Software Verification with Abstraction-Based Methods

Ákos Hajdu

PhD student

Department of Measurement and Information Systems, Budapest University of Technology and Economics
MTA-BME Lendület Cyber-Physical Systems Research Group
Electrical and Computer Engineering Department, McGill University
Background
Background – Formal Verification

- Formal verification
  - Prove or disprove the correctness of a system with respect to a formal property (specification) relying on sound mathematical basis

- Model checking
  - Exhaustively enumerate the possible states and transitions (the state space) of the system and check if it meets the property
Model checking in general

- Automata, formulas, state machines, ...
- An algorithm, a software, a protocol, a circuit, ...
- Assertions, temporal logic, reference automata, ...
- Explicit, symbolic, abstraction, ...

Real-life system

Formal model

Formalized property

Model checking algorithm

Ok

Counterexample
This talk: focus on software and abstraction

- Real-life system
- Formal model
- Formalized property
- Source code
- Assertions

Model checking algorithm

Ok
Violation execution
Counterexample

Control Flow Automata

Assertions + CEGAR

Violating execution
Background – Model and Property

- Control-Flow Automaton
  - Set of control locations (PC)
  - Set of edges with operations over a set of variables
    - E.g., guard, assignment ...

Typical property: “error” location should not be reachable
Background – States and Transitions

- **State**: location + valuation of variables \((L, x_1, x_2, \ldots, x_n)\)
- **Transition**: operations
- **Problem**: state space explosion caused by data variables
  - E.g., 10 locations and 2 integers: \(10 \cdot 2^{32} \cdot 2^{32}\) possible states
- **Goal**: reduce the state space representation by abstraction
Counterexample-Guided Abstraction Refinement (CEGAR)
CEGAR – Introduction

Concrete state space
Abstraction
Abstract state space
Abstract counterexample
Spurious counterexample
Refined state space

Property holds
OK
Counterexample

Model, property
Init
Abstraction
Check
Concretize
Concrete
Refine
Abstract counterexample
**CEGAR – Initial Abstraction**

- **Predicate abstraction**
  - Track predicates instead of concrete values
  - $|P|$ predicates $\rightarrow 2^{|P|}$ possible abstract states
  - Label of a state: predicates, e.g. $\neg(x > y) \land (y = 3)$

<table>
<thead>
<tr>
<th>Variables:</th>
<th>$x, y$; $D_x = D_y = {1, 2, 3}$</th>
<th>Predicates:</th>
<th>$(x &gt; y), (y = 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg(y = 3)$</td>
<td>$(x=2, y=1)$</td>
<td>$(x=1, y=1)$</td>
<td></td>
</tr>
<tr>
<td>$\neg(y = 3)$</td>
<td>$(x=3, y=1)$</td>
<td>$(x=1, y=2)$</td>
<td></td>
</tr>
<tr>
<td>$\neg(y = 3)$</td>
<td>$(x=3, y=2)$</td>
<td>$(x=2, y=2)$</td>
<td></td>
</tr>
<tr>
<td>$(y = 3)$</td>
<td>$(x=1, y=3)$</td>
<td>$(x=1, y=3)$</td>
<td></td>
</tr>
<tr>
<td>$(y = 3)$</td>
<td>$(x=2, y=3)$</td>
<td>$(x=2, y=3)$</td>
<td></td>
</tr>
<tr>
<td>$(y = 3)$</td>
<td>$(x=3, y=3)$</td>
<td>$(x=3, y=3)$</td>
<td></td>
</tr>
</tbody>
</table>

**Model, property**

- **Init**

  **Check**

  **Property holds**

  - **OK**
  - **Refine**
  - **Concretize**

  **Concrete**

  **Counterexample**
CEGAR – Initial Abstraction

- Explicit value abstraction
  - Partition variables: visible / invisible
  - Track values for visible variables only
  - Label of a state: assignment, e.g. \((x = 1) \land (y = 2)\)

Variables: \(x, y, z\)
\(D_x = \{0, 1\}\), \(D_y = \{0, 1, 2\}\), \(D_z = \{0, 1\}\)
Visible = \(\{x, y\}\)

<table>
<thead>
<tr>
<th></th>
<th>(x=0)</th>
<th>(x=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y=0)</td>
<td>((x=0, y=0, z=0))</td>
<td>((x=1, y=0, z=0))</td>
</tr>
<tr>
<td></td>
<td>((x=0, y=0, z=1))</td>
<td>((x=1, y=0, z=1))</td>
</tr>
<tr>
<td>(y=1)</td>
<td>((x=0, y=1, z=0))</td>
<td>((x=1, y=1, z=0))</td>
</tr>
<tr>
<td></td>
<td>((x=0, y=1, z=1))</td>
<td>((x=1, y=1, z=1))</td>
</tr>
<tr>
<td>(y=2)</td>
<td>((x=0, y=2, z=0))</td>
<td>((x=1, y=2, z=0))</td>
</tr>
<tr>
<td></td>
<td>((x=0, y=2, z=1))</td>
<td>((x=1, y=2, z=1))</td>
</tr>
</tbody>
</table>
CEGAR – Model Checking

- **Traverse** abstract state space
  - Search strategy
- **Search for** error state
- **Optimizations**
  - On-the-fly
  - Incremental

Diagram:

1. Model, property → Init
2. Abstraction → Check
3. Property holds → OK
4. Abstract counterex → Concretize
5. Concrete → Counterexample
CEGAR – Concretization

- Traverse subset of concrete state space
  - Concretizable counterexample
  - Spurious counterexample
    - Failure state ($S_f$)
  - Use SMT solver, e.g. Microsoft Z3
    - $S_1 \land T_1 \land S_2 \land T_2 \land \ldots \land T_{n-1} \land S_n$
CEGAR – Abstraction Refinement

- **Classify states** mapped to the failure state
  - D = Dead-end: reachable
  - B = Bad: transition to next state
  - IR = Irrelevant: others

- **Goal:** finer abstraction mapping D and B to separate abstract states
  - SMT solver: interpolation formula \( \phi \)
  - Use \( \phi \) as predicate or extract its variables

---

Model, property

Init

Abstraction

Check

Property holds

OK

Abstract counterex

Concretize

Concretize

Refine

Counterexample

Concrete
CEGAR – Summary

- CEGAR is a general concept
  - Explore abstract state space
  - Refine abstraction if needed

- Many variants exist (for various formal models)
  - Abstract domains, e.g., predicates, explicit values, zones
  - Refinement strategies, e.g., interpolation, unsat cores
  - Exploration strategies, e.g., BFS, DFS
Research Questions

- **Integrate variants into common framework? Combine?**
  - A configurable CEGAR framework with interpolation-based refinements. Ákos Hajdu, Tamás Tóth, András Vörös, and István Majzik. FORTE 2016, vol. 9688 of LNCS.

- **Which variants perform well for given verification tasks?**
  - Exploratory analysis of the performance of a configurable CEGAR framework. Ákos Hajdu and Zoltán Micskei. PhD Mini-Symposium 2017, BME DMIS.
  - Towards evaluating size reduction techniques for software model checking. Gyula Sallai, Ákos Hajdu, Tamás Tóth, and Zoltán Micskei. VPT 2017. (Accepted)

- **Domain specific CEGAR variants?**
  - Exploiting hierarchy in the abstraction-based verification of statecharts using SMT solvers. Bence Czipó, Ákos Hajdu, Tamás Tóth, and István Majzik. FESCA 2017, vol. 245 of EPTCS.

Theta Verification Framework
Theta Verification Framework

Generic
Various kinds of formal models

Configurable
Different algorithms and strategies

Modular
Reusable and combinable modules

http://theta.inf.mit.bme.hu
Theta Verification Framework

### Architecture

**Formal models and language front-ends**
- AIGER
- PLC
- C source code
- UPPAAL XTA

**Transition systems**
- Control Flow Automata
- Timed Automata

### Verification back-end

**CEGAR loop**
- Abstractor
- Refiner

**SMT solver interface**
- States + transitions

**Abstract domain**
- Interpreter
Theta Verification Framework

- Configurability

**Abstract domain**
- Predicate
- Explicit value
- Zone
- Location
- Composition

**Refinement strategy**
- Binary interp. forw.
- Binary interp. backw.
- Sequence interp.
- Unsat core

**Search strategy**
- BFS
- DFS

**Initial precision**
- Empty
- Property-based

**Precision granularity**
- Constant
- Location-based

**Predicate split**
- Atoms
- Conjuncts
- Whole
Theta Verification Framework

- **Evaluation**
  - Really **diverse** results
  - Current research: data analysis & heuristics
Conclusions
Conclusions

- **Formal verification**
  - Formal model + property
  - Model checking

- **Abstraction-based methods**
  - CEGAR

- **Theta Framework**
  - Generic, modular, configurable
  - Evaluation

*hajdua@mit.bme.hu*
*inf.mit.bme.hu/en/members/hajdua*