the size of $P$

   a) Update $varStat$ using $assignment_{i-1}$
   b) Generate $CNF_i = BMC(D, p_i, bound_i)$
   c) $(assignment_i, test_i) = SAT(CNF_i)$
   d) $Test-suite = Test-suite \cup \{test_i\}$

   endfor

6. Return $Test-suite$
An Illustrative Example with 3 properties

**Approach:** Using the statistics of the counterexamples when checking the properties in a cluster
- Count the number of values ➔ bit value ordering
- Variance of counts of two literals ➔ variable ordering

<table>
<thead>
<tr>
<th>VarStat</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0] V</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>…</td>
</tr>
<tr>
<td>[1] V</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>…</td>
</tr>
</tbody>
</table>

P1: a=0, b=0, c=1, d=1

<table>
<thead>
<tr>
<th>VarStat</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0] V</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>…</td>
</tr>
<tr>
<td>[1] V</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>…</td>
</tr>
</tbody>
</table>

P2: a=0, b=0, c=1, d=0

<table>
<thead>
<tr>
<th>VarStat</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0] V</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>…</td>
</tr>
<tr>
<td>[1] V</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>…</td>
</tr>
</tbody>
</table>

P3: a=0, b=0, c=1, d=?

- score(a) ↑, score(a') ↑
- score(b) ↑, score(b') ↑
- score(c) ↑, score(c') ↑

Predict ordering for P3
Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion
## Case Study: MIPS Processor

<table>
<thead>
<tr>
<th>Property (test)</th>
<th>zChaff (sec)</th>
<th>Conflict Clause Forwarding</th>
<th>Improvement Factor</th>
<th>Decision Ordering</th>
<th>Improvement Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>23.20</td>
<td>23.20</td>
<td>1</td>
<td>23.20</td>
<td>1</td>
</tr>
<tr>
<td>P1</td>
<td>20.73</td>
<td>2.74</td>
<td>7.57</td>
<td>0.18</td>
<td>15.22</td>
</tr>
<tr>
<td>P2</td>
<td>21.33</td>
<td>3.01</td>
<td>7.09</td>
<td>0.15</td>
<td>20.07</td>
</tr>
<tr>
<td>P3</td>
<td>18.03</td>
<td>2.70</td>
<td>6.68</td>
<td>0.29</td>
<td>9.31</td>
</tr>
<tr>
<td>DIV</td>
<td>18.78</td>
<td>18.78</td>
<td>1</td>
<td>18.78</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>23.55</td>
<td>2.72</td>
<td>8.66</td>
<td>0.13</td>
<td>20.92</td>
</tr>
<tr>
<td>P5</td>
<td>18.31</td>
<td>3.60</td>
<td>5.09</td>
<td>0.14</td>
<td>25.71</td>
</tr>
<tr>
<td>P6</td>
<td>18.11</td>
<td>3.72</td>
<td>4.87</td>
<td>0.18</td>
<td>20.67</td>
</tr>
<tr>
<td>FADD</td>
<td>22.90</td>
<td>22.90</td>
<td>1</td>
<td>22.90</td>
<td>1</td>
</tr>
<tr>
<td>P7</td>
<td>16.95</td>
<td>4.46</td>
<td>3.80</td>
<td>0.23</td>
<td>19.39</td>
</tr>
<tr>
<td>P8</td>
<td>18.89</td>
<td>2.71</td>
<td>6.97</td>
<td>0.16</td>
<td>16.94</td>
</tr>
<tr>
<td>P9</td>
<td>19.80</td>
<td>4.70</td>
<td>4.21</td>
<td>0.39</td>
<td>12.05</td>
</tr>
<tr>
<td>MUL</td>
<td>64.21</td>
<td>64.21</td>
<td>1</td>
<td>64.21</td>
<td>1</td>
</tr>
<tr>
<td>P10</td>
<td>59.15</td>
<td>3.36</td>
<td>17.60</td>
<td>0.24</td>
<td>14.00</td>
</tr>
<tr>
<td>P11</td>
<td>59.65</td>
<td>3.85</td>
<td>15.49</td>
<td>0.45</td>
<td>8.56</td>
</tr>
<tr>
<td>P12</td>
<td>73.98</td>
<td>6.28</td>
<td>11.78</td>
<td>0.18</td>
<td>34.89</td>
</tr>
</tbody>
</table>
Case Study: MIPS Processor

**Indications:** Test generation complexity is significantly improved
- Reduction of conflict clauses
- Reduction of implication number
This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Size</th>
<th>zChaff</th>
<th>Conflict Forward</th>
<th>Improvement Factor</th>
<th>Decision Ordering</th>
<th>Improvement Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3</td>
<td>1.18</td>
<td>2.18</td>
<td>0.54</td>
<td>0.70</td>
<td>3.11</td>
</tr>
<tr>
<td>C2</td>
<td>4</td>
<td>14.53</td>
<td>9.53</td>
<td>1.52</td>
<td>0.78</td>
<td>12.22</td>
</tr>
<tr>
<td>C3</td>
<td>8</td>
<td>375.91</td>
<td>170.06</td>
<td>2.21</td>
<td>36.19</td>
<td>4.70</td>
</tr>
<tr>
<td>C4</td>
<td>4</td>
<td>12.98</td>
<td>8.33</td>
<td>1.56</td>
<td>1.24</td>
<td>6.72</td>
</tr>
<tr>
<td>C5</td>
<td>4</td>
<td>7.13</td>
<td>16.88</td>
<td>0.42</td>
<td>1.02</td>
<td>16.55</td>
</tr>
<tr>
<td>C6</td>
<td>8</td>
<td>720.13</td>
<td>474.68</td>
<td>1.52</td>
<td>28.60</td>
<td>16.60</td>
</tr>
<tr>
<td>C7</td>
<td>4</td>
<td>10.80</td>
<td>24.55</td>
<td>0.44</td>
<td>1.95</td>
<td>12.59</td>
</tr>
<tr>
<td>C8</td>
<td>8</td>
<td>656.95</td>
<td>321.14</td>
<td>2.05</td>
<td>77.65</td>
<td>4.14</td>
</tr>
<tr>
<td>C9</td>
<td>8</td>
<td>248.17</td>
<td>82.42</td>
<td>3.01</td>
<td>37.93</td>
<td>2.17</td>
</tr>
</tbody>
</table>

Average | - | 227.53 | 123.21 | 1.85 | 20.67 | 5.97 |
Conclusions

- Functional validation is a major bottleneck
  - SAT-based approaches are promising for automated test generation.

- Proposed an efficient technique for generation of directed tests using learning techniques
  - Developed a novel decision ordering technique using both bit-value ordering and variable ordering

- Successfully applied on both hardware and software designs
  - Significant reduction in overall validation effort
Thank you!