Lazy Abstraction and SAT-based Reachability in Hardware Model Checking

Yakir Vizel
Orna Grumberg
Sharon Shoham

FMCAD 2012

Computer Science Department, Technion, Israel
Outline

• Background
  – Reachability Analysis
  – Abstraction
  – Lazy Abstraction
  – IC3

• Lazy Abstraction with IC3
Model Checking

- Given a system and a specification, does the system satisfy the specification.
Reachability Analysis

Does AGp hold?

INIT $S_1$ $S_2$ ..... $S_n$ Bad=$\neg P$
Abstraction

• Fights the state explosion problem
• Removes or simplifies details that are irrelevant
• Abstract model contains less states
Visible Variables Abstraction
Abstraction-Refinement

• Abstract model may contain spurious behaviors
  – Spurious counterexample may exist
• Refinement is applied to remove the spurious behavior
Lazy Abstraction

- Different abstractions at different steps of verification

- Refinement applied *locally*, where needed
SAT-based Reachability with IC3
IC3 Basics

- Iteratively compute Over-approximated Reachability Sequence (OARS) \( <F_0, F_1, \ldots, F_k> \) s.t.
  - \( F_0 = \text{INIT} \)
  - \( F_i \Rightarrow P \)
  - \( F_i \Rightarrow F_{i+1} \)
  - \( F_i \land \text{TR} \Rightarrow F'_{i+1} \)

- \( F_i \) - CNF formula represented by a set of clauses
- \( \text{TR} \) - the \textit{concrete} transition relation
- \( F'_{i+1} \) is over the next state variables
Iteration of IC3

\[ F_k \land TR \not\models F'_{k+1} \]
Locality in IC3

• IC3 applies checks of the form
  – $F_k \land TR \land \neg P'$
    • Finds a state in $F_k$ that can reach $\neg P$
  – $F_i \land TR \land s'$
    • Finds a predecessor in $F_i$ to the state $s$

• Using only one TR
  – No unrolling
Our Approach - L-IC3

• Use IC3’s local checks for Lazy Abstraction
  – Different abstraction at different time frames
  – Use visible variables abstraction
  – Different variables are visible at different time frames
Concrete Model

INIT $F_1$ $\cdots$ $F_{k-1}$ $F_k$
Using Abstraction
Using Lazy Abstraction
Lazy Abstraction + IC3 = L-IC3

- \(<F_0,F_1,\ldots,F_{k+1}> - \text{Reachable states}\)

- \(<U_1,U_2,\ldots,U_{k+1}> - \text{Abstractions}\)
  - \(U_i\) - set of visible variables
    - \(U_i\) variables have a \textit{next state function}\n    - The rest, \textit{inputs}\n  - \(U_i \subseteq U_{i+1}\)
    - \(U_{i+1}\) is a \textit{refinement} of \(U_i\)
L-IC3 Iteration

- Initialize $F_{k+1}$ to $P$
- Initialize $U_{k+1}$ to $U_k$
- Same problem, the sequence may not be an OARS
Abstract Counterexample

\[ F_i \land TR_{i+1} \land s' \]

\[ F_k \land TR_{k+1} \land \neg P' \]

\[ \neg P \]
Check Spuriousness

- An abstract CEX of length $k+1$ exists
- Use an IC3 iteration with the concrete TR
- If a real CEX exists, it will be found
Check Spuriousness (2)

• If no real CEX exists:
  – Compute a *strengthened* sequence
    \(<F_{r_0}, F_{r_1}, \ldots, F_{r_{k+1}}\>
  – The strengthened sequence is an OARS
  – Strengthening eliminates all CEXs of length \(k+1\)
Lazy Abstraction Refinement

• If no real CEX is found by (concrete) IC3 even though (abstract) L-IC3 strengthening failed
  – Abstraction is too coarse

• Refine the sequence \(<U_1, U_2, \ldots, U_{k+1}>\) as follows:

• Since \( Fr_i \land TR \Rightarrow Fr'_{i+1} \)
  – \( Fr_i \land TR \land \neg Fr'_{i+1} \) is unsatisfiable
  – Use the UnSAT Core to add visible variables
    • \( U_{r,i+1} = U_{i+1} \cup U\text{Core}_i \)
Incrementality

• The concrete IC3 iteration works on the already computed sequence $<F_0, F_1, ..., F_{k+1}>$

• At the end of refinement, L-IC3 continues from iteration $k+2$
## Experiments

<table>
<thead>
<tr>
<th>Test</th>
<th>#Vars</th>
<th>T/F</th>
<th>#V[Ω]</th>
<th>#V[Ω]</th>
<th>#C[Ω]</th>
<th>#C[Ω]</th>
<th>k</th>
<th>k_L</th>
<th>T</th>
<th>T_L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ind_1</td>
<td>5693</td>
<td>T</td>
<td>236</td>
<td>11</td>
<td>617</td>
<td>62</td>
<td>14</td>
<td>8</td>
<td>133</td>
<td>9.2</td>
</tr>
<tr>
<td>Ind_2</td>
<td>5693</td>
<td>T</td>
<td>104</td>
<td>24</td>
<td>2101</td>
<td>32</td>
<td>32</td>
<td>14</td>
<td>513</td>
<td>13.6</td>
</tr>
<tr>
<td>Ind_3</td>
<td>11866</td>
<td>F</td>
<td>1001</td>
<td>816</td>
<td>8457</td>
<td>3939</td>
<td>15</td>
<td>18</td>
<td>1646</td>
<td>599</td>
</tr>
<tr>
<td>Ind_4</td>
<td>1204</td>
<td>T</td>
<td>114</td>
<td>105</td>
<td>18698</td>
<td>229</td>
<td>8</td>
<td>8</td>
<td>818</td>
<td>3</td>
</tr>
<tr>
<td>Ind_5</td>
<td>3854</td>
<td>T</td>
<td>&gt;470</td>
<td>666</td>
<td>&gt;8320</td>
<td>5363</td>
<td>&gt;6</td>
<td>11</td>
<td>TO</td>
<td>730</td>
</tr>
<tr>
<td>Ind_6</td>
<td>1389</td>
<td>F</td>
<td>397</td>
<td>417</td>
<td>12455</td>
<td>19742</td>
<td>13</td>
<td>19</td>
<td>262</td>
<td>1268</td>
</tr>
</tbody>
</table>
# Experiments - Laziness

<table>
<thead>
<tr>
<th>Test</th>
<th>#Vars</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
<th>#TF</th>
<th>#AV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ind₂</td>
<td>5693</td>
<td>1-7</td>
<td>31</td>
<td>8</td>
<td>42</td>
<td>9</td>
<td>51</td>
<td>10-14</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ind₃</td>
<td>11866</td>
<td>1</td>
<td>323</td>
<td>2</td>
<td>647</td>
<td>3</td>
<td>686</td>
<td>4</td>
<td>699</td>
<td>5</td>
<td>705</td>
</tr>
<tr>
<td>Ind₅</td>
<td>3854</td>
<td>1</td>
<td>428</td>
<td>2</td>
<td>453</td>
<td>3</td>
<td>495</td>
<td>4</td>
<td>499</td>
<td>5</td>
<td>503</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>560</td>
<td>7</td>
<td>574</td>
<td>8</td>
<td>576</td>
<td>9-11</td>
<td>577</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

• Novel lazy abstraction algorithm for hardware model checking
• Abstraction-Refinement is done incrementally
• More efficient generalization
  
  Up to two orders of magnitude runtime improvement

• Also in the paper: may vs. must proof obligations
Thank You