Automatic Abstraction Refinement of Transition Relation for PDR

Kuan Fan, Ming-Jen Yang, Chung-Yang (Ric) Huang

2016.1.26
Outline

• Introduction
  – Property Directed Reachability
  – Abstraction
• The Proposed Method
• Experimental Results
• Conclusion
Introduction
Property Directed Reachability

- PDR\(^1\) aka IC3\(^2\), is a SAT-based model checking algorithm developed by Aaron Bradley in 2011.
- IC3 won the 3rd place in HWMCC’10 and only lost, by a narrow margin, to two mature engines (ABC and PdTRAV)
- Best single engine algorithm

\(^1\)N. Ee´n, A. Mishchenko, R. Brayton: Efficient Implementation of Property Directed Reachability (FMCAD’11)
\(^2\)A. R. Bradely, SAT-based model checking without unrolling (VMCAI’11)
PDR: The Big Picture

- Transition system: $M = (V, S, \text{Init}(S), Tr(V, S, S'))$
- Invariant property: $P$
- $i$-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
- Five invariants:
  1. $F_0 = \text{Init}$
  2. $F_i \Rightarrow F_{i+1}$ for $0 \leq i \leq k-1$
  3. $F_i \land Tr \Rightarrow F_{i+1}$ for $0 \leq i \leq k-1$
  4. $F_i \supseteq F_{i+1}$, as sets of clauses. for $0 \leq i \leq k-1$
  5. $F_i \Rightarrow P$ for $0 \leq i \leq k$

---

$^1$N. Eé'n, A. Mishchenko, R. Brayton: Efficient Implementation of Property Directed Reachability (FMCAD'11)
PDR: The Big Picture

- Transition system: $M = (V, S, \text{Init}(S), Tr(V, S, S'))$
- Invariant property: $P$
- i-step over-approximation sets of clauses: $F_0, F_1, ..., F_k$
- Termination criteria:
  - A counterexample is found.
  - When $\exists i \leq k. F_i = F_{i+1}$. Then:
    1. $\text{Init} \Rightarrow F_i$
    2. $F_i \land Tr \Rightarrow F_i$
    3. $F_i \Rightarrow P$
PDR in The State Space

• i-step over-approximation sets of clauses: $F_0, F_1, ..., F_k$
PDR in The State Space

- i-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
PDR in The State Space

• i-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
PDR in The State Space

- i-step over-approximation sets of clauses: $F_0$, $F_1$, ..., $F_k$

$F_0 \neq F_1 \neq F_2 \neq F_3 \neq F_4$
Abstraction of Latch Variable

• i-step over-over-approximation sets of clauses:
  \[ F_0', F_1', ..., F_k' \]
Abstraction of Latch Variable

• i-step over-over-approximation sets of clauses:
  \[F_0', F_1', ..., F_k'\]
Abstraction of Transition Relation

- $i$-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
Abstraction of Transition Relation

- i-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
Abstraction of Transition Relation

- i-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
Abstraction of Transition Relation

• i-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
Abstraction of Transition Relation

• i-step over-approximation sets of clauses: $F_0, F_1, \ldots, F_k$
Abstraction of Transition Relation

- i-step over-over-approximation sets of clauses:
  \[ F_0'', F_1'', \ldots, F_k'' \]
Previous Works

- IC3-Guided Abstraction\(^1\)
- Lazy Abstraction\(^2\)

- Major difference:
  - flop-level abstraction & gate-level abstraction
  - Heuristics to handle counterexamples

---

\(^1\)Jason Baumgartner, Alexander Ivrii, Arie Matsliah, and Hari Mony. Ic3-guided abstraction. (FMCAD’12)
\(^2\)Yakir Vizel, Orna Grumberg, and Sharon Shoham. Lazy abstraction and sat-based reachability in hardware model checking. (FMCAD’12)
Granularity of Abstraction

- Flop-level abstraction
Granularity of Abstraction

- Gate-level abstraction
Priority-based Abstraction Refinement

Goal: find a minimal subset of PPIs s.t. restricting them to values in the given Cex implies the property fails.

Rules of priority propagation:

Priority: smaller number represents higher priority

---

1Alan Mishchenko et al.: Variable time-frame abstraction (IWLS’12)
PDR: Program Flow

\[ P_0 = \text{Init}, \, t = 1, \, P_1 = \text{newFramo0} \]
GLA: a Gate-level, Hybrid Approach\(^1\)

1Alan Mishchenko et al. Gate-level abstraction revisited(DATE’13)

Initial abstraction \(A'\)

Refine

Shrink

Cex

\(k = k+1\)

Model check \(A'\)

Proved

Safe
The Proposed Method
The Proposed Method: An Overview

- Embed GLA-like abstraction refinement in PDR
The Proposed Method: An Overview

- Embed GLA-like abstraction refinement in PDR
The Proposed Method: An Overview

- Embed GLA-like abstraction refinement in PDR
The Proposed Method

- **Varying** Abstract Model $M_a$ of $M$:
  \[ M_a = (V_a, S_a, Init, Tr_a(V_a, S_a, S_a')) \]
  w.r.t. abstraction $A'$

Invariant properties $P$
Blocking and Refining Phase

• Using abstract transition relation $Tr_\alpha$ when doing local reachability checks.
• Any Blocked cube is valid w.r.t. the concrete model.
• May refine abstract counterexamples longer than current depth $k$.
• Gates added now are remembered for later incremental UNSAT cores extraction.
Shrinking Phase

- Remove superfluous logic added during **Blocking and Refining Phase**
- Make sure five invariants still hold while changing $M_a$ to $M_a$:
  1. $F_0 = \text{Init}$
  2. $F_i \Rightarrow F_{i+1}$
  3. $F_i \land Tr_{a} \Rightarrow F_{i+1}$
  4. $F_i \supseteq F_{i+1}$
  5. $F_i \Rightarrow P$
Experimental Results
Experiments

• The proposed method, called AbsPDR, was implemented in ABC.
• We compared it with PDR as implemented in ABC.
• Benchmark: HWMCC’13/14 benchmark suits, 392 instances
• Machine: Intel Xeon, 2.5 GHz freq; 32 GB mem.
• Timeout: 900 sec
Results Summary

- Focus: the impact of abstraction refinement to original PDR (run \textit{pdr m} in ABC).
- AbsPDR refines only minimal (shortest) counterexamples.
- AbsPDR-a refines long counterexamples as PDR does.
- All other features used in AbsPDR(-a) are identical to PDR.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>#Solved</th>
<th>( \Delta_{\text{baseline}} )</th>
<th>Gained</th>
<th>Lost</th>
<th>Cumulative time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDR</td>
<td>31</td>
<td>98</td>
<td>129</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PDR-d</td>
<td>36</td>
<td>108</td>
<td>144</td>
<td>+15</td>
<td>19</td>
</tr>
<tr>
<td>AbsPDR</td>
<td>33</td>
<td>115</td>
<td>148</td>
<td>+19</td>
<td>33</td>
</tr>
<tr>
<td>AbsPDR-a</td>
<td>33</td>
<td>114</td>
<td>147</td>
<td>+18</td>
<td>32</td>
</tr>
</tbody>
</table>
Runtime Comparison
### Abstraction Results

Instances unsolved by PDR

| Instance   | Original |                  |                  |                  |                  |                  |                  |
|------------|----------|------------------|------------------|------------------|------------------|------------------|
|            | #FFs     | #Ands            | #LUTs            | #FFs            | #FFs %           | #LUTs            | #LUTs %           |
| 6s350rb35  | 243399   | 1550409          | 840338           | 659             | 0.3              | 1607             | 0.2              |
| 6s350rb46  | 243399   | 1550412          | 840339           | 946             | 0.4              | 2320             | 0.3              |
| 6s353rb036 | 102390   | 622040           | 319623           | 366             | 0.4              | 1014             | 0.3              |
| 6s353rb101 | 102390   | 622040           | 319623           | 836             | 0.8              | 2861             | 0.9              |
| 6s361rb52584 | 186401 | 1773868          | 846836           | 77              | 0.04             | 299              | 0.04             |
| 6s361rb54373 | 186401 | 1773868          | 846836           | 403             | 0.2              | 1716             | 0.2              |
| 6s364rb12666 | 202686 | 922963           | 613587           | 74              | 0.04             | 245              | 0.04             |
| 6s218b2950  | 58676    | 250531           | 192162           | 2806            | 4.8              | 14732            | 7.7              |
| 6s286rb07843 | 101639 | 737673           | 366690           | 778             | 0.8              | 2589             | 0.7              |
| 6s202b00    | 68881    | 473964           | 236741           | 277             | 0.4              | 791              | 0.3              |
| 6s203b19    | 68957    | 474324           | 236993           | 307             | 0.4              | 883              | 0.4              |
| 6s203b41    | 68957    | 474322           | 236994           | 416             | 0.6              | 1335             | 0.6              |
| AbsPDR-a    |          |                  |                  |                  |                  |                  |                  |
|            | #FFs     | #FFs %           | #LUTs            | #LUTs %          |                  |                  |                  |
| 6s350rb35  | 624      | 0.3              | 1644             | 0.2              |
| 6s350rb46  | 851      | 0.4              | 2247             | 0.3              |
| 6s353rb036 | 513      | 0.5              | 1441             | 0.5              |
| 6s353rb101 | 776      | 0.4              | 3793             | 0.4              |
| 6s361rb52584 | 77       | 0.04             | 299              | 0.04             |
| 6s361rb54373 | 777      | 0.4              | 3793             | 0.4              |
| 6s364rb12666 | 86       | 0.04             | 280              | 0.05             |
| 6s218b2950  | 86       | 0.04             | 280              | 0.05             |
| 6s286rb07843 | 916      | 0.9              | 3102             | 0.8              |
| 6s202b00    | 266      | 0.4              | 761              | 0.3              |
| 6s203b19    | 338      | 0.5              | 951              | 0.4              |
| 6s203b41    | 525      | 0.8              | 1733             | 0.7              |

The sizes of final abstractions are below 1%
Conclusion
Conclusion

• We present an efficient algorithm that embeds GLA-like abstraction refinement in PDR.

• Experimental results show that our approach outperforms original PDR and complements it in a large number of benchmark instances.
Thank You!
Abstraction : How To?

• Counterexample-based abstraction (CBA/CEGAR):
  – Start with one gates of property/state variable
  – See if target hit
  – Otherwise, **Refine** by adding more gates.

• Proof-based abstraction (PBA):
  – Look at the UNSAT-core to further decide which logic(gate) is necessary

• Hybrid method:
  – Interleave CBA and PBA
Priority-based Abstraction Refinement

Goal: find a minimal subset of PPIs s.t. restricting them to values in the given Cex implies the property fails.

lowest priority needed to produce the value

Rules of priority propagation:

Priority: smaller number represents higher priority

1Alan Mishchenko et al.: Variable time-frame abstraction(IWLS’12)
Priority-based Abstraction Refinement

Goal: find a minimal subset of PPIs s.t. restricting them to values in the given Cex implies the property fails.

Initial priority: \{PIs, Constant node, FOs in time-frame 0\} = 0
PPIs = 1 \sim n

Priority: smaller number represents higher priority

Cex is concrete!
Priority-based Abstraction Refinement

Goal: find a minimal subset of PPIs s.t. restricting them to values in the given Cex implies the property fails.

Initial priority: \{PIs, Constant node, FOs in time-frame 0\} = 0
PPIs = \{1 \sim n\}

Priority: smaller number represents higher priority

Added to abstraction

Abstract Cex!
Shrinking Phase

• Remove superfluous logic added during **Blocking and Refining Phase**

• Make sure five invariants still hold while changing $M_a$ to $M_a$:
  1. $F_0 = \text{Init}$
  2. $F_i \Rightarrow F_{i+1}$
  3. $F_i \land Tr_a \Rightarrow F_{i+1}$
  4. $F_i \supseteq F_{i+1}$
  5. $F_i \Rightarrow P$
Shrinking Phase

- Recall that an incremental UNSAT core is recorded only in terms of those gates added in current iteration $k$
- The gates included in previous iterations are NEVER removed
- Extract incremental UNSAT cores:
  - $G_i = \text{gates included in } \text{UNSATCore}(F_i \land Tr_a \land \neg F_{i+1}) \text{ for } 0 \leq i \leq k-1$
  - $G_k = \text{gates included in } \text{UNSATCore}(F_k \land \neg P)$
  - $G_r = G_0 \cup G_1 \cup ... \cup G_k$
  - $A'' = \text{remove gates do not exist in } G_r \text{ from } A'$