A Method for Automatic Runtime Verification of Automata-Based Programs

Oleg Stepanov  oleg.stepanov@gmail.com

Anatoly Shalyto  shalyto@mail.ifmo.ru

Saint Petersburg State University of Information Technologies,
Mechanics and Optics
Problem Statement

Design a method for runtime verification of automata programs
Existing Approaches

- Rely on static verification, most commonly on model checking
- Use available verifiers like SPIN and Bogor
- Build Kripke structure which has exponential size of the program used to build it
Existing Approaches: Performance

- SPIN-based method can perform verification of models containing 100 to 500 automata depending on the number of transitions

- Methods based on model checking take exponential time to verify a system of automata
Runtime Verification

Runtime verification is verification of program execution traces which can be run in parallel with the verified program.
Advantages of Runtime Verification

- Allows for verification of larger systems of automata
- Verifies *implementation*, not model
- Can be used for soft handling of exceptional cases in critical applications
Method Performance

Depends on trace size

Depends on formula complexity

Does not depend on program complexity
Method Drawbacks

Does not guarantee valid program

Even worse for parallel programs
Runtime Verification: Workflow

Specification

LTL formula

Alternating Büchi automata

Verifier

Counter-example

Program

Execution trace

Manual translation
Samples of Alternating Büchi Automata

$\mathcal{A}(\square \varphi)$

$\mathcal{A}(\varphi)$

$\mathcal{A}(\varphi U \psi)$

$\mathcal{A}(\varphi W \psi)$

$\mathcal{A}(\varphi)$

$\mathcal{A}(\varphi)$

$\mathcal{A}(\varphi)$

$\mathcal{A}(\varphi)$

$true(\ast)$

$true$
Traversal of Alternating Büchi Automata

1. Depth-first traversal
   Choice depends on formula

2. Breadth-first traversal

3. Reverse traversal
   Optimal choice when entire trace is available
Systems of Mealy Automata
Components of the Method

1. Trace construction algorithm
2. Set of atomic propositions
3. Algorithm for evaluating propositions at each trace point
Trace Construction

\[ s_{1,1} s_{2,1} [e_{1,1} x_1 x_2 s_{1,2}] [e_{1,2} x_1 x_2 [e_{2,3} x_1 x_2 z_1 s_{2,2}] s_{1,3} ] \]

- **Trace header**
- **Handling section**
- **Nested section**
  - **Root event**
  - **Variable values**
  - **End state**
Atomic Propositions

$e_{i,j}$ — $i$-th automaton is handling event $e_j$

$x_i$ — value of input variable $x_i$ is true

$z_i$ — output action $z_i$ is performed

$s_{i,j}$ — $i$-th automaton is in state $s_{i,j}$
Method Performance

![Graph showing performance vs trace length]

Verification time, ms

Trace length, thousands of records

- \( G \times 1 \rightarrow F \ Z_1 \)
- \( G \times 1 \lor (F \ Z_1 \land F \ Z_2) \)
Further Research

- Apply test input generation algorithms
- Build an efficient implementation for breadth-first algorithm
- Apply to real large systems
- Investigate applicability of automata programming in dynamically executed environments
Questions?

oleg.stepanov@gmail.com
shalyto@mail.ifmo.ru

http://is.ifmo.ru