

Model Checking in Uppaal

Mohammad Mousavi

Halmstad University, Sweden

<http://ceres.hh.se/mediawiki/DIT085>

Testing and Verification (DIT085),
Chalmers and GU, February 13, 2015

Outline

Alternatives to Testing

Model Checking

Specification in Uppaal

Verification in Uppaal

Dynamic Testing

Dynamic testing: invoking faults and detecting failures through execution of the program code on an actual execution platform

Pros:

- ▶ Quick and scalable techniques
- ▶ Natural extension of programming skills

Cons:

- ▶ No proof of correctness
- ▶ Gaps and redundancies

Alternatives to Dynamic Testing

Static Analysis / Abstract Interpretation

1. **Approximating** the program behavior into a mathematical structure
2. Using analysis techniques to detect a **fixed category** of faults
3. **Refining** the approximation by removing the false negatives

Alternatives to Dynamic Testing

Static Analysis / Abstract Interpretation

1. **Approximating** the program behavior into a mathematical structure
2. Using analysis techniques to detect a **fixed category** of faults
3. **Refining** the approximation by removing the false negatives

Model Checking

1. Translating program or specification into a **behavioral model** on an abstract machine
2. Correctness properties as **logical formula**
3. Checking whether behavior satisfies formula, producing counter-example if it does not

Static Analysis: Division by Zero

Input(x)

Input(y)

...

$x = x - 1 ;$

$y = y/x$

Static Analysis: Division by Zero

Input(x)

Input(y)

...

if $x > 20$ **then**

$x = x - 1$;

end if

$y = y/x$

Static Analysis: Pros and

Pros

1. **Scalable** and efficient, often push button (integrated in IDEs)
2. Useful for common faults (e.g., division by zero, null pointer deref.)

Cons

1. Usually for a **fixed property**
2. Possibility of **false negatives**

Outline

Alternatives to Testing

Model Checking

Specification in Uppaal

Verification in Uppaal

Model Checking

Turing Award 2007 (abridged)

A program (i.e., model checker) can exhaustively construct every possible sequence of actions a system might perform, and for every action it could evaluate a property in logic. If the program found the property to be true for every possible sequence, the possible execution sequences form a model of the specified property.



Outline

Alternatives to Testing

Model Checking

Specification in Uppaal

Verification in Uppaal

Gossiping Girls: Specification

The Scene

1. n girls, each knowing a set of facts,
2. they call each other, and gossip so much that they know the same facts afterwards
3. continue until everyone knows everything

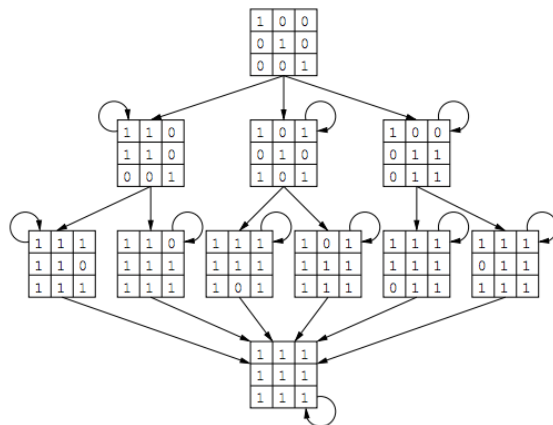


Gossiping Girls: Code Snippet

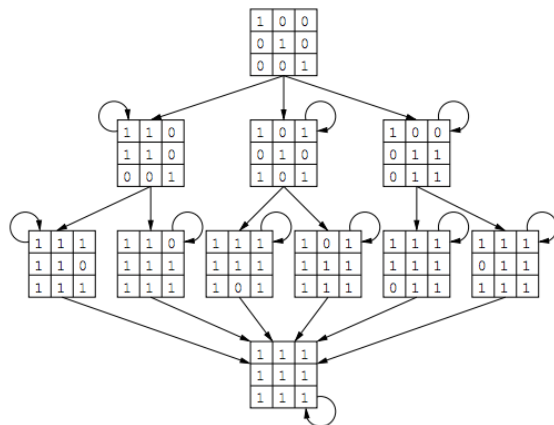
```
typedef int[1,3] girls;
bool knows[girls][girls];
void share (girls a, girls b) {
  for (c : girls) {
    knows[a][c] := knows[a][c] or knows[b][c];
    knows[b][c] := knows[a][c];
  }
}
```



Gossiping Girls: State Space

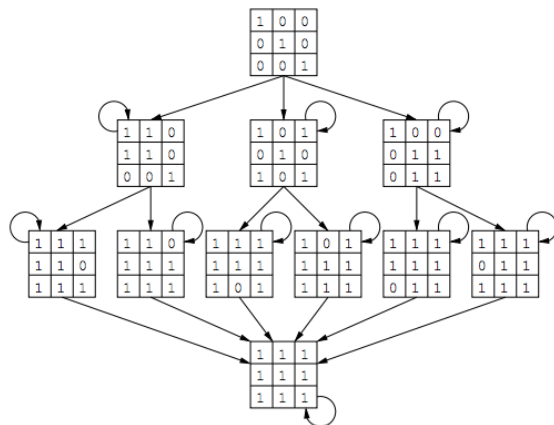


Gossiping Girls: State Space



How about more girls, say 6?

Gossiping Girls: State Space



How about more girls, say 6? **6 trillion** possible combinations!

Gossiping Girls: Property

Eventually every girl will know everything that every other girl knows.

Uppaal Tool

<http://www.uppaal.org>

- ▶ Developed at Uppsala and Aalborg (with contributions from other universities)
- ▶ Free for academic and private use
- ▶ Java-based implementation, socket-based server
- ▶ Toolsets for: **simulation**, **verification**, test case generation, optimization, statistical verification, and scheduling

Uppaal 101

System Descriptions : Networks of (Communicating) Timed Automata

Properties: Timed Computational Tree Logic (a sort of temporal logic)

Uppaal Templates

Timed Automata:

- ▶ Name
- ▶ Parameters
- ▶ Locations (nodes, states):
 - ▶ Name
 - ▶ Invariant
 - ▶ Initial
 - ▶ Urgent or Committed: time freezes, in case of committed state, one of the enabled committed states should be left next

Uppaal Templates

- ▶ Transitions (edges, vertices):
 - ▶ Select: choice of a parameter (to be read as “for some”)
 - ▶ Guards: logical conditions on variables and clocks
 - ▶ Synchronizations: messages sent and received on channels (see the next slide)
 - ▶ Updates: change of variable values, resetting clocks

Uppaal Templates

- ▶ Channels:
 - ▶ Hand-shaking synchronization: receiving and sending synchronizations must be enabled
 - ▶ Broadcast: sender always succeeds, as many receiving synchronizations as possible synchronize

Outline

Alternatives to Testing

Model Checking

Specification in Uppaal

Verification in Uppaal

Timed Computational Tree Logic

- ▶ Expressions on variables and location names
- ▶ Usual logical connectives (and, or, not, imply)
- ▶ path quantifiers: A in every execution vs. E in some execution
- ▶ temporal operators: \square globally in every state vs. $\langle \rangle$ eventually in some state,
 - ▶ $A \square p$ invariantly (at every state of every execution) p holds
 - ▶ $E \langle \rangle p$ possibly (there exists a state state in some execution) p holds
 - ▶ $A \langle \rangle p$ inevitably (there exists a state state in every execution) p holds
 - ▶ $p \dashrightarrow q$ “leads to” is an acronym for $A \square (p \text{ imply } A \langle \rangle q)$

Monitoring behavior

- ▶ To check for certain desired / forbidden sequence of state / transitions:
 - ▶ Define global variables to expose the state,
 - ▶ Make a monitor template that checks for a sequence of states / transitions using the global variables as guards,
 - ▶ Give the final state of the desired / forbidden order a name, e.g., “error”,
 - ▶ Create an instance of your monitor template with the rest of the system,
 - ▶ Check for reachability of “error” .

Jobshop

The Scene (simplified)

1. two workers at a jobshop, putting pegs into blocks,



Jobshop

The Scene (simplified)

1. two workers at a jobshop, putting pegs into blocks,
2. one hammer and one mallet available



Jobshop

The Scene (simplified)

1. two workers at a jobshop, putting pegs into blocks,
2. one hammer and one mallet available
3. 2 types of jobs:
 - ▶ easy: requiring either hammer or mallet,
 - ▶ difficult: requiring both



Jobshop

The Scene (simplified)

1. two workers at a jobshop, putting pegs into blocks,
2. one hammer and one mallet available
3. 2 types of jobs:
 - ▶ easy: requiring either hammer or mallet,
 - ▶ difficult: requiring both
4. finish after 3 jobs



Due to the late Robin Milner.

Acknowledgment

The material presented today is based on Frits Vaandrager's chapter on Uppaal; see the course page.

Liked It?

Also check out our new book...

