Automated Support for the Design and Validation of Fault Tolerant Parameterized Systems: a case study

F. Alberti\textsuperscript{1} S. Ghilardi\textsuperscript{3} E. Pagani\textsuperscript{3} S. Ranise\textsuperscript{2} G.P. Rossi\textsuperscript{3}

\textsuperscript{1}Università della Svizzera Italiana, Lugano, Switzerland

\textsuperscript{2}FBK-Irst, Trento, Italy

\textsuperscript{3}Università degli Studi di Milano, Milano, Italy

10\textsuperscript{th} International Workshop on Automated Verification of Critical Systems (AVOCS 2010)
Fault-tolerant systems

Systems that show a correct behavior regardless failures

- Parameterized
- Numerous and heterogeneous applications
  - Network protocols
  - E-commerce
  - Distributed databases
  - Sensor networks
  - Real-time systems

Parameterized verification

Checking that a system satisfies a given property regardless the number of processes
Desiderata on tools for parameterized verification

- Natural input language
- Counterexample (if the case)
- High degree of automation
  - As much as possible automatic verification
  - Avoid the introduction of bugs from user interactions

Problems

- Number of processes unknown
- Processes manipulate variables defined over unbounded domains, e.g.
  - Integer or real variables
  - Pointers to other processes of the system

⇒ we need a tool to handle infinite state systems
Our case study: the problem of Reliable Broadcast

Ingredients of the problem

- Process $p$ wants to send a message $m$ to all processes
- Broadcast primitives not available

$\Rightarrow$ $p$ must send $m$ to each process separately

Failures

Temporary (omission) or persistent (crash) failure may cause inconsistencies!
Reliable Broadcast: a classical solution

T. D. Chandra and S. Toueg.
Time and message efficient reliable broadcasts.

Parametric verification

Never been formally verified (to the best of our knowledge)
Our case study: Reliable Broadcast

Safety property: agreement
If a correct process delivers a message $m$, all correct processes deliver $m$

Correctness is defined w.r.t. different failure models
- Stopping failure
- Send-Omission
- General-Omission
- Arbitrary (Byzantine or malicious)
## Why MCMT?

Because it matches the desiderata for infinite state verification tools

- Natural input language
- Counterexample (if the case)
- High degree of automation
  - As much as possible automatic verification
  - Avoid the introduction of bugs from user interactions

---

S. Ghilardi and S. Ranise.

**MCMT: a Model Checker Modulo Theories.**

Main features

- Symbolic approach: formulae are used to represent set of states
- Declarative specification of topology and data with first order theories
- Predictable symbolic model checking supported by SMT-solvers
- Accept hints from user and use them without compromising correctness
- Easily integrated in the design methodology
Example

initial (universal p:nat) {
    estimate[p] = unknown AND round[p] = 1 AND
decided[p] = false AND faulty[p] = false
}

unsafe (existential p1:nat, existential p2:nat) {
}

transition (existential p:nat, universal all:nat) {
    guard: (coord[p] = false) AND (aCoord[p] = false)
    uguard: (coord[all] = false)
    update:
        round := 1;
        coord[p] := true;
        done := lambda (j:nat) { false }
        request := false;
}
Assumptions on termination

Termination of the algorithm is ensured by some assumption on the topology and data:

- Assumptions on topology are satisfied by many theories of interest like:
  - Pure equality
  - Linear orders
  - Trees/forests
  - Graphs

  ... but are not met by ring topology.

- We want $T_I$ and $T_E$ to be disjoint
Using invariants

- Large protocols may require big amount of resources (both time and space).
- Using invariants can help to reduce resource requirements, but...
- ...are we sure that we are using real invariants?

Solution

- Declarative approach: if \( S \) is safe w.r.t. \( \varphi \), then \( \neg \varphi \) is a safe invariant for the system.
- Draw a plan of work!
- We can tell to MCMT:
  1. Try to check these invariants: \( \varphi_1, \varphi_2, \varphi_3, ... \)
  2. Use only those you have found to be real safe invariants in the main verification process.
Our case study: Reliable Broadcast

Metodology

<table>
<thead>
<tr>
<th>Crash</th>
<th>More difficult failure models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send-Omission</td>
<td></td>
</tr>
<tr>
<td>General-Omission</td>
<td></td>
</tr>
</tbody>
</table>

Pr1 → Pr1’ → Pr2 → Pr3

Refinement
Our case study: Reliable Broadcast

Metodology

- Crash
  - Pr1
  - Pr1' → Pr2

- Send-Omission
  - Pr1 → Pr1' → Pr2

- General-Omission
  - Pr2 → Pr3

... More difficult failure models
### Our case study: Reliable Broadcast

#### Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1.18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian**
- **MCMT (v. 1.0.1) executed in default mode**
## Our case study: Reliable Broadcast

### Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td>UNSAFE</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1.18</td>
<td>17.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td>20.009</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td>11 tr.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian
- MCMT (v. 1.0.1) executed in default mode
## Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td>UNSAFE</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1,18</td>
<td>17,66</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td>20.009</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td>11 tr.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian
- MCMT (v. 1.0.1) executed in default mode

Shorter than [1]
### Our case study: Reliable Broadcast

#### Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td></td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td>9</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td>16</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1.18</td>
<td>17.66</td>
<td>1.709,93</td>
<td></td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td></td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td></td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td></td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

- Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian
- **MCMT (v. 1.0.1)** executed in default mode
### Our case study: Reliable Broadcast

#### Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td></td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td>9</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td>16</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1.18</td>
<td>17.66</td>
<td>1.709,93</td>
<td></td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td></td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td></td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td></td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Same as in [1]

- Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian
- MCMT (v. 1.0.1) executed in default mode
## Our case study: Reliable Broadcast

### Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td>SAFE</td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td>16</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1.18</td>
<td>17.66</td>
<td>1.709,93</td>
<td>4.719,51</td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td>11.158 (-1.290)</td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td>2.558.986</td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td>×</td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>19 (+7)</td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

- Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian
- MCMT (v. 1.0.1) executed in default mode
## Our case study: Reliable Broadcast

### Results

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Result</strong></td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td>SAFE</td>
</tr>
<tr>
<td><strong># State variables</strong></td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td><strong># Transitions</strong></td>
<td>13</td>
<td>16</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td><strong>Time (s)</strong></td>
<td>1.18</td>
<td>17.66</td>
<td>1.709,93</td>
<td>4.719,51</td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td>11.158 (-1.290)</td>
</tr>
<tr>
<td><strong># SMT calls</strong></td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td>2.558.986</td>
</tr>
<tr>
<td><strong>Length unsafe trace</strong></td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td>×</td>
</tr>
<tr>
<td><strong># Invariants</strong></td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>19 (+7)</td>
</tr>
<tr>
<td><strong>Max # processes</strong></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

*Intel Core2 Duo @ 2.66 GHz, 2 GB RAM, Linux Debian*

*MCMT (v. 1.0.1) executed in default mode*
Our case study: Reliable Broadcast

Remarks

- No toy problems!
- Finding unsafe traces is not trivial!
The problem

- Verifying large protocols requires a large amount of resources
- Invariants can help reducing time and space requirements
- For the last protocol:
  - Without invariants, MCMT suffers a memory out after 1 day of computation
  - With seven invariants, MCMT ends the computation after 78 mins.
Our case study: Reliable Broadcast

Adding hints as invariants

Features of invariants

The seven invariants concerns how data structures are updated (no deep properties of the protocol!).

An example

- Rotating coordinator paradigm.
- Cyclically one process becomes the coordinator of the network.
- In the network there’s only one coordinator.
- Local boolean array coord to say which process is the coordinator;

A natural candidate invariant is this:

$$\forall i, j ((\text{coord}[i] = \text{true} \land \text{coord}[j] = \text{true}) \rightarrow i = j)$$
Another example

- Send Omission failure model
- Processes become coordinators in order of address
- We use a local boolean array aCoord to say which processes have already been coordinator;

\[
\forall i, j \left( \left( aCoord[i] = true \land aCoord[j] = false \land \right) \left( crash[i] = false \land crash[j] = false \right) \rightarrow i < j \right)
\]
Adding hints as invariants

Prooving Invariants

Recall that we don’t have to prove invariants by hand! We can use MCMT to prove them...

\[ \forall i, j ((\text{coord}[i] = \text{true} \land \text{coord}[j] = \text{true}) \rightarrow i = j) \]

is a safe invariant if the system is safe w.r.t. this unsafe configuration:

\[ \neg \forall i, j ((\text{coord}[i] = \text{true} \land \text{coord}[j] = \text{true}) \rightarrow i = j) \]
Our case study: Reliable Broadcast
Adding hints as invariants

Prooving Invariants
Recall that we don’t have to prove invariants by hand! We can use MCMT to prove them...

\( \forall i, j ((\text{coord}[i] = \text{true} \land \text{coord}[j] = \text{true}) \rightarrow i = j) \)

is a safe invariant if the system is safe w.r.t. this unsafe configuration:

\( \neg \forall i, j ((\text{coord}[i] = \text{true} \land \text{coord}[j] = \text{true}) \rightarrow i = j) \)

or, rewriting it,

\( \exists i, j (\text{coord}[i] = \text{true} \land \text{coord}[j] = \text{true} \land i \neq j) \)

that is, the unsafe configuration that tells that there are two coordinators in the system!

F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, G.P. Rossi
Automated support for the design and validation of ...
Conclusions and Future works

**Conclusions**

- Parameterized verification of fault-tolerant protocols:
  - ✔ Symbolic representation $\Rightarrow$ heterogeneous applications
  - ✔ *Candidates* invariants $\Rightarrow$ “secure” interaction with user
  - ✗ Manual search for invariants
  - ✔ Interactive methodology for developing complex protocols

- Analyzed protocols:
  - ✔ First formal parametric verification of this protocols
  - ✔ Completely automatic (except the fourth)
  - ✔ Low resource consumption

**Future works**

- More difficult failure models (e.g. *general omission*)
- Timing constraints
Why General Omission is difficult?

**Formalization**

- New assumptions on the protocol:
  - Processes know the size of the network $n$
  - At least $\frac{n+1}{2}$ are correct

**Verification**

- Processes handle pointers to other processes identifiers.
- Processes perform arithmetic operations:
  - Processes count how many message they receive.
  - We can formalize it adding a transition that increment a counter.
  - Without acceleration, this lead to divergence.
Thank you!
Questions?

Francesco Alberti
francesco.alberti@usi.ch