Parameterized Verification with

Byzantine Model Checker (2)

Igor Konnov

`<igor@informal.systems>`

Tutorial at FORTE, June 15, 2020

streaming from Vienna / Austria to Valletta / Malta

informal

INTERCHAIN
FOUNDATION

Timeline

Fault-tolerant distributed algorithms and threshold automata

Safety of **asynchronous** threshold-guarded algorithms

Liveness and **beyond** asynchronous algorithms

The examples and links for this talk:

[bit.ly/2z8mE51]

Byzantine model checker:

[github.com/konnov/bymc]
[forsyte.at/software/bymc]

(source code, benchmarks, virtual machines, etc.)

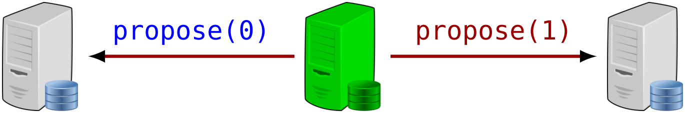
Verifying **asynchronous** threshold-guarded distributed algorithms

[K., Veith, Widder. CAV'15]
[K., Lazić, Veith, Widder. POPL'17]
[K., Lazić, Veith, Widder. FMSD'17]
[K., Widder. ISoLA'18]
...



Faults and communication

Byzantine behavior: [Lamport, Shostak, Pease, 1982]



More than two-thirds must be correct: $n > 3t$ (resilience)

Communication is **reliable**:

[Fischer, Lynch, Paterson, 1985]

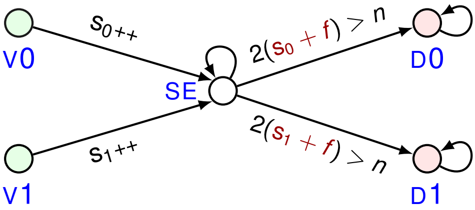
if a correct process sends a message *m*,

m is eventually delivered to all correct processes

Formalizing pseudo-code of naïve majority voting…

input $u_i \in \{0, 1\}$
send u_i **to** all
wait until some value $v_i \in \{0, 1\}$ is **received** $\lceil \frac{n+1}{2} \rceil$ times
decide **on** v_j

for **Byzantine** faults:



run $n - f$ copies for $n > 3t$ and $t \geq f$

Let’s run ByMC again

this time for Byzantine faults. . .

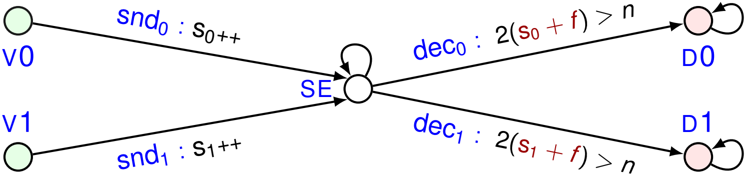
```
user@bymc: ~/fault-tolerant-benchmarks/forte20
user@bymc: ~/fault-tolerant-benchmarks/forte20 80x29
  --limit-time: limit (in seconds) cpu time of subprocesses (ulimit -t)
  --limit-mem: limit (in MB) virtual memory of subprocesses (ulimit -v)
  -h|--help: show this help message

 bymc_options are as follows:
  -O schema.tech=ltl          (default, safety + liveness as in POPL'17)
  -O schema.tech=ltl-mpi      (parallel safety + liveness as in ISOLA'18)
  -O schema.tech=cav15        (reachability as in CAV'15)
  --smt 'lib2|z3|-smt2|-in'   (default, use z3 as the backend solver)
  --smt 'lib2|mysolver|arg1|arg2|arg3'  (use an SMT2 solver)
  --smt 'yices'               (use yices 1.x as the backend solver, DEPRECATED)
  -v              (verbose output, all debug messages get printed)

 Fine tuning of schema.tech=ltl:
  -O schema.incremental=1 (enable the incremental solver, default: 0)

  -O schema.noflowopt=1  (disable the control flow optimizations, default: 0
                          may lead to a combinatorial explosion of guards)
  -O schema.noreachopt=1 (disable the reachability optimization, default: 0
                          i.e., reachability is not checked on-the-fly)
  -O schema.noadaptive=1 (disable the adaptive reachability optimization, defaul
t: 0
                          i.e., the tool will not try to choose between
                          enabling/disabling the reachability optimization)
  -O schema.noguardpreds=1 (do not introduce predicates for
                            the threshold guards, default: 0)
  -O schema.compute-nschemas=1 (always compute the total number of
                                schemas, even if takes long, default: 0)
user@bymc:~/fault-tolerant-benchmarks/forte20$
```

Counterexample to agreement



$\mathbf{f} = 1$ Byzantine, $\mathbf{n} - \mathbf{f} = 6$ correct

proc. 1	snd_0					dec$_0$		
proc. 2		snd_1						dec$_1$
proc. 3			snd_0					
proc. 4				snd_1				
proc. 5					snd_0			
proc. 6							snd_1	

Representative of the counterexample

$$\mathsf{snd}_0, \mathsf{snd}_1, \mathsf{snd}_0, \mathsf{snd}_1, \mathsf{snd}_0, \mathsf{dec}_0, \mathsf{snd}_1, \mathsf{dec}_1$$

becomes:

$$\mathsf{snd}_0, \mathsf{snd}_0, \mathsf{snd}_1, \mathsf{snd}_1, \underline{\mathsf{snd}_0}, \mathsf{dec}_0, \underline{\mathsf{snd}_1}, \mathsf{dec}_1$$

and this gives us a pattern (schema):

$$\mathsf{snd}_0^*, \mathsf{snd}_1^*, \underline{\mathsf{snd}_0}, \mathsf{snd}_0^*, \mathsf{snd}_1^*, \mathsf{dec}_0^*, \underline{\mathsf{snd}_1}, \mathsf{snd}_0^*, \mathsf{snd}_1^*, \mathsf{dec}_0^*, \mathsf{dec}_1^*$$

Execution patterns

one pattern

$$\mathrm{snd}_0^*, \mathrm{snd}_1^*, \underline{\mathrm{snd}_0}, \mathrm{snd}_0^*, \mathrm{snd}_1^*, \mathrm{dec}_0^*, \underline{\mathrm{snd}_1}, \mathrm{snd}_0^*, \mathrm{snd}_1^*, \mathrm{dec}_0^*, \mathrm{dec}_1^*$$

and another one:

$$\mathrm{snd}_0^*, \mathrm{snd}_1^*, \underline{\mathrm{snd}_1}, \mathrm{snd}_0^*, \mathrm{snd}_1^*, \mathrm{dec}_1^*, \underline{\mathrm{snd}_0}, \mathrm{snd}_0^*, \mathrm{snd}_1^*, \mathrm{dec}_0^*, \mathrm{dec}_1^*$$

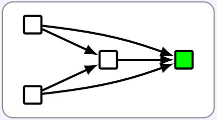
how many are there?

can we construct them?

do they work for all parameters?

The theoretical framework behind ByMC

Parameterized verification problem:

$\forall n, f.$ $n - f$ copies of  $\models \varphi$

Our approach:

(I) Counting processes,

(II) Acceleration,

(III) Bounded model checking, and

(IV) Schemas

(I) Counting processes

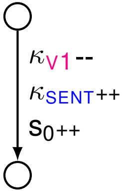
Threshold guards (e.g., $s_0 + s_1 + f \geq n - t$) do not use process ids

A transition by a single process:

$$\left\{\kappa_{V1} = 4 \wedge \kappa_{SENT} = 1 \wedge s_0 = 1\right\}$$

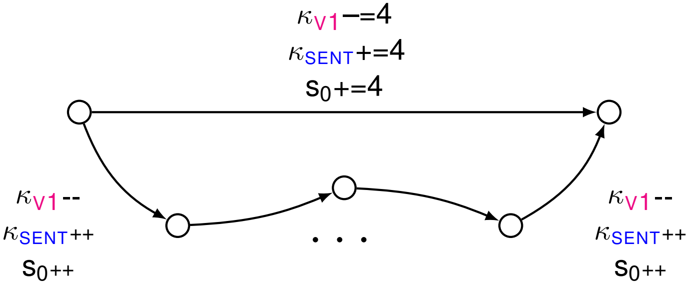
$$\kappa_{V1}\text{--}\ ;\ \kappa_{SENT}\text{++}\ ;\ s_0\text{++}\ ;$$

$$\left\{\kappa_{V1} = 3 \wedge \kappa_{SENT} = 2 \wedge s_0 = 2\right\}$$



(II) Acceleration

The same transition by unboundedly many processes in one step:

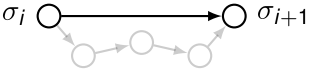


Acceleration factor can be any natural number δ

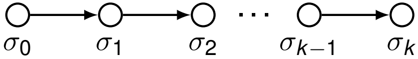
(III) Bounded model checking with SMT

A transition by δ_i processes (in linear integer arithmetic):

$$T(\sigma_i, \sigma_{i+1}, \delta_i) = \begin{bmatrix} \kappa_{\mathrm{V1}}^{i+1} = \kappa_{\mathrm{V1}}^{i} - \delta_i \wedge \\ \kappa_{\mathrm{SENT}}^{i+1} = \kappa_{\mathrm{SENT}}^{i} + \delta_i \wedge \\ \mathrm{s_0}^{i+1} = \mathrm{s_0}^{i} + \delta_i \end{bmatrix}$$



Execution:

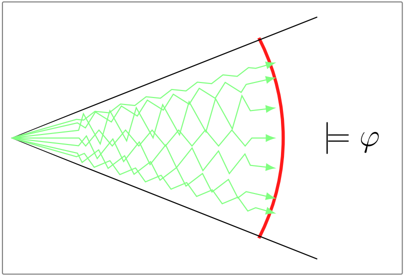


SMT formula: $T(\sigma_0, \sigma_1, \delta_0) \wedge T(\sigma_1, \sigma_2, \delta_1) \wedge \cdots \wedge T(\sigma_{k-1}, \sigma_k, \delta_{k-1}) \wedge \mathrm{Spec}$

how long should the executions be?

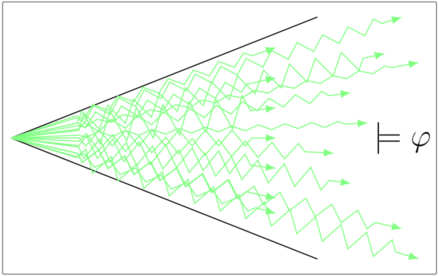
Completeness of bounded model checking

What we **can** do:



iff

What we **want** to do:

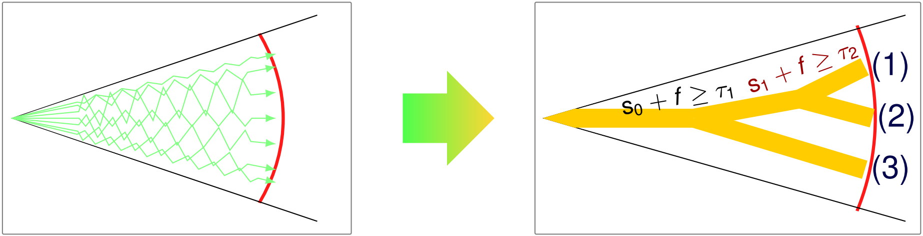


Complete and efficient BMC for:

- reachability [K., Veith, Widder: CAV'15]
- safety and liveness [K., Lazić, Veith, Widder: POPL'17]

Mover analysis

Exploring all bounded executions is inefficient



The argument contains:

- reordering: s_0++; s_1++; s_0++ becomes s_0++; s_0++; s_1++
- acceleration s_0++; s_0++; s_1++ becomes s_0 += 2; s_1++

(IV) Schemas — encoding representatives

Schema: $\{pre_1\}$ $actions_1$ $\{post_1\}$ $\dots$ $\{pre_k\}$ $actions_k$ $\{post_k\}$

Example:

$\{\}$ $(\mathsf{V0} \to \mathsf{SE0})^{\delta_1}$ $\{s_0 + f \geq \tau_{D0}\}$ $(\mathsf{V1} \to \mathsf{SE1})^{\delta_2}$ $\{\dots, s_1 + f \geq \tau_{D1}\}$
$(\mathsf{V0} \to \mathsf{SE0})^{\delta_3}, (\mathsf{V1} \to \mathsf{SE1})^{\delta_4}$ $\{\dots, \phi_A\}$ $(\mathsf{SE0} \to \mathsf{D0})^{\delta_5}, (\mathsf{SE1} \to \mathsf{D1})^{\delta_6}$
$\{\boldsymbol{\kappa}^6_{D0} \neq 0 \wedge \boldsymbol{\kappa}^6_{D1} \neq 0\}$

SMT solver tries to find: parameters n, t, f,
acceleration factors $\delta(1), \dots, \delta(6)$,
counters $\boldsymbol{\kappa}^i_{D0}, \boldsymbol{\kappa}^i_{D1}, \dots$

(a) the schema does not violate the property (**UNSAT**), or
(b) there is a counterexample (**SAT**)

(IV) Schemas — encoding representatives

Schema: $\{pre_1\}$ $actions_1$ $\{post_1\}$ ... $\{pre_k\}$ $actions_k$ $\{post_k\}$

Example:

$\{\}$ $(\mathsf{V0} \to \mathsf{SE0})^{\delta_1}$ $\{s_0 + f \geq \tau_{D0}\}$ $(\mathsf{V1} \to \mathsf{SE1})^{\delta_2}$ $\{\ldots, s_1 + f \geq \tau_{D1}\}$

$(\mathsf{V0} \to \mathsf{SE0})^{\delta_3}, (\mathsf{V1} \to \mathsf{SE1})^{\delta_4}$ $\{\ldots, \phi_A\}$ $(\mathsf{SE0} \to \mathsf{D0})^{\delta_5}, (\mathsf{SE1} \to \mathsf{D1})^{\delta_6}$

$\{\kappa^6_{D0} \neq 0 \wedge \kappa^6_{D1} \neq 0\}$

SMT solver tries to find: parameters n, t, f,
acceleration factors $\delta(1), \ldots, \delta(6)$,
counters $\kappa^i_{D0}, \kappa^i_{D1}, \ldots$

(a) the schema does not violate the property (**UNSAT**), or
(b) there is a counterexample (**SAT**)

(IV) Schemas — encoding representatives

Schema: $\{pre_1\}$ $actions_1$ $\{post_1\}$ $\ldots$ $\{pre_k\}$ $actions_k$ $\{post_k\}$

Example:

$\{\}$ $(\mathsf{V0} \to \mathsf{SE0})^{\delta_1}$ $\{s_0 + f \geq \tau_{\mathsf{D0}}\}$ $(\mathsf{V1} \to \mathsf{SE1})^{\delta_2}$ $\{\ldots, s_1 + f \geq \tau_{\mathsf{D1}}\}$
$(\mathsf{V0} \to \mathsf{SE0})^{\delta_3}, (\mathsf{V1} \to \mathsf{SE1})^{\delta_4}$ $\{\ldots, \phi_{\mathsf{A}}\}$ $(\mathsf{SE0} \to \mathsf{D0})^{\delta_5}, (\mathsf{SE1} \to \mathsf{D1})^{\delta_6}$
$\{\boldsymbol{\kappa}^6_{\mathsf{D0}} \neq 0 \wedge \boldsymbol{\kappa}^6_{\mathsf{D1}} \neq 0\}$

SMT solver tries to find: parameters n, t, f,
acceleration factors $\delta(1), \ldots, \delta(6)$,
counters $\boldsymbol{\kappa}^i_{\mathsf{D0}}, \boldsymbol{\kappa}^i_{\mathsf{D1}}, \ldots$

(a) the schema does not violate the property (**UNSAT**), or
(b) there is a counterexample (**SAT**)

(IV) Schemas — encoding representatives

Schema: $\{pre_1\}$ $actions_1$ $\{post_1\}$ $\ldots$ $\{pre_k\}$ $actions_k$ $\{post_k\}$

Example:

$\{\}$ $(\mathsf{V0} \to \mathsf{SE0})^{\delta_1}$ $\{s_0 + f \geq \tau_{\mathsf{D0}}\}$ $(\mathsf{V1} \to \mathsf{SE1})^{\delta_2}$ $\{\ldots, s_1 + f \geq \tau_{\mathsf{D1}}\}$
$(\mathsf{V0} \to \mathsf{SE0})^{\delta_3}$, $(\mathsf{V1} \to \mathsf{SE1})^{\delta_4}$ $\{\ldots, \phi_{\mathsf{A}}\}$ $(\mathsf{SE0} \to \mathsf{D0})^{\delta_5}$, $(\mathsf{SE1} \to \mathsf{D1})^{\delta_6}$

$\{\boldsymbol{\kappa}^6_{\mathsf{D0}} \neq 0 \wedge \boldsymbol{\kappa}^6_{\mathsf{D1}} \neq 0\}$

SMT solver tries to find: parameters n, t, f,
acceleration factors $\delta(1), \ldots, \delta(6)$,
counters $\boldsymbol{\kappa}^i_{\mathsf{D0}}, \boldsymbol{\kappa}^i_{\mathsf{D1}}, \ldots$

(a) the schema does not violate the property (**UNSAT**), or
(b) there is a counterexample (**SAT**)

(IV) Schemas — encoding representatives

Schema: $\{pre_1\}$ $actions_1$ $\{post_1\}$ ... $\{pre_k\}$ $actions_k$ $\{post_k\}$

Example:

$\{\}$ $(\mathsf{V0} \to \mathsf{SE0})^{\delta_1}$ $\{s_0 + f \geq \tau_{D0}\}$ $(\mathsf{V1} \to \mathsf{SE1})^{\delta_2}$ $\{\dots, s_1 + f \geq \tau_{D1}\}$
$(\mathsf{V0} \to \mathsf{SE0})^{\delta_3}$, $(\mathsf{V1} \to \mathsf{SE1})^{\delta_4}$ $\{\dots, \phi_A\}$ $(\mathsf{SE0} \to \mathsf{D0})^{\delta_5}$, $(\mathsf{SE1} \to \mathsf{D1})^{\delta_6}$
$\{\boldsymbol{\kappa}^6_{D0} \neq 0 \wedge \boldsymbol{\kappa}^6_{D1} \neq 0\}$

SMT solver tries to find: parameters n, t, f,
acceleration factors $\delta(1), \dots, \delta(6)$,
counters $\boldsymbol{\kappa}^i_{D0}, \boldsymbol{\kappa}^i_{D1}, \dots$

(a) the schema does not violate the property (**UNSAT**), or
(b) there is a counterexample (**SAT**)

Overview of the verification algorithm

Threshold automaton $\longrightarrow$ schemas $\{S_1, \dots, S_k\}$

$Z3 \models S_1$

sat

$Z3 \models S_2$

counterexample

$\dots$

$Z3 \models S_k$

unsat?

Overview of the verification algorithm

Threshold automaton $\longrightarrow$ schemas $\{S_1, \ldots, S_k\}$

$Z3 \models S_1$

$Z3 \models S_2$

$\ldots$

$Z3 \models S_k$

sat

counterexample

unsat?

Threshold automaton $\longrightarrow$ schemas $\{S_1, \ldots, S_k\}$

$Z3 \models S_1$

$Z3 \models S_2$

$\ldots$

$Z3 \models S_k$

sat

counterexample

unsat?

©VSC / Claudia Blaas-Schenner

Vienna Scientific Cluster

GRID 5000 (France)

Time for questions!

Threshold automata to model asynchronous algorithms

Bounded model checking of counter systems

Completeness due to the bounds

(liveness and general safety in part 3)