Mutation of Formally Verified SysML Models

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

Modelsward'2023, Lisbon

Context

Two new trends in MDE: agility, digital twins

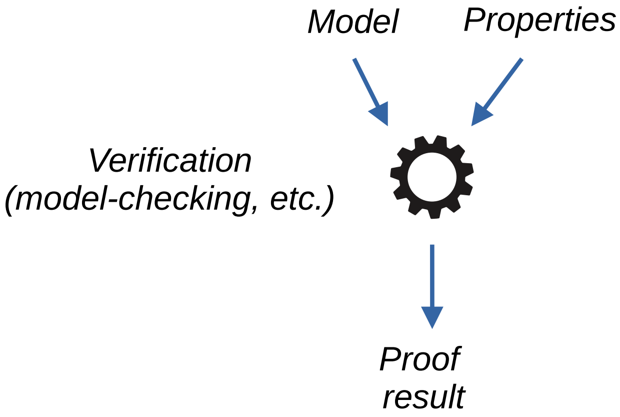
Agile / incremental development

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

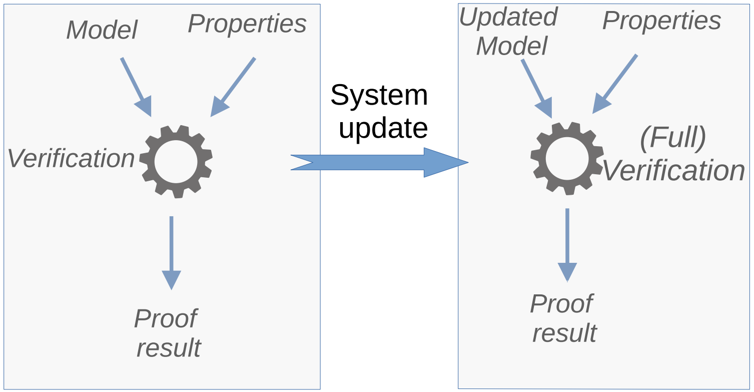
Digital twin

- +: 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

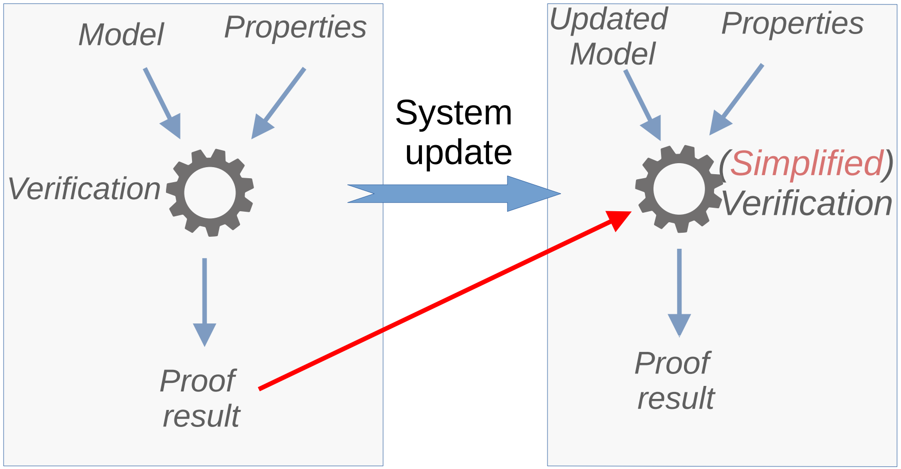


Usual Verification After System Update



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]

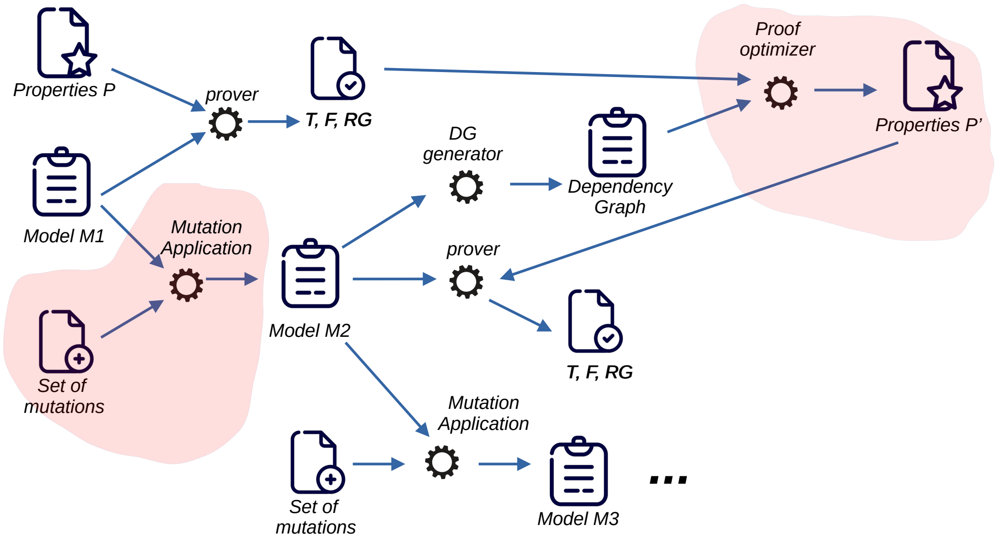
Incremental verification

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

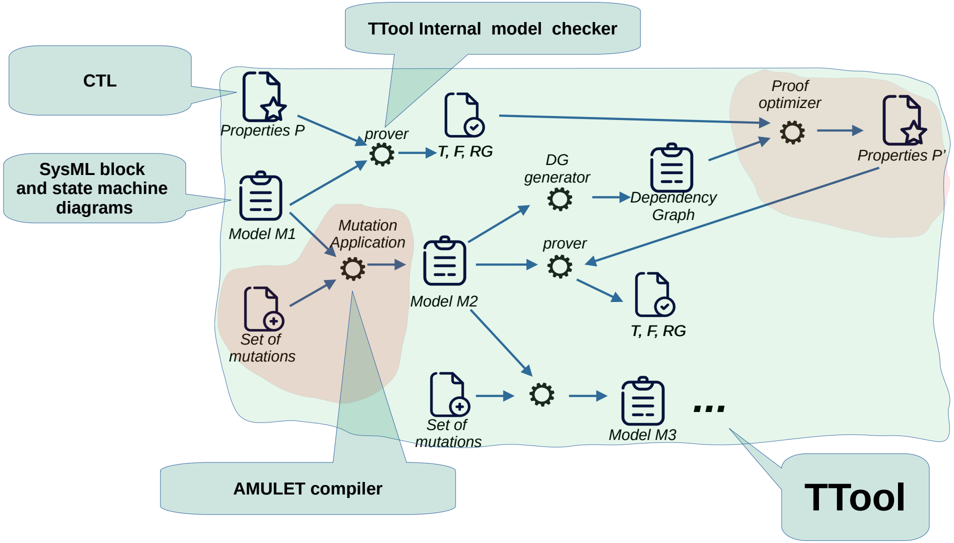
Model mutation

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

Contribution Overview



Models and Tools



Mutations: Definition and Notation

Incremental verification currently supports only additions to models

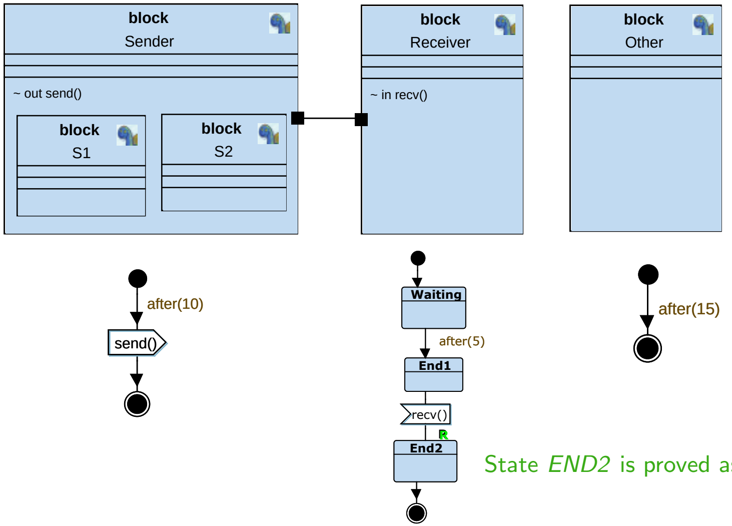
Block diagrams

- Adding a block
  - $\mathcal{M} \xrightarrow{\mathfrak{B}^+(B)} \mathcal{M}'$
- Adding a port, connecting two ports
- Adding attributes / signals to blocks
  - $\mathcal{M} \xrightarrow{\text{Attr}^+(B,a)} \mathcal{M}'$

State machine diagrams

- Adding a state
  - $\mathcal{M} \xrightarrow{\text{State}^+(B,s)} \mathcal{M}'$
- Adding a transition
  - $\mathcal{M} \xrightarrow{\text{Trans}^+(B,t)} \mathcal{M}'$
- Adding a guard, an *after*, an action to a transition

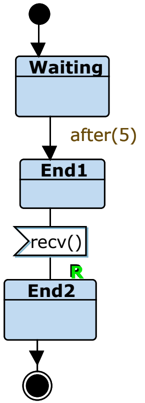
Contribution: Illustration with An Example



State *END2* is proved as reachable

Contribution: Illustration with An Example (Cont.)

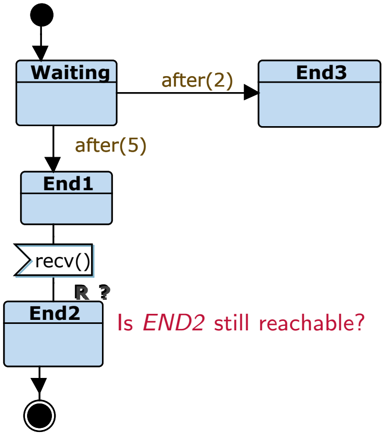
We apply mutations to the state machine of *Receiver*



$\xrightarrow{addState(END3)}$

then

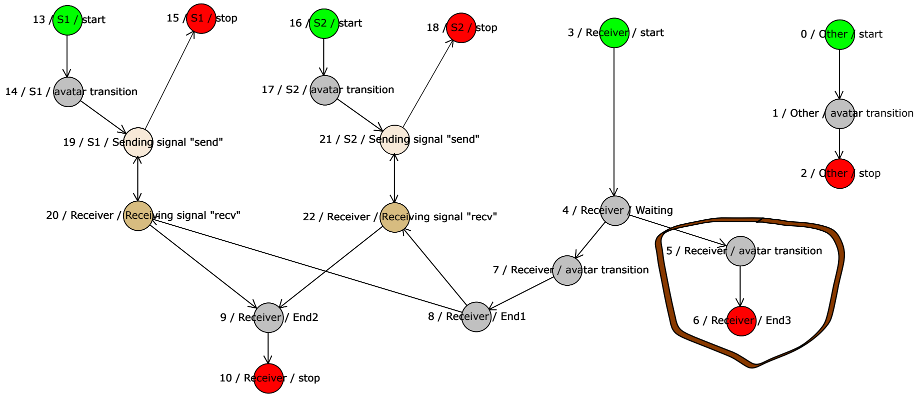
$\xrightarrow{addTransition(Waiting, END3, after(2))}$



Is *END2* still reachable?

Contribution: Illustration with An Example (Cont.)

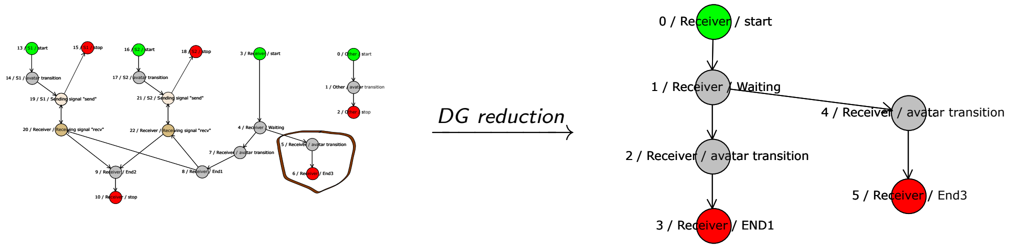
Dependency graph of the system after mutation. Brown circle: added elements.


13 / S1 / start
15 / S1 / stop
16 / S2 / start
18 / S2 / stop
3 / Receiver / start
0 / Other / start
14 / S1 / avatar transition
17 / S2 / avatar transition
1 / Other / avatar transition
19 / S1 / Sending signal "send"
21 / S2 / Sending signal "send"
2 / Other / stop
20 / Receiver / Receiving signal "recv"
22 / Receiver / Receiving signal "recv"
4 / Receiver / Waiting
5 / Receiver / avatar transition
7 / Receiver / avatar transition
9 / Receiver / End2
8 / Receiver / End1
6 / Receiver / End3
10 / Receiver / stop

Contribution: Illustration with An Example (Cont.)

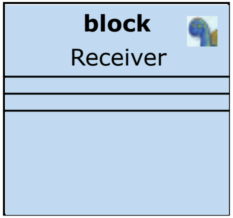
Proof Optimizer

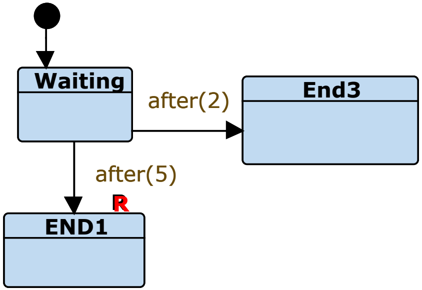
1. Our algorithm first computes that: END1 reachable $\Rightarrow$ END2 reachable
2. Our algorithm replaces the reachability of END2 by the reachability of END1 in the list of properties to be proved
3. The model is reduced for the proof of the reachability of END1


13 / S1 / start
15 / S1 / stop
16 / S2 / start
18 / S2 / stop
3 / Receiver / start
0 / Other / start
14 / S1 / avatar transition
17 / S2 / avatar transition
1 / Other / avatar transition
19 / S1 / Sending signal "send"
21 / S2 / Sending signal "send"
2 / Other / stop
20 / Receiver / Receiving signal "recv"
22 / Receiver / Receiving signal "recv"
4 / Receiver / Waiting
5 / Receiver / avatar transition
7 / Receiver / avatar transition
9 / Receiver / End2
8 / Receiver / End1
6 / Receiver / End3
10 / Receiver / stop
DG reduction
0 / Receiver / start
1 / Receiver / Waiting
4 / Receiver / avatar transition
2 / Receiver / avatar transition
5 / Receiver / End3
3 / Receiver / 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

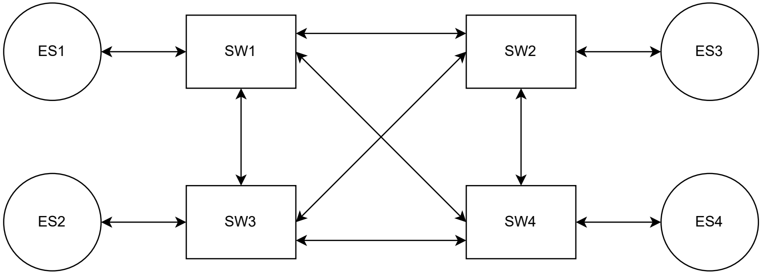
- Performance trade-off between:
  - Reproving the same properties on the new design
  - Computing potentially simpler properties to be proved on the new design

A performance study follows. . .

Case Study: TSN

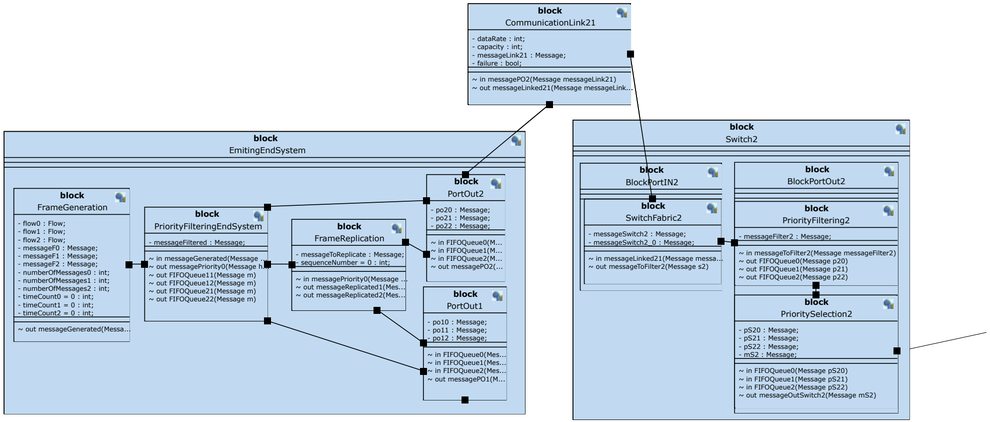
Time-Sensitive Networking (TSN) [IEEE 802.1]

- 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



block
CommunicationLink21
- dataRate : int;
- capacity : int;
- messageLink21 : Message;
- failure : bool;
~ in messagePO2(Message messageLink21)
~ out messageLinked21(Message messageLink...
block
EmitingEndSystem
block
FrameGeneration
- flow0 : Flow;
- flow1 : Flow;
- flow2 : Flow;
- messageF0 : Message;
- messageF1 : Message;
- messageF2 : Message;
- numberOfMessages0 : int;
- numberOfMessages1 : int;
- numberOfMessages2 : int;
- timeCount0 = 0 : int;
- timeCount1 = 0 : int;
- timeCount2 = 0 : int;
~ out messageGenerated(Messa...
block
PriorityFilteringEndSystem
- messageFiltered : Message;
~ in messageGenerated(Message ...
~ out messagePriority0(Message h...
~ out FIFOQueue11(Message m)
~ out FIFOQueue12(Message m)
~ out FIFOQueue21(Message m)
~ out FIFOQueue22(Message m)
block
FrameReplication
- messageToReplicate : Message;
- sequenceNumber = 0 : int;
~ in messagePriority0(Message ...
~ out messageReplicated1(Mes...
~ out messageReplicated2(Mes...
block
PortOut2
- po20 : Message;
- po21 : Message;
- po22 : Message;
~ in FIFOQueue0(M...
~ in FIFOQueue1(M...
~ in FIFOQueue2(M...
~ out messagePO2(...
block
PortOut1
- po10 : Message;
- po11 : Message;
- po12 : Message;
~ in FIFOQueue0(Mes...
~ in FIFOQueue1(Mes...
~ in FIFOQueue2(Mes...
~ out messagePO1(M...
block
Switch2
block
BlockPortIN2
block
SwitchFabric2
- messageSwitch2 : Message;
- messageSwitch2_0 : Message;
~ in messageLinked21(Message messa...
~ out messageToFilter2(Message s2)
block
BlockPortOut2
block
PriorityFiltering2
- messageFilter2 : Message;
~ in messageToFilter2(Message messageFilter2)
~ out FIFOQueue0(Message p20)
~ out FIFOQueue1(Message p21)
~ out FIFOQueue2(Message p22)
block
PrioritySelection2
- pS20 : Message;
- pS21 : Message;
- pS22 : Message;
- mS2 : Message;
~ in FIFOQueue0(Message pS20)
~ in FIFOQueue1(Message pS21)
~ in FIFOQueue2(Message pS22)
~ out messageOutSwitch2(Message mS2)

Case Study: Mutations and Performance

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

Reachability	States/ Transi- tions	Proof time (ms)	Muta tion	States Transitions	Proof time (ms)	Proof time (ms)	DG: vertices/edges/ time to generate
					no red.	*reduction*	
RG generation	2k/3k	16	M_1	13k/50k	240126		617/934/5ms
Get packet in flow 0	-	5		-	227	2	
Get packet in flow 2	-	7		-	231	2	
Get packet in SW#2	-	5		-	232	2	

Case Study: Mutations and Performance (Cont.)

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

Reachability	States/ Transi- tions	Proof time (ms)	Muta tion	States Transitions	Proof time (ms)	Proof time (ms)	DG: vertices/edges/ time to generate
					no red.	*reduction*	
RG generation	80k/200k	292	M2	300k/677k	1170		389/594/2ms
Get packet in flow 3	-	8		-	11	7	
Get packet in flow 0	-	9		-	10	5	
Get packet in SW#3	-	7		-	10	4	

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/

AN OPEN SOURCE TOOLKIT PROVIDED BY:

TELECOM Paris

IP PARIS

- *L. Apvrille, P. de Saqui-Sannes, O. Hotescu. and A. Calvino, "SysML Models Verification Relying on Dependency Graphs", In Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development, ISBN 978-989-758-550-0, ISSN 2184-4348, pages 174-181*
- *L. Apvrile, B. Sultan, O. Hotescu, P. de Saqui-Sannes, S. Coudert, "Mutation of Formally Verified SysML Models", Proccedings of the 11th internationl conference on Model-Based Software and Systems Engineering (Modelsward'2023), Lisbon, Portual, Feb. 19-21, 2023*