Mutation of Formally Verified SysML Models

Ludovic Apvrille, Bastien Sultan, Oana Hotescu, Pierre de Saqui-Sannes, and Sophie Coudert

Modelsward’2023, Lisbon
Two new trends in MDE: agility, digital twins

**Agile / incremental development**
- +: Expected to improve system reliability
- +: It helps handling complexity
- -: Improving models can be cumbersome
- -: Full system verification must be done again after each improvement

**Digital twin**
- +: Used for handling problems occurring in systems in exploitation (e.g., an attack)
  - The twin helps handle complexity by reasoning on an abstract view of the system
- -: Improving model can be cumbersome
- -: Full system verification must be done again after each system modification
Usual Verification Approach

Verification
*(model-checking, etc.)*

Proof result
Usual Verification After System Update

Verification of Model Properties

Proof result

Updated Model

Properties

(Full) Verification

Proof result
Towards Incremental Verification

Our idea: reusing previous proof results after system update
Our Proposal

Incremental verification

- Applied to reachability properties given in CTL
- Automated
- Verification reuses verification results obtained before system update

Application to design performed with SysML models

- Block diagrams
- State machine diagrams
- Updates on model are called mutations
Related Work

**Formal verification of SysML models**
- SysML $\rightarrow$ formal specification
- Petri nets, NuSMV, Timed automata (UPPAAL), RT-LOTOS, ...
- E.g, [Delatour et al. 1998; Szmuc 2018; Huang et al. 2019; Rahim et al. 2020]

**Incremental verification**
- Compositional verification [Xie et al., 2022]
- Correct-by-construction [Bougacha et al., 2022]

**Model mutation**
- Investigating specification change [Aichernig et al., 2013]
- Understanding impact of attacks on systems [Sultan et al., 2017]
Contribution Overview

Properties P

Model M1
T, F, RG
prover
Set of mutations

Model M2
Mutation Application
DG generator
Dependency Graph

Properties P'

Proof optimizer

Model M3
...
Mutation Application
Set of mutations
Models and Tools

- Properties $P$
- Model $M_1$
- $T, F, RG$
- Set of mutations
- Model $M_2$
- Mutation Application
- $DG$
- Dependency Graph
- Properties $P'$
- Proof optimizer
- Model $M_3$
- Set of mutations
- TTool Internal model checker
- AMULET compiler
- SysML block and state machine diagrams
- CTL

TTool
Mutations: Definition and Notation

Incremental verification currently supports only additions to models

<table>
<thead>
<tr>
<th>Block diagrams</th>
<th>State machine diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Adding a block</td>
<td>• Adding a state</td>
</tr>
<tr>
<td>• $\mathcal{M} \xrightarrow{\mathcal{B}^+(B)} \mathcal{M}'$</td>
<td>• $\mathcal{M} \xrightarrow{\text{State}^+(B,s)} \mathcal{M}'$</td>
</tr>
<tr>
<td>• Adding a port, connecting two ports</td>
<td>• Adding a transition</td>
</tr>
<tr>
<td>• Adding attributes / signals to blocks</td>
<td>• $\mathcal{M} \xrightarrow{\text{Trans}^+(B,t)} \mathcal{M}'$</td>
</tr>
<tr>
<td>• $\mathcal{M} \xrightarrow{\text{Attr}^+(B,a)} \mathcal{M}'$</td>
<td>• Adding a guard, an <em>after</em>, an action to a transition</td>
</tr>
</tbody>
</table>

Contribution: Illustration with An Example

State END2 is proved as reachable
Contribution: Illustration with An Example (Cont.)

We apply mutations to the state machine of Receiver

![State Machine Diagram]

- `addState(END3)`
- Then
- `addTransition(Waiting, END3, after(2))`

Is END2 still reachable?
**Contribution: Illustration with An Example (Cont.)**

Proof Optimizer

1. Our algorithm first computes that: END1 reachable $\Rightarrow$ END2 reachable
2. Our algorithm replaces the reachability of END2 by the reachability of END1 in the list of properties to be proved
3. The model is reduced for the proof of the reachability of END1
Contribution: Illustration with An Example (Cont.)

Resulting model, and proof of reachability:

END1 is not reachable so END2 is not reachable after mutation
Contribution: Proof Optimizer Algorithm

**Input:** proof that state $s$ is reachable in initial Design $D_I$. New design $D_M$.

**Output:** set of properties to be proved on $D_M$

1. New logical paths to $s$ in $D_M$ are computed and added to $Paths$ along with their immediate successors.

2. Computation of the shortest prefixes in $Paths$ that cannot lead to any mutated element. If an immediate successor is reachable then $s$ is reachable (Proof done for $D_I$): exit.

3. Identification of new paths (due to new loops, choices, variables, ...) leading to $s$ that have not been proved for $D_I$. Computation of the reachability of $s$ via these paths.
Contribution: Discussion

• Performance trade-off between:
  • Reproving the same properties on the new design
  • Computing potentially simpler properties to be proved on the new design

A performance study follows...
Case Study: TSN

Time-Sensitive Networking (TSN) [IEEE 802.1]

- Guaranteed bounded latency, low packet delay variation, and low packet loss
- Adapted to safety-critical systems with deterministic real-time communication
- Built upon:
  - Transmitting End Systems $Tx\ ES$ and Receiving End System $Rx\ ES$
  - Switches $SW$ and network paths
Case Study: Model

2 Tx ES, 2 Rx ES, 2 SW. More than 20 blocks, complex state machines
### Case Study: Mutations and Performance

- **System** $S_1$: 1 Tx ES, 2 SW, 1 Rx ES, 2 flows
- **Mutation** $M_1 = S_1 \xrightarrow{+1\text{TxES,}+2\text{SW,}+1\text{RxES,}+2\text{flows}} S_2$

<table>
<thead>
<tr>
<th>Reachability</th>
<th>States/Transitions</th>
<th>Proof time (ms)</th>
<th>Muta tion</th>
<th>States Transitions</th>
<th>Proof time (ms)</th>
<th>Proof time (ms)</th>
<th>DG: vertices/edges/time to generate</th>
</tr>
</thead>
<tbody>
<tr>
<td>RG generation</td>
<td>2k/3k</td>
<td>16</td>
<td>$M_1$</td>
<td>13k/50k</td>
<td>240126</td>
<td></td>
<td>617/934/5ms</td>
</tr>
<tr>
<td>Get packet in flow 0</td>
<td>-</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>227</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Get packet in flow 2</td>
<td>-</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>231</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Get packet in SW#2</td>
<td>-</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>232</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Case Study: Mutations and Performance (Cont.)

- Mutation $M_2 = S_3 \xrightarrow{+1 \text{flow}} S_4$

<table>
<thead>
<tr>
<th>Reachability</th>
<th>States/Transitions</th>
<th>Proof time (ms)</th>
<th>M2</th>
<th>States Transitions</th>
<th>Proof time (ms)</th>
<th>Proof time (ms)</th>
<th>DG: vertices/edges/time to generate</th>
</tr>
</thead>
<tbody>
<tr>
<td>RG generation</td>
<td>80k/200k</td>
<td>292</td>
<td>M2</td>
<td>300k/677k</td>
<td>1170</td>
<td></td>
<td>389/594/2ms</td>
</tr>
<tr>
<td>Get packet in flow 3</td>
<td>-</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>11</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Get packet in flow 0</td>
<td>-</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Get packet in SW#3</td>
<td>-</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>10</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion and Future Work

Better agility in design

- Specification of mutations
- Incremental verification
- Demonstrated in the scope of a complex real-time system (TSN)
  - Proof time reduced for all tested reachability properties

Improvements

- Decrease complexity of our incremental verification approach
- Extension to more complex CTL properties
- Support for deletion mutations (today: only model additions)
- Full implementation in TTool
Questions?

Download TTool!

• http://ttool.telecom-paris.fr/
