

# Statistical Model Checking for SystemC Models

Van Chan Ngo   Axel Legay   Jean Quilbeuf

Inria Rennes, France

HASE 2016, San Francisco, USA

# Executive summary

- PMC: **state space exploration** is infeasible for large systems
  - Partial Order Reduction
  - Symbolic Model Checking
  - SAT-based Bounded Model Checking
  - Predicate Abstraction
  - Counterexample Guided Abstraction Refinement
- SMC: often easier to simulate a system
- Our goal: provide **probabilistic guarantees of correctness** of stochastic SystemC models using a number of simulations
  - How to define an execution trace?
  - How to generate each execution trace?
  - How many simulation runs to make?

# An example



- Message's length and FIFO's buffer are **fixed** (i.e. of 10)
- Producer writes 1 character to the FIFO with **probability  $p_1$**  every 1 time unit
- Consumer reads 1 character from the FIFO with **probability  $p_2$**  every 1 time unit
- Quantitative analysis: What is the **probability that messages are transferred completely** within 15 time units during 10000 time units of operation?
- Qualitative analysis: Is this probability **at least 0.6**?

## A solution - PMC

- Given a **stochastic model**  $\mathcal{M}$  such as a Markov chain
- A property  $\varphi$  expressed in **Bounded Linear Temporal Logic** (BLTL) and a probability threshold  $\theta \in (0, 1)$
- Does  $\mathcal{M}$  satisfy  $\varphi$  with probability at least  $\theta$ ?

$$\mathcal{M} \models \text{Pr}_{\geq \theta}(\varphi)$$

**Example:** messages are transferred completely within  $T_1$  time units during  $T$  time units of operation

$$G_{\leq T}((c\_read = '\&') \rightarrow F_{\leq T_1}(c\_read = '@'))$$

# PMC - Scalability

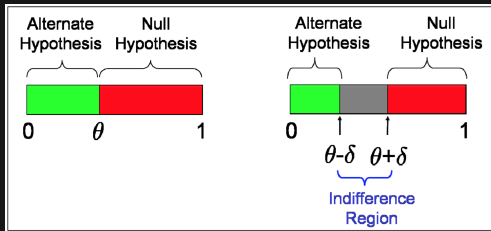
- PMC is infeasible for large systems due to the **state space exploration**
  - PMC employs symbolic model checking which can scale to  $\sim 10^{100}$  states
  - Scalability** depends on the structure of the system
- PMC does not **work directly** with SystemC models
  - Formal model is sometime much **over-approximated**. It cannot capture the concrete implementation of the system
- **Example**: PRISM checker created at Oxford and Birmingham

## Another solution - SMC

- Associate the  $i^{th}$  execution trace with a random variable  $B_i$  having a Bernoulli distribution,  
 $\Pr[B_i = 1] = p, \Pr[B_i = 0] = 1 - p$
- An observation  $b_i = 1$  if the trace satisfies the property,  
 $b_i = 0$ , otherwise
- Decide between two hypotheses:  
Null hypothesis:  $H : p \geq \theta$   
Alternate hypothesis:  $K : p < \theta$
- Or estimate the probability  $p$  instead of hypothesis testing  
Simulation is feasible for many more systems  
Easier to parallelize  
Answers may be wrong. But error probability can be bounded (i.e. at most  $\alpha \sim 0$ )  
Simulation is incomplete

# SMC - Existing work

- [Younes et al. 06, CMU] use Wald's Sequential Probability Ratio Test
- The simple null hypothesis  $H_0 : p \geq p_0 = \theta + \delta$
- The simple alternate hypothesis  $H_1 : p < p_1 = \theta - \delta$



- [Plasma Lab, Inria] checker uses MonteCarlo method, Chernoff and Hoeffding bounds, etc. to estimate the probability

$$\tilde{p} = \frac{1}{n} \sum_{i=1}^n b_i \text{ st } \Pr[|\tilde{p} - p| < \delta] \geq 1 - \alpha$$

# BLTL

- An extension of LTL with time bounds on temporal operations (i.e.  $\varphi_1 U_{\leq T} \varphi_2$ )

- The **semantics** of BLTL for a trace suffix

$\omega^k = (s_k, t_k), (s_{k+1}, t_{k+1}), \dots$  is defined as follows

$$\omega^k \models \text{true} \text{ and } \omega^k \not\models \text{false}$$

$$\omega^k \models p, p \in \text{AP} \text{ iff } p \in L(s_k)$$

$$\omega^k \models \varphi_1 \wedge \varphi_2 \text{ iff } \omega^k \models \varphi_1 \text{ and } \omega^k \models \varphi_2$$

$$\omega^k \models \neg \varphi \text{ iff } \omega^k \not\models \varphi$$

$$\omega^k \models \varphi_1 U_{\leq T} \varphi_2 \text{ iff there exists an integer } i \text{ such that}$$

$$\omega^{k+i} \models \varphi_2$$

$$\sum_{0 < j \leq i} (t_{k+j} - t_{k+j-1}) \leq T$$

$$\text{for each } 0 \leq j < i, \omega^{k+j} \models \varphi_1$$



# SystemC model state

A state is an evaluation of variables which represent

- Simulation kernel state

- Current phase of the simulation scheduler (i.e. delta-cycle notification, simulation-cycle notification)

- Events notified during the execution of the model

- SystemC model state: full state of the C++ code

- All module's attributes,

- Location of the program counter (i.e. executed statement, function call)

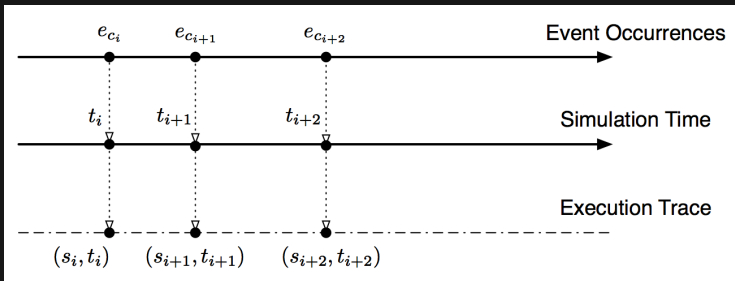
- Call stack (i.e. function parameters and return values)

- Status of module processes

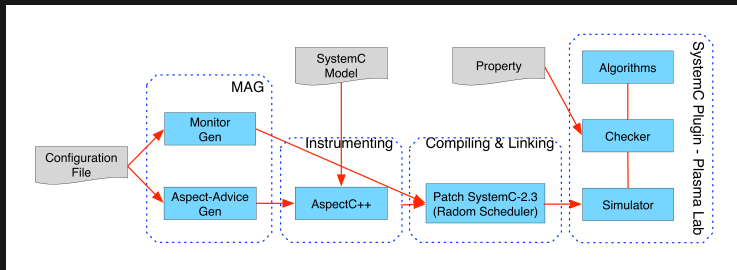
Note: external libraries are considered as block boxes

# Temporal resolution

- Is a set of Boolean expressions, called **temporal events**, defined over the simulation kernel states, location of the program counter, and processes' status
- Whenever a temporal event is **true**, a **new state** is sampled
- A time unit is duration between two **event occurrences**
- States are **snapshots** of system at event occurrences



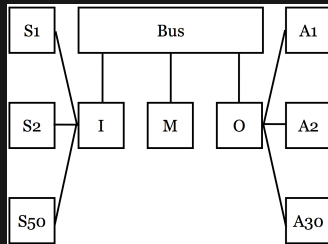
# SMC for SystemC models



- **MAG**: automatically instruments SystemC model with the help of AspectC++ in order to generate the traces and communicate with the checker
- **SystemC plugin**: communicates with the instrumented model and applies appropriate statistical algorithms provided by Plasma Lab

# Case study - Dependability analysis

- 50 groups of 3 sensors, 30 groups of 2 actuators
- Main, input and output processors communicate via a reliable bus



Component	Mean time
Sensor	<i>1 month</i>
Actuator	<i>2 months</i>
Transient Fault	<i>1 day</i>
Processor	<i>1 year</i>
Reboot to Repair	<i>30 seconds</i>

# Dependability analysis

- Time to failure of sensors, actuators and processors and time to repair of I/O processors can be modeled by exponential distributions
- The reliability is modelled as a Continuous Time Markov Chain (CTMC)

Sensor group: 4 states

Actuator group: 3 states

Main processor: 2 states

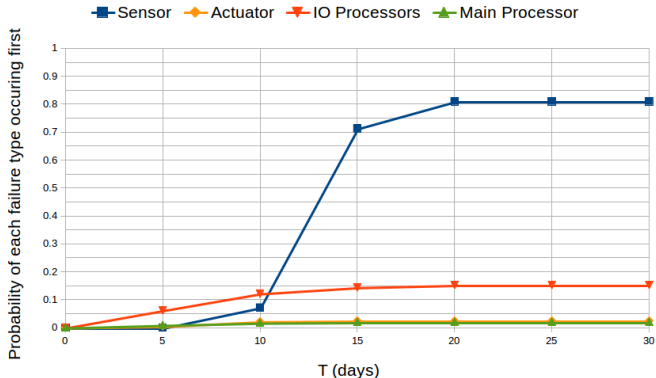
I/O processors: 3 states (including 1 state of transient failure)

The model has  $4^{50} \times 3^{30} \times 2 \times 3^2 \sim 2^{150}$  states

# Results

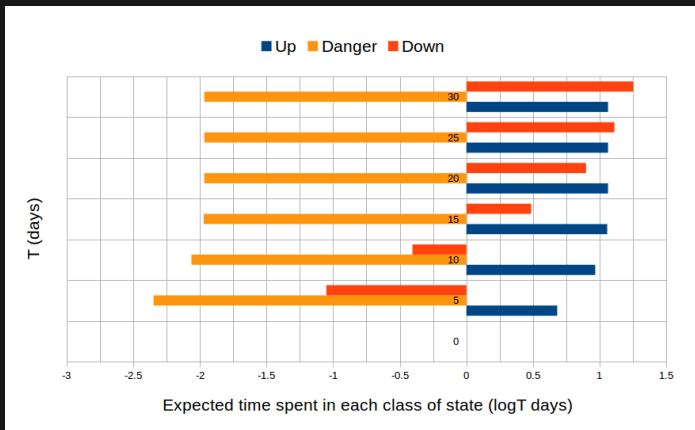
The probability that each of the 4 failure types is the cause of system shutdown in the first T time of operation

$$\text{shutdown} = \bigvee_{i=1}^4 \text{failure}_i \quad \neg \text{shutdown} \bigwedge_{i=1}^4 \neg \text{failure}_i$$



# Results

- The expected amount of time spent in each of the states: “up”, “danger” and “shutdown”
- $X_{\leq T} \text{reward\_c}$  returns the mean of **reward\_c** after **T** time of execution



# Tool in action

File selection:

Project: **ecs**  
Model: **ecs\_model**  
Requirements: **ev Actuators\_43200**  
**ev Actuators\_57600**  
**ev Actuators\_72000**  
**ev Actuators\_86400**  
Algorithm: **MonteCarli**  
Total samples: **1000**

Execution:

☐ distributed ☒ local  
Part: **0111**  
Threads: **1**  
Batch: **500**  
**Start**  
Algorithm completed - 0881.393s

Experimentation results plot

#	Name	# Simulations	# Positive Simulation	Result
68	261 <sub>ev_up_28800</sub>	1000	26156021	26156.021
69	320 <sub>ev_up_43200</sub>	1000	32053084	32053.084
70	323 <sub>ev_up_57600</sub>	1000	32349305	32349.305
71	323 <sub>ev_up_72000</sub>	1000	32349305	32349.305
72	323 <sub>ev_up_86400</sub>	1000	32349305	32349.305
73	13 <sub>ev_danger_14400</sub>	1000	13005	13.005
74	21 <sub>ev_danger_28800</sub>	1000	24503	24.503
75	30 <sub>ev_danger_43200</sub>	1000	30021	30.021
76	30 <sub>ev_danger_57600</sub>	1000	30313	30.313
77	30 <sub>ev_danger_72000</sub>	1000	30313	30.313
78	30 <sub>ev_danger_86400</sub>	1000	30313	30.313
79	301 <sub>ev_shutdown_14400</sub>	1000	301998	301.998
80	133 <sub>ev_shutdown_28800</sub>	1000	1339941	1339.941
81	917 <sub>ev_shutdown_43200</sub>	1000	9171632	9171.632
82	232 <sub>ev_shutdown_57600</sub>	1000	23249379	23249.379
83	376 <sub>ev_shutdown_72000</sub>	1000	37649379	37649.379
84	520 <sub>ev_shutdown_86400</sub>	1000	52049379	52049.379
85	46 <sub>ev_sensors_14400</sub>	1000	46757	46.757
86	40 <sub>ev_sensors_28800</sub>	1000	40166	40.166
87	32 <sub>ev_sensors_43200</sub>	1000	32829	32.829
88	25 <sub>ev_sensors_57600</sub>	1000	25982	25.982
89	20 <sub>ev_sensors_72000</sub>	1000	20145	20.145
90	15 <sub>ev_sensors_86400</sub>	1000	15336	15.336
91	29 <sub>ev Actuators_14400</sub>	1000	29820	29.82
92	29 <sub>ev Actuators_28800</sub>	1000	29316	29.316
93	28 <sub>ev Actuators_43200</sub>	1000	28560	28.56
94	27 <sub>ev Actuators_57600</sub>	1000	27619	27.619
95	26 <sub>ev Actuators_72000</sub>	1000	26581	26.581
96	25 <sub>ev Actuators_86400</sub>	1000	25444	25.444

ecs Simulation Experimentation Optimization

fr.inria.plasmalab.bri

1 : 191

<https://project.inria.fr/pscv/>



# Conclusion

- An introduction about **Statistical Model Checking**
- Some evidences that SMC scales to **large** systems
  - SystemC models
  - Simulink models [Clarke et al. 09, CMU]
- Initial experiments on SystemC for dependability analysis are carried out

## Future work

- More SystemC examples
- **Parallel implementation** of the statistical analyzer
- Consider the implementation of a **random scheduler** for SystemC kernel