Automated Debugging of Missing Assumptions

BRIAN KENG¹, EVEAN QIN², ANDREAS VENERIS¹, BAO LE¹

¹UNIVERSITY OF TORONTO
²VENNSA TECHNOLOGIES INC.
Outline

- Motivation
- Background
- Debugging Missing Assumptions
  - Using a Single Counter-example
  - Using Multiple Counter-examples
- Experimental Results
- Conclusion
Outline

- Motivation
- Background
- Debugging Missing Assumptions
  - Using a Single Counter-example
  - Using Multiple Counter-examples
- Experimental Results
- Conclusion
Motivation

- **Formal Property Checkers**
  - Exhaustively verify an assertion which encodes the design intent
    - Returns counter-example that excites failure in the design
    - Can locate hard-to-find corner case failures

- **Debugging formal counter-examples can be challenging, as observed failures can be due to:**
  - A design bug
  - An incorrectly written assertion
  - Or a missing assumption
Motivation

Formal Verification
State Space

Simulation Coverage

False Bug
Invalid States

Real Bug
Motivation

• Causes of Missing Assumptions
  ○ The design specification
  ○ Undocumented assertions
  ○ Functionality of adjacent design blocks

• The engineer needs to find the missing assumptions in order to prune the returned counter-example list

• This will expose counter-examples encoding “real” design bugs
Outline

• Motivation
• Background
• Debugging Missing Assumptions
  o Using a Single Counter-example
  o Using Multiple Counter-examples
• Experimental Results
• Conclusion
MUS and MCS

- Given a UNSAT Boolean formula $\Phi$ in CNF:
  - UNSAT Cores:
    - Subset of clauses in $\Phi$ that are UNSAT
  - Minimal Unsatisfiable Subset (MUS)
    - UNSAT core where every proper subset is SAT
  - Minimal Correct Set (MCS)
    - Minimal subset of clauses in $\Phi$ such that removing these clauses will make $\Phi$ SAT
UNSAT Core Example

\[(a) \land (b) \land (a \lor b)\]

UNSAT Core and MUS

MCSs
MUIS and MCIS

- **Minimal Unsatisfiable Input Subset (MUIS)**
  - A minimal unsatisfiable set of input unit clauses that result in $\Phi$ being UNSAT

- **Minimal Correction Input Set (MCIS)**
  - A minimal set of input unit clauses that when removed, will result in $\Phi$ being SAT

- **MUIS (MCIS) are analogous to MUS (MCS)**
MUIS and MCIS Example

Input Clauses

\((a) \land (b) \land (c)\)

Transition Clauses

\(\neg (a \lor d) \land (b \lor \neg d) \land (\neg a \lor \neg b \lor d)\)

\(\neg (c \lor f) \land (\neg d \lor f) \land (c \lor d \lor \neg f) \land (\neg f)\)

MCIS
Outline

- Motivation
- Background
- Debugging Missing Assumptions
  - Using a Single Counter-example
  - Using Multiple Counter-examples
- Experimental Results
- Conclusion
Debugging Missing Assumptions

- Idea:
  - Give the engineer suggestions for the missing assumptions
- Extract all MUIS, $U^i$, from the design CNF to build a filtering function $F = U^0 \ldots U^k$
- Given an input constraint $A$:
  - If $F \land A$ is SAT, the failure seen in the counter-example is not prevented
  - If $F \land A$ is UNSAT, then $A$ will ensure that future failures will not occur in the same way as the given counter-example.
- MUISs can be computed in terms of MCISs
Debugging Missing Assumptions

Outline

- Motivation
- Background
- **Debugging Missing Assumptions**
  - Using a Single Counter-example
  - Using Multiple Counter-examples
- Experimental Results
- Conclusion
Using Multiple Counterexamples: Overall Flow

- Generate multiple distinct counterexamples using formal tool
- Generate input assumptions that can prevent failures seen in the counter-examples
- More counter examples can aid general debugging
Generating Multiple Counter-examples

- It is difficult to generate a ‘useful’ second counter-example
  - The assertion should fail in a different manner
  - Therefore, distinct counter-examples must be found
- Two counter-examples, R and S, are distinct given their set of MUSs, $M_R$ and $M_S$, such that:

$$M_R \cap M_S = \emptyset$$
Generating Multiple Counter-examples

- To generate distinct counter-examples, we must prevent previously seen MUSs from occurring again
  - The MUS can be prevented if at least one of its clauses is not present
  - Functionality of the design must not be changed
  - Only input clauses can be modified to retain functionality
Generating Multiple Counter-examples

- As a result, previously found MUISs can be blocked.
- Using the duality between MUISs and MCISs, the blocking constraint can be computed from a single MCIS.

\[
B^k = c_0^k \land c_1^k \land \ldots \land c_{C^k}^k
\]

\[
B = B^1 \land \cdots \land B^k
\]
A Practical Algorithm

- Generate Multiple Counter-examples: Generate Assumptions, Extract MCISs, Property Checker, Counter-example
- Generate Assumptions: Filter Assumptions, Pruned Assumptions, Candidates, Assumption Model, MCIS
- Extract MCISs
- Run Formal Tool
- Extract All MCIS
- MCIS set != 0 && max Cex not reached
- More Cex Returned?
  - No
    - Add blocking constraint. Keep Cex
  - Yes
    - Extract Blocking
- Run Formal Tool
- Extract all MCIS
- No
  - Yes
Outline

- Motivation
- Background
- Debugging Missing Assumptions
  - Using a Single Counter-example
  - Using Multiple Counter-examples
- Experimental Results
- Conclusion
**Experimental Results**

<table>
<thead>
<tr>
<th>Crt Name</th>
<th># CE</th>
<th>MCIS Time (s)</th>
<th>Formal Time (s)</th>
<th>Gen Time (s)</th>
<th>Total Candidate Assumpt</th>
<th>Filter Cand. Assumpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu</td>
<td>10</td>
<td>255</td>
<td>100</td>
<td>5</td>
<td>31</td>
<td>3</td>
</tr>
<tr>
<td>ddr2</td>
<td>9</td>
<td>383</td>
<td>1395</td>
<td>1504</td>
<td>4094</td>
<td>333</td>
</tr>
<tr>
<td>hpdmc</td>
<td>10</td>
<td>70</td>
<td>60</td>
<td>4</td>
<td>90</td>
<td>33</td>
</tr>
<tr>
<td>mips</td>
<td>4</td>
<td>278</td>
<td>93</td>
<td>9</td>
<td>59</td>
<td>22</td>
</tr>
<tr>
<td>mrisc</td>
<td>8</td>
<td>88</td>
<td>1126</td>
<td>5</td>
<td>39</td>
<td>10</td>
</tr>
<tr>
<td>pci</td>
<td>8</td>
<td>611</td>
<td>761</td>
<td>7</td>
<td>25</td>
<td>10</td>
</tr>
</tbody>
</table>

- The design of ddr2 leads to the excessive number of filtered candidate assumptions. This is likely due to the large number of inputs with assertions written on them. The user is presented with a relatively large number of filtered candidate assumptions, which makes it challenging for them to select the most appropriate one.
## Experimental Results

<table>
<thead>
<tr>
<th>Crt Name</th>
<th># CE</th>
<th>MCIS Time (s)</th>
<th>Form Time (s)</th>
<th>Total Cand. Assumptions</th>
<th>Filt Using n CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu</td>
<td>15</td>
<td>653</td>
<td>356</td>
<td>154</td>
<td>2 2 2 2</td>
</tr>
<tr>
<td>ddr2</td>
<td>3</td>
<td>625</td>
<td>86</td>
<td>226</td>
<td>68 - - -</td>
</tr>
<tr>
<td>hpdmc</td>
<td>15</td>
<td>112</td>
<td>148</td>
<td>97</td>
<td>17 16 11 11</td>
</tr>
<tr>
<td>mips</td>
<td>4</td>
<td>278</td>
<td>93</td>
<td>163</td>
<td>36 - - -</td>
</tr>
<tr>
<td>mrisc</td>
<td>8</td>
<td>88</td>
<td>1126</td>
<td>92</td>
<td>11 5 - -</td>
</tr>
<tr>
<td>pci</td>
<td>8</td>
<td>611</td>
<td>761</td>
<td>267</td>
<td>9 9 - -</td>
</tr>
</tbody>
</table>

- The ideal behavior is when a counter example is found that excites the design differently than the original one but shifted in time. The percentage of filters used for each counter example is: 2%, 5%, 10%, and 15%.
Outline

- Motivation
- Background
- Debugging Missing Assumptions
  - Using a Single Counter-example
  - Using Multiple Counter-examples
- Experimental Results
- Conclusion
Conclusion

- **Debugging missing assumptions**
  - Generate multiple formal counter-examples for the failure
  - Generate a function that encodes the input combinations that caused the assertion to fail
  - Use the function to generate a list of fixed cycle assumptions that prevent the failures
- **These can be used as hints for the actual missing assumption**