SAT Based Efficient Directed Test Generation Techniques

Presented by
Mingsong Chen

Software Engineering Institute
East China Normal University
May 5, 2011
Outline

- Introduction
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause
- Efficient Test Generation Approaches
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- Conclusion
Outline

- Introduction
  - Model Checking Based Test Generation
    - SAT-based Bounded Model Checking
      - DPLL algorithm
      - Conflict clause
  - Efficient Test Generation Approaches
    - Conflict clause forwarding based approaches
    - Decision ordering based techniques
    - Property decomposition based methods
- Conclusion
Functional validation is a major bottleneck during SoC development! (up to 70% of time and resources are used)

Source: G. Spirakis, keynote address at DATE 2004
Functional Validation Methods

- **Simulation (Validation)**
  - The process of gaining confidence by examining the behavior of the implementation using input/output test vectors
  - Incompleteness verification: not possible for all input vectors
  - Applicable to large designs

- **Formal (Verification)**
  - **Mathematical proof** that a system (implementation) behaves according to a given set of requirements (specification)
  - Complete verification
  - Applied to small and critical components due to the state space explosion problem
Validation using a combination of simulation based techniques and formal methods.
Test Generation using Model Checking

- **Model Checking (MC)**
  - Specification is translated to formal models, e.g., SMV
  - Desired behaviors in temporal logic properties, e.g., LTL
  - Property falsification leads to counterexamples (tests)

- **Test Generation**
  - Generate a counterexample: sequence of variable assignments

**Problem:** Test generation is very costly or not applicable in many complex scenarios.

**Approach:** Exploit learning to reduce validation complexity
- Reduction of test generation time
- Enables test generation in complex scenarios
Outline

Introduction

- Model Checking Based Test Generation
- SAT-based Bounded Model Checking
  - DPLL algorithm
  - Conflict clause

Efficient Test Generation Approaches

- Conflict clause forwarding based approaches
- Decision ordering based techniques
- Property decomposition based methods

Conclusion
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.
SAT Decision Procedure

Given a $\varphi$ in CNF: $(x+y+z)(\neg x+y)(\neg y+z)(\neg x+\neg y+\neg z)$
DPLL Algorithm

while (1) {
    run_periodic_function();
    if ( decide_next_branch() ){
        while ( deduce() == CONFLICT ) {
            blevel = analyze_conflicts();
            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
Conflict clause can be treated as the knowledge learned during the SAT solving. It is a restriction of the variable assignment.
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
Outline

- Introduction
  - Model Checking Based Test Generation
  - SAT-based Bound Model Checking
    - DPLL algorithm
    - Conflict clause
- Efficient Test Generation Approaches
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- Conclusion
Same Design, Different Properties

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

Forward
Property Clustering

- Clustering properties is to exploit the structural and behavior similarity and maximize the validation reuse

- Property clustering methods:
  - Based on structural similarity
  - Based on textual similarity
  - Based on Influence (Cone of Influence)
  - Based on CNF intersections
Identification of Common Conflict Clauses

Let $\wedge$ be the bit “AND” operation. $(0111 \wedge 1010 \wedge 1111 \wedge 1010 \wedge 1110) = 0010$. So the conflict clause $(\neg X_1 \lor X_5 \lor X_6 \lor \neg X_7)$ can be reused for property 2.
Test Generation For A Property Cluster

1. **Cluster** the properties based on similarity

2. **for** each cluster $i$, of properties
   
   ① **Select** base property $p_{i1}$, and generate $\text{CNF}_{i1}$
   
   ② **for** each $\text{CNF}_{ij}$ of $p_j$ (j≠1) in cluster $i$
      
      a) Perform **name substitution** on $\text{CNF}_{ij}$
      
      b) Compute **intersection** $\text{INT}_{ij}$ between $\text{CNF}_{i1}$ and $\text{CNF}_{ij}$
      
      c) Mark the clauses of $\text{CNF}_{i1}$ using $\text{INT}_{ij}$
   
   endfor
   
   ③ **Solve** $\text{CNF}_{i1}$ to get the conflict clauses $\text{CC}_{i1}$ and $\text{test}_{i1}$
   
   ④ **for** each $\text{CNF}_{ij}$ (j≠1)
      
      a) $\text{CNF}_{ij} = \text{CNF}_{ij} + \text{Filter} (\text{CC}_{ij}, j)$
      
      b) **Solve** $\text{CNF}_{ij}$ to get $\text{test}_{ij}$
   
   endfor

endfor
Case Study 1: MIPS Processor

The Architecture

MIPS Processor
- 20 nodes
- 24 edges
- 91 instructions
MIPS Processor Results

- The processor has five pipeline stages: fetch, decode, execute, memory and writeback.
- There are totally 171 properties generated.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Structure</th>
<th>Textual</th>
<th>Influence</th>
<th>Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of Clusters</td>
<td>16</td>
<td>32</td>
<td>27</td>
<td>17</td>
</tr>
<tr>
<td>zChaff (sec.)</td>
<td>3275.07</td>
<td>3266.73</td>
<td>3241.00</td>
<td>3323.34</td>
</tr>
<tr>
<td>(Existing Approach)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our Method (sec.)</td>
<td>957.42</td>
<td>879.19</td>
<td>754.58</td>
<td>751.36</td>
</tr>
<tr>
<td>Speedup</td>
<td>3.42</td>
<td>3.72</td>
<td>4.33</td>
<td>4.42</td>
</tr>
</tbody>
</table>

*zChaff* is a state-of-the-art SAT Solver.
Case Study 2: OSES
This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths. There are totally 51 properties.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Structure</th>
<th>Textual</th>
<th>Influence</th>
<th>Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of Clusters</td>
<td>18</td>
<td>9</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>zChaff (sec.) (Existing Approach)</td>
<td>2119.16</td>
<td>2159.92</td>
<td>2311.47</td>
<td>2134.26</td>
</tr>
<tr>
<td>Our Method (sec.)</td>
<td>939.25</td>
<td>926.98</td>
<td>966.19</td>
<td>794.48</td>
</tr>
<tr>
<td>Speedup</td>
<td>2.26</td>
<td>2.33</td>
<td>2.44</td>
<td>2.69</td>
</tr>
</tbody>
</table>
Outline

- Introduction
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause

- Efficient Test Generation Approaches
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods

- Conclusion
Decision Ordering Problem

Given a $\varphi$ in CNF: $(x+y+z)(\neg x+y)(\neg y+z)(\neg x+\neg y+\neg z)$

- A wise decision ordering can quickly locate the true assignment.
  - Bit value ordering
  - Variable Ordering

Best decision: $\neg x$, $z$
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

With bit value learning, 4 conflicts in SAT2.
With bit value + variable order learning, 1 conflict in SAT2.
Our method — An Example with 3 properties

**Approach:** Using the statistics of the counterexamples when checking the properties in a cluster
- Count 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=?

Predict ordering for P3

score(a) ↑, score(a’)↑

score(b) ↑, score(b’)↑

score(c) ↑, score(c’)↑
Case Study 1: MIPS Processor

- For each function unit (ALU, DIV, FADD, and MUL) in the pipelined processor. We generate 4 properties.

<table>
<thead>
<tr>
<th>Property (test)</th>
<th>zChaff (sec)</th>
<th>Clustering</th>
<th>Speedup (over zChaff)</th>
<th>Decision Ordering</th>
<th>Speedup (over Clustering)</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 1: MIPS Processor

Test generation time is significantly improved
- Drastic reduction of conflict clauses
- Drastic reduction in number of implications
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>Clustering</th>
<th>Speedup (over zChaff)</th>
<th>Decision Ordering</th>
<th>Speedup (over Clustering)</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>
<tr>
<td>Average</td>
<td>-</td>
<td>227.53</td>
<td>123.21</td>
<td>1.85</td>
<td>20.67</td>
<td>5.97</td>
</tr>
</tbody>
</table>
Outline

Introduction

- Model Checking Based Test Generation
- SAT-based Bounded Model Checking
  - Implication graph
  - SAT decision procedure – DPLL algorithm

Efficient Test Generation Approaches

- Conflict clause forwarding based approaches
- Decision ordering based techniques
- Property decomposition based methods

Conclusion
Property Decomposition Techniques


Drawback: Hard to automate

Learnings

BMC

Test
Spatial Decomposition

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

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

Learning from P1 can reduce the Time(P)?
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: $e_1 < e_3 < e_4 < e_5 < e_2 < e_6$
Temporal Decomposition

- Temporal Decomposition
- Cause-effect: Happen-before
- Diagram:
  - Relations: $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_6 \rightarrow e_7 \rightarrow e_8 \rightarrow e_9$

- Formula:
  - $!F(e_1) \rightarrow !F(e_3) \rightarrow !F(e_7) \rightarrow !F(e_9)$
Case Study 1: MIPS Processor

- We generated 6 complex properties based on interaction faults on various function unit (ALU, DIV, FADD and MUL), which cannot 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
This case study is an 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: 3-62 times
Outline

- Introduction
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause

- Efficient Test Generation Approaches
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods

- Conclusion
Conclusion

- Validation is a major bottleneck in HW/SW designs.
- This presentation discusses how to reduce the overall validation effort for directed test generation from models.

  1. Conflict clause forwarding and property clustering methods
  2. Efficient decision ordering approaches
  3. Property decomposition techniques

- Successfully applied on both HW/SW designs
  - Several orders of magnitude reduction in overall validation effort
Thank you!