Lazy Abstraction

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar & Gregoire Sutre

Presented by Kashyap Thimmaraju
ECS 289C
Winter 2015
UC Davis

Some slides from Rupak Majumdar
Agenda

- Introduction
- Motivation
- Lazy abstraction through an example
- BLAST
- Conclusion
Introduction

- What is model checking?
- Why should we care?
- Who uses it?
- And what is it used for?
Flavors of Model checking

- Symbolic algorithms
- Bounded model checking
- Partial order reduction
- Abstraction
- Counterexample guided abstraction refinement (CEGAR)
Motivation

• The paper states there are 2 trends that have brought model checking to software verification
  • More than just BDD crunching.
  • SAT solvers are getting better.
• The traditional flow of CEGAR can be improved upon by “short-circuiting”.
Lazy Abstraction

- Avoid repetitive work by refining what is already known about the model from a “pivot state”.
- Unite abstraction, verification and counter-example refinement on-the-fly.
- Do as much with as little as possible.
- Semi-algorithm. What?? If yes, finite time verifiable. If no, might not halt.
Lazy Abstraction

- Abstraction through control flow automaton
  - Directed edges labelled with basic block or an assume predicate.
- Forward search (DFS)
  - Till an error state is reached.
- Backwards counterexample analysis
  - Reachable and bad regions
  - Weakest precondition
  - Pick out predicates that appear 1st in the proof of unsatisfiability as refinements.
- Search with new predicates from pivot.
Lazy Abstraction working eg.
BLAST toolkit

- Written in Objective Cml
- Program->CFA using C-Breeze C Compiler Infrastructure
- Data structure for C regions and pre, post and focus functions.
- BDDs for data regions.
- Theorem prover- Simplify
- Proof generating theorem prover- Vampyre.
Driver verification using BLAST

- Check locking disciplines, null pointer dereferences, correct handling of dispatch routines.

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Predicates Total</th>
<th>Predicates Active</th>
<th>Thm Prover Calls Total</th>
<th>Thm Prover Calls Cached</th>
<th>Running Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver.c</td>
<td>95</td>
<td>3</td>
<td>3</td>
<td>260</td>
<td>165</td>
<td>0.08</td>
</tr>
<tr>
<td>funlock.c</td>
<td>40</td>
<td>4</td>
<td>3</td>
<td>340</td>
<td>182</td>
<td>0.14</td>
</tr>
<tr>
<td>read.c</td>
<td>370</td>
<td>28</td>
<td>18</td>
<td>5643</td>
<td>2862</td>
<td>4.42</td>
</tr>
<tr>
<td>floppy.c</td>
<td>6473</td>
<td>5</td>
<td>5</td>
<td>4137</td>
<td>3759</td>
<td>2.05</td>
</tr>
<tr>
<td>qpmouse.c</td>
<td>400</td>
<td>3</td>
<td>3</td>
<td>3117</td>
<td>2925</td>
<td>0.74</td>
</tr>
<tr>
<td>ll_rw_block.c</td>
<td>1281</td>
<td>9</td>
<td>7</td>
<td>10143</td>
<td>9483</td>
<td>5.82</td>
</tr>
</tbody>
</table>
Pros and cons

- Save cost of verifying previously visited states.
- Sound.
- Adds path sensitivity to dataflow analysis.
- Proving correctness needs data-dependent analysis.
- May never terminate (need terminating conditions)
- Subtle savings
Conclusion

**Theorem Proving**
- loop invariants
+ Behaviors encoded in logic
Refine
+ Theorem provers
Computing Successors, Refine

**Program Analysis**
- Imprecise
+ Abstraction
Shrink state space

**Model Checking**
- Finite-state model, state explosion
+ State Space Exploration
Path Sensitive Analysis
+ Counterexamples
Finding Relevant Facts

Lazy Abstraction
Discussion
