Decision Ordering Based Property Decomposition for Functional Test Generation

Mingsong Chen  
Software Engineering Institute  
East China Normal University, China

Prabhat Mishra  
CISE Department  
University of Florida, USA
Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion
Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - 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 Functional Validation
- **Test Generation using Model Checking**
- **Test Generation using SAT-based BMC**

Test Generation using Decision Ordering
- Learning-oriented property decomposition
- Decision ordering based learning techniques
- Test generation using our methodology

Experiments

Conclusion
Directed test generation based on the automation of model checking techniques.
Test Generation using Model Checking

- Model Checking
  - Designs are in formal specifications, 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 some learning to reduce complexity
- Reduction of TG time & memory requirements
- Enables test generation in complex scenarios
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.
Given a $\varphi$ in CNF: $(x+y+z)(\neg x+y)(\neg y+z)(\neg x+\neg y+\neg z)$

Best decision: $\neg x, z$
Same Design, Different Properties

Benefit:
Original: Red + Blue + Green

Now: Red + (Blue – $\Delta_{\text{blue}}$) + (Green – $\Delta_{\text{green}}$)

Save: $\Delta_{\text{blue}}$ + $\Delta_{\text{green}}$

Problem: There is no learning for P1?

Koo et al. *Functional Test Generation using Property Decomposition Techniques*. ACM *TECS, 2009*
Promising Observations

- Sub-properties may have a large overlap in counter-examples (variable assignments) with original property.
  - Such important information can be reused as a kind of decision ordering.

- The learning from sub-properties can drastically reduce the overall test generation time.
  - The SAT instance for sub-properties can be much smaller than that of original property
  - The learning from sub-properties can drastically accelerate the falsification of original property.
Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion
Spatial Property Decomposition

Learn from the sub-properties with smaller COI.

COI(p1) < COI(p2) < COI(p3) < COI(P)

Time(p1) < Time(p2) < Time(p3) < Time(P)

Learning from P1 can reduce the Time(P)?
A MIPS Processor Example

Checked Property

P: The units MUL5 and FADD3 can be activated together at 8th clock cycle.

LTL: ! F(MUL5=active & FADD3=active & clk=8)
Checked sub Property

P1: The units MUL5 can be activated at 8th clock cycle.

LTL: \(!F(MUL5=active \& clk=8)\)

Counterexample for P1

<table>
<thead>
<tr>
<th>Cycles</th>
<th>P1’s test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOP</td>
</tr>
<tr>
<td>2</td>
<td>MUL R2, R2, R0</td>
</tr>
<tr>
<td>3</td>
<td>NOP</td>
</tr>
<tr>
<td>4</td>
<td>NOP</td>
</tr>
</tbody>
</table>
Checked sub Property
P2: The units FADD3 can be activated at 8th clock cycle.

LTL: !F(FADD3=active & clk=8)

Counterexample for P2

<table>
<thead>
<tr>
<th>Cycles</th>
<th>P2’s test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOP</td>
</tr>
<tr>
<td>2</td>
<td>NOP</td>
</tr>
<tr>
<td>3</td>
<td>NOP</td>
</tr>
<tr>
<td>4</td>
<td>FADD R1, R1, R0</td>
</tr>
</tbody>
</table>
Learning from Spatial Property Decomposition

Countereexample for P2 guided by P1

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOP</td>
</tr>
<tr>
<td>2</td>
<td>MUL R2, R2, R0</td>
</tr>
<tr>
<td>3</td>
<td>NOP</td>
</tr>
<tr>
<td>4</td>
<td>FADD R1, R1, R0</td>
</tr>
</tbody>
</table>

Countereexample for P

<table>
<thead>
<tr>
<th>Cycles</th>
<th>Learnings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOP</td>
</tr>
<tr>
<td>2</td>
<td>MUL R2, R2, R0</td>
</tr>
<tr>
<td>3</td>
<td>NOP</td>
</tr>
<tr>
<td>4</td>
<td>FADD R1, R1, R0</td>
</tr>
</tbody>
</table>
Temporal Decomposition

Cause effect relation: $e_1 \rightarrow e_2 \quad e_3 \rightarrow e_4 \quad e_5 \rightarrow e_6$

Happen before relation: $e_1 < e_3 < e_4 < e_5 < e_2 < e_6$

Learn from the sub-properties with smaller bound.
Event Relation Analysis

!F(e1) → !F(e3) → !F(e7) → !F(e9)
A MIPS Processor Example

Checked Property
P: The units MUL5 and FADD3 can be activated together at 8th clock cycle.

LTL: ! F(MUL5=active & FADD3=active & clk=8)

A sub-property example
LTL: ! F(MUL4=active & FADD2=active & clk=7)
Event Relation Construction

Original Property

\[ P_{e7} : \neg F(\text{MUL5}=\text{active} \land \text{FADD3}=\text{active} \land \text{clk}=8) \]

Temporally Decomposed Properties

\[ P_{e1} : \neg F(\text{MUL1}=\text{active} \land \text{Fetch}=\text{active} \land \text{clk}=4) \]

\[ P_{e4} : \neg F(\text{MUL3}=\text{active} \land \text{FADD1}=\text{active} \land \text{clk}=6) \]
• Let \( vstat[sz][2] \) be a 2-dimension array to record the statistics of sub-property results. It is used to indicate the decision ordering of unchecked properties.

• The term \( \text{bias}(vi) \) is used to indicate the variable assignment variance of \( vi \).

\[
\text{bias} (vi) = \frac{\text{Max}( vstat[i][0], vstat[i][1]) + 1}{\text{Min}( vstat[i][0], vstat[i][1]) + 1}
\]
Our decision ordering is based on VSIDS but our method considers decision ordering learned from sub-properites.

Initialization

\[ \text{score}(li) = \text{literal count of } li \text{ in CNF clauses} \]

Periodical update (include initialization)

\[
\text{score}(li) = \begin{cases} 
\text{max}(vi) \times \text{bias}(vi) & (\text{varStat}[i][1] > \text{varStat}[i][0] \& li = vi) \\
\text{score}(li) & \text{otherwise} \\
\text{max}(vi) & (\text{varStat}[i][0] > \text{varStat}[i][1] \& li = vi') 
\end{cases}
\]

where \( \text{max}(vi) = \text{MAX}(\text{score}(vi), \text{score}(vi')) + 1. \)
An Example of Learning

Initialization

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

P2: a=0, b=1, c=1
Test Generation Using Our Method

Inputs: a) Formal model of the Design, $D$
     b) Property $P$ and satisfiable bound $\text{bound}_P$
     c) Decomposed properties prop and satisfiable bounds

Output: A test $\text{test}_p$ for $P$

1. $\text{CNFs} = \text{BMC}(D, \text{props}, \text{bounds})$;
2. $(\text{CNF1, CNF2, ..., CNFn}) = \text{Sort CNF using increasing file size}$
3. Initialize $\text{vstat}$;
4. for $i$ is from 1 to $n$ do
   a) $\text{test}_i = \text{SAT}(\text{CNF}_i, \text{vsat})$;
   b) $\text{Update(vstat, test}_i, \text{bounds[i])}$;
endfor
5. Generate $\text{CNF} = \text{BMC}(D, P, \text{bound}_P)$;
6. return $\text{test}_p = \text{SAT}(\text{CNF, vstat})$;
Case Study 1: MIPS Processor

- We generated 20 properties based on interaction faults on various function unit (ALU, DIV, FADD and MUL). 6 of them cannot be handled by temporal decomposition.

<table>
<thead>
<tr>
<th>Property (test)</th>
<th>zChaff (sec)</th>
<th>Num. of Clusters</th>
<th>Num. of Sub-props</th>
<th>Spatial (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>127.52</td>
<td>3</td>
<td>2</td>
<td>49.41</td>
<td>2.58</td>
</tr>
<tr>
<td>P2</td>
<td>49.24</td>
<td>3</td>
<td>2</td>
<td>15.73</td>
<td>3.13</td>
</tr>
<tr>
<td>P3</td>
<td>9.18</td>
<td>2</td>
<td>1</td>
<td>4.99</td>
<td>1.84</td>
</tr>
<tr>
<td>P4</td>
<td>13.78</td>
<td>2</td>
<td>1</td>
<td>7.28</td>
<td>1.89</td>
</tr>
<tr>
<td>P5</td>
<td>31.63</td>
<td>3</td>
<td>2</td>
<td>12.74</td>
<td>2.48</td>
</tr>
<tr>
<td>P6</td>
<td>120.72</td>
<td>3</td>
<td>2</td>
<td>54.21</td>
<td>2.23</td>
</tr>
</tbody>
</table>

Speedup: 1.84-3.13 times
Case Study 1: MIPS Processor

- For the remaining **14** properties, we adopts both spatial and temporal decompositions.

**Indications:** Test generation complexity is significantly improved
- Spatial decomposition is better in this example
- Temporal decomposition can still get 2.5X speedup
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>Property</th>
<th>zChaff (sec)</th>
<th>Bound</th>
<th>Num. of Sub-properties</th>
<th>Temporal (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>25.99</td>
<td>8</td>
<td>3</td>
<td>0.78</td>
<td>33.32</td>
</tr>
<tr>
<td>P2</td>
<td>48.99</td>
<td>10</td>
<td>4</td>
<td>2.69</td>
<td>18.21</td>
</tr>
<tr>
<td>P3</td>
<td>39.67</td>
<td>11</td>
<td>5</td>
<td>3.45</td>
<td>11.50</td>
</tr>
<tr>
<td>P4</td>
<td>247.26</td>
<td>11</td>
<td>5</td>
<td>22.46</td>
<td>11.01</td>
</tr>
<tr>
<td>P5</td>
<td>160.73</td>
<td>11</td>
<td>5</td>
<td>15.68</td>
<td>10.25</td>
</tr>
<tr>
<td>P6</td>
<td>97.54</td>
<td>11</td>
<td>4</td>
<td>1.56</td>
<td>62.53</td>
</tr>
<tr>
<td>P7</td>
<td>31.39</td>
<td>10</td>
<td>4</td>
<td>12.31</td>
<td>2.55</td>
</tr>
<tr>
<td>P8</td>
<td>161.74</td>
<td>11</td>
<td>4</td>
<td>12.62</td>
<td>12.82</td>
</tr>
<tr>
<td>P9</td>
<td>142.91</td>
<td>10</td>
<td>4</td>
<td>17.57</td>
<td>8.13</td>
</tr>
<tr>
<td>P10</td>
<td>33.77</td>
<td>10</td>
<td>4</td>
<td>1.76</td>
<td>19.19</td>
</tr>
</tbody>
</table>

**Speedup: 3-63 times**
Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion
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 two novel property decomposition techniques based on decision ordering learning.

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