Parameterized Verification with

Byzantine Model Checker (3)

Igor Konnov

<igor@informal.systems>

Tutorial at FORTE, June 15, 2020

streaming from Vienna / Austria to Valletta / Malta

informal

INTERCHAIN
FOUNDATION

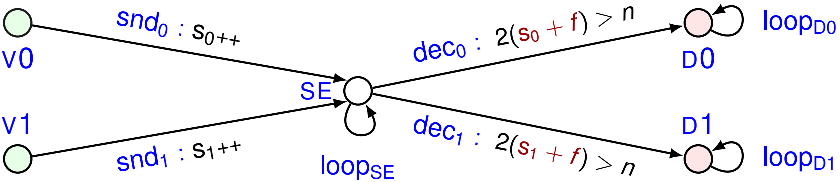
Timeline

Fault-tolerant distributed algorithms and threshold automata

Safety of **asynchronous** threshold-guarded algorithms

Liveness and **beyond** asynchronous algorithms

$\mathbf{n} = 7$ processes: $\mathbf{f} = 2$ Byzantine, $\mathbf{n} - \mathbf{f} = 5$ correct



$$\textbf{termination:} \quad \Diamond\Box\, \mathit{fair} \rightarrow \Diamond\,(\kappa_{V0} = 0 \wedge \kappa_{V1} = 0 \wedge \kappa_{SE} = 0)$$

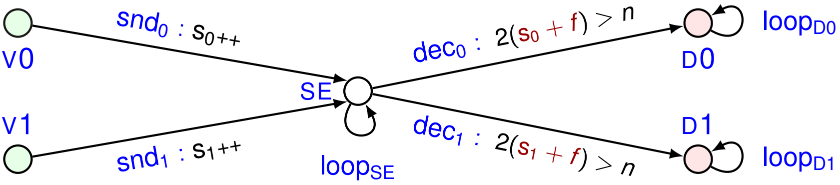
$$\neg\textbf{termination:} \quad \Diamond\Box\, \mathit{fair} \wedge \Box\,(\kappa_{V0} \neq 0 \vee \kappa_{V1} \neq 0 \vee \kappa_{SE} \neq 0)$$

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

$$(\underbrace{\mathsf{loop}_{SE}, \mathsf{loop}_{D0}, \mathsf{loop}_{D1}}_{\Box\,\mathit{fair} \wedge \Box\,(\kappa_{V0} \neq 0 \vee \kappa_{V1} \neq 0 \vee \kappa_{SE} \neq 0)})^{\omega}$$

Naïve voting and termination

$\mathbf{n} = 7$ processes: $\mathbf{f} = 2$ Byzantine, $\mathbf{n} - \mathbf{f} = 5$ correct



$$\textbf{termination:}\quad \Diamond\Box\, fair \rightarrow \Diamond\,(\boldsymbol{\kappa}_{V0} = 0 \wedge \boldsymbol{\kappa}_{V1} = 0 \wedge \boldsymbol{\kappa}_{SE} = 0)$$

$$\neg\textbf{termination:}\quad \Diamond\Box\, fair \wedge \Box\,(\boldsymbol{\kappa}_{V0} \neq 0 \vee \boldsymbol{\kappa}_{V1} \neq 0 \vee \boldsymbol{\kappa}_{SE} \neq 0)$$

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

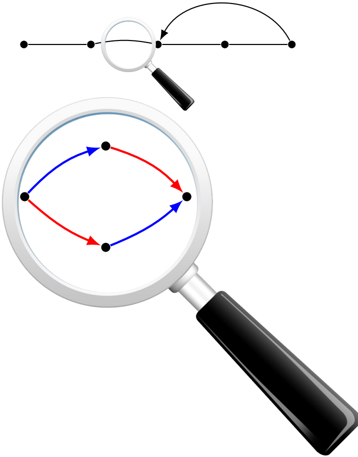
$$(\underbrace{\mathrm{loop}_{SE}, \mathrm{loop}_{D0}, \mathrm{loop}_{D1}}_{\Box\, fair \wedge \Box\,(\boldsymbol{\kappa}_{V0} \neq 0 \vee \boldsymbol{\kappa}_{V1} \neq 0 \vee \boldsymbol{\kappa}_{SE} \neq 0)})^{\omega}$$

From reachability to safety & liveness

A) A temporal logic for bad executions

$$\mathbf{E}\left(\varphi_1 \wedge \Diamond \Box (\varphi_2 \vee \varphi_3)\right)$$

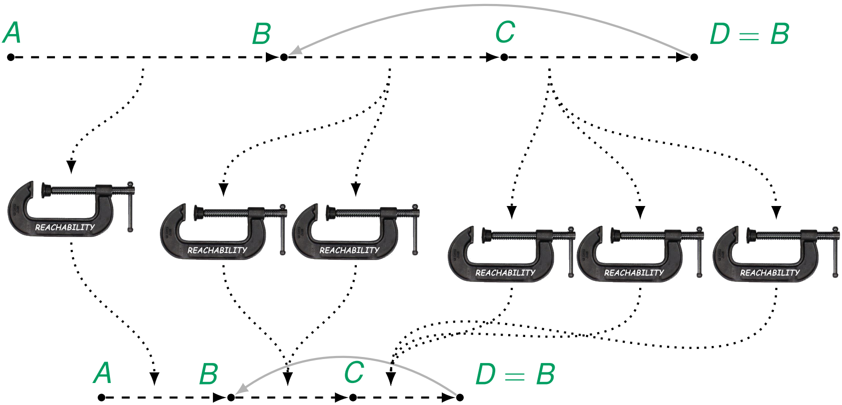
B) Enumerating shapes of counterexamples



C) Property specific mover analysis

Details in [K., Lazić, Veith, Widder. POPL'17]

Short counterexamples for safety or liveness



Safety & liveness (POPL'17)

Every lasso can be transformed into a bounded one. The bound depends on the process code and the specification, not the parameters.

The shape of temporal formulas

Termination: every process eventually decides

$$\Diamond \Box \, \psi_{\mathsf{fair}} \longrightarrow \Diamond \left(\boldsymbol{\kappa}_{\mathsf{V1}} = 0 \lor \boldsymbol{\kappa}_{\mathsf{V0}} = 0 \lor \boldsymbol{\kappa}_{\mathsf{SE}} = 0 \right)$$

The shape of temporal formulas

Termination: every process eventually decides

$$\Diamond \Box \boxed{\psi_{\text{fair}}} \longrightarrow \Diamond \left(\boxed{\boldsymbol{\kappa}_{\text{V1}} = 0 \vee \boldsymbol{\kappa}_{\text{V0}} = 0 \vee \boldsymbol{\kappa}_{\text{SE}} = 0} \right)$$

Propositional formulas:

(1) $\bigwedge_{\ell \in S} \boldsymbol{\kappa}_\ell = 0$

(2) $\bigvee_{\ell \in S} \boldsymbol{\kappa}_\ell \neq 0$

(3) $\bigvee_{S \subseteq T} \bigwedge_{\ell \in S} \boldsymbol{\kappa}_\ell = 0$

(4) $\text{Bool(Guards)} \rightarrow (1) \wedge (2) \wedge (3)$

Temporal formulas:

$\psi ::= \textit{prop} \mid \Box\, \psi \mid \Diamond\, \psi \mid \psi \vee \psi$

Warning about formulas

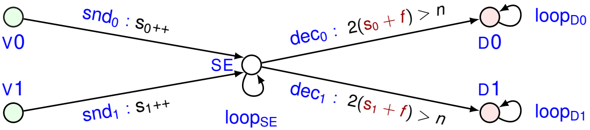
[POPL'17] defines the logic $\mathrm{ELTL}_{\mathrm{FT}}$ for counterexamples

$\mathrm{ELTL}_{\mathrm{FT}}$ talks about one execution (the shape of a counterexample)

ByMC uses the logic LTL for all executions

That is, ByMC accepts $\neg\varphi$ for $\varphi \in \mathrm{ELTL}_{\mathrm{FT}}$

The hard part: fairness



All correct processes take infinitely many steps:

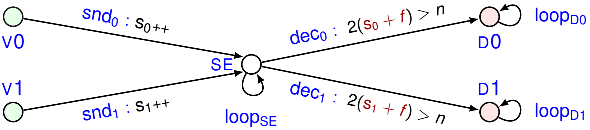
$$\Diamond \Box \left(\kappa_{\mathrm{V0}} = 0 \land \kappa_{\mathrm{V1}} = 0\right)$$

Every message sent by a correct process is...

eventually received by all correct processes:

$$\Diamond \Box \left(\left(\underbrace{2s_0 \le n}_{\neg\mathrm{ENABLED}(\mathrm{dec}_0)} \lor \kappa_{\mathrm{SE}} = 0 \right) \land /^{*}\ \mathrm{dec}_1 \ldots\ ^{*}/ \right)$$

The hard part: fairness



All correct processes take infinitely many steps:

$$\Diamond \Box (\kappa_{V0} = 0 \land \kappa_{V1} = 0)$$

Every message sent by a correct process is...

eventually received by all correct processes:

$$\Diamond \Box \left((\underbrace{2s_0 \leq n}_{\neg \mathrm{ENABLED}(\mathrm{dec}_0)} \lor \kappa_{SE} = 0) \land /^* \mathrm{dec}_1 \ldots\ ^*/ \right)$$

More complex algorithm: BOSCO

One-step Byzantine asynchronous consensus

every process starts with a value $v_i \in \{0, 1\}$

agreement: no two processes decide differently

validity: if a correct process decides on v,
then v was the initial value of at least one process

unanimity: if all correct processes are initialized with v,
every deciding correct process must decide on v

termination: all correct processes eventually decide

---

decide in one communication step,

when there are “not too many faults”

One-step Byzantine asynchronous consensus

every process starts with a value $v_i \in \{0, 1\}$

agreement: no two processes decide differently

validity: if a correct process decides on v,
then v was the initial value of at least one process

unanimity: if all correct processes are initialized with v,
every deciding correct process must decide on v

termination: all correct processes eventually decide

---

decide in one communication step,

when there are "not too many faults"

---

input v_p
send $\langle \texttt{VOTE}, v_p \rangle$ **to** all processors;

wait until $n - t$ VOTE *messages have been received*;

if *more than* $\frac{n+3t}{2}$ VOTE *messages contain the same value* v
then DECIDE(v);

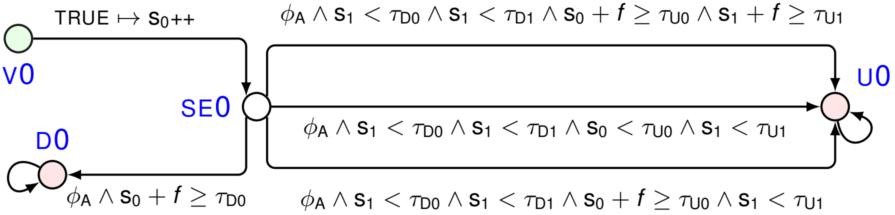
if *more than* $\frac{n-t}{2}$ VOTE *messages contain the same value* v,
and there is only one such value v
then $v_p \leftarrow v$;

call Underlying-Consensus(v_p);

resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

fast termination: when $n > 5t$ and $f = 0$ and $n > 7t$

Threshold automaton



(similar for V1, SE1, D1, U1, ...)

threshold guards, e.g., ϕ_A is defined as $s_0 + s_1 + f \geq n - t$

increments of shared variables, e.g., s_0++

run $n - f$ copies provided that there are $f \leq t$ Byzantine faults and $n > 3t$

Let’s run ByMC on BOSCO. . .

```
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$
```

Performance tuning

Incremental vs. offline SMT: `-O schema.incremental=(0|1)`

Reachability optimization: `-O schema.noreachopt=(0|1)`

Dependencies between the guards: `-O schema.noflowopt=(0|1)`

e.g., $x \geq t + 1$ precedes $x \geq 2t + 1$

Liveness vs. parallel liveness vs. reachability:

`-O schema.tech=(ltl|ltl-mpi|cav15)`

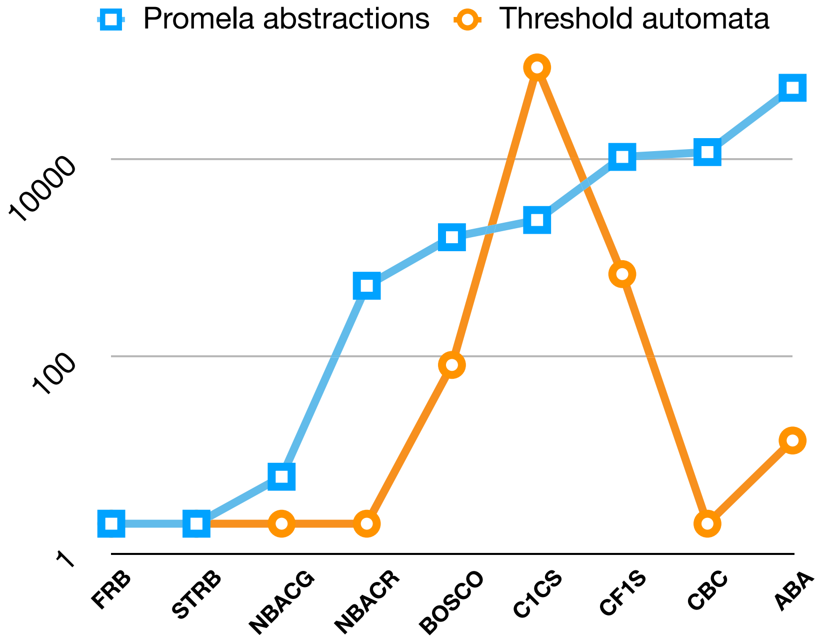
More algorithms

More threshold guards...

Reliable broadcast	$x \geq t + 1$ $x \geq n - t$	[Srikanth, Toueg'86]
Hybrid broadcast	$x \geq t_b + 1$ $x \geq n - t_b - t_c$	[Widder, Schmid'07]
Byzantine agreement	$x \geq \lceil \frac{n}{2} \rceil + 1$	[Bracha, Toueg'85]
Non-blocking atomic commitment	$x \geq n$	[Raynal'97], [Guerraoui'01]
Condition-based consensus	$x \geq n - t$ $x \geq \lceil \frac{n}{2} \rceil + 1$	[Mostéfaoui, Mourgaya, Parvedy, Raynal'03]
Consensus in one communication step	$x \geq n - t$ $x \geq n - 2t$	[Brasileiro, Greve, Mostéfaoui, Raynal'03]
Byzantine one-step consensus	$x \geq \lceil \frac{n+3t}{2} \rceil + 1$	[Song, van Renesse'08]

In general, there is a resilience condition, e.g., $n > 3t$, $n > 7t$

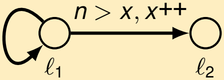
Time to check the algorithms [ISOLA'18]



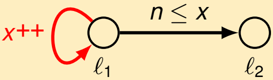
Beyond asynchrony and threshold automata

Extending threshold automata

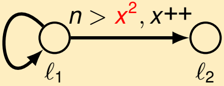
standard TA



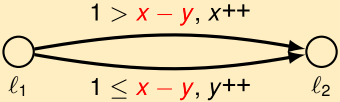
increments in loops (NCTA)



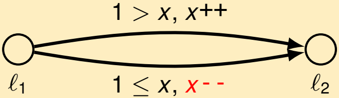
piecewise monotone (PMTA)



bounded difference (BDTA)



reversible (RTA)



reversal bounded (RBTA)

Like reversible automata, but increments and decrements of variables may alternate a bounded number of times.

All flavors of threshold automata [CONCUR'18]

Level	Reversals	Canonical	Bounded diameter	Flattable	Decidable reachability	Fragment
x	0	✓	✓	✓	✓	TA
p.m. $f(x)$	0	✓	✓	✓	✓	PMTA
x	$\leq k$	✓	✓	✓	✓	RBTA
x	0	✗	✗	✓	✓	NCTA
$x - y$	0	✓	✗	✗	✗	BDTA
x	∞	✓	✗	✗	✗	RTA

Jure Kukovec

I.K.

Josef Widder

Probabilistic threshold-guarded algorithms [CONCUR'19]

No consensus algorithm for asynchronous systems (FLP'85)

Coin toss to break ties: $\mathit{value} := \mathit{random}(\{0, 1\})$

Ben-Or's, Bracha's consensus, RS-Bosco, *k*-set agreement

Compositional reasoning and reduction for multiple rounds

ByMC to reason about a single round

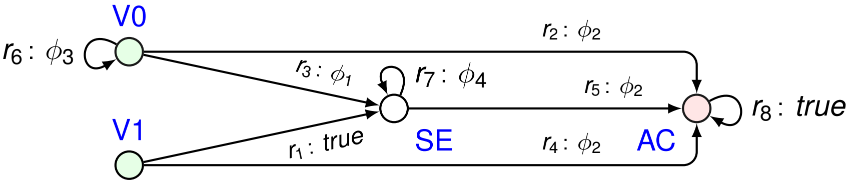
Nathalie Bertrand

I.K.

Marijana Lazić

Josef Widder

Synchronous threshold-guarded algorithms



All processes move in lockstep

Counting processes in local states, not the sent messages, e.g.:

ϕ_1 is $\#\{V1, SE, AC\} \geq t + 1 - f$
ϕ_2 is $\#\{V1, SE, AC\} \geq n - t - f$

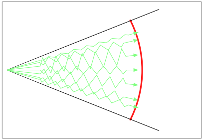
Synchronous threshold automata

Reachability for synchronous TA [TACAS'19]

In general, even reachability is undecidable!

Bounded diameter for trapped synchronous TA

A procedure for finding diameters with SMT



Reliable broadcast, phase king/queen, *k*-set agreement, FloodSet

tiny diameters from **1** to **4**

Ilina Stoilkovska

I.K.

Josef Widder

Florian Zuleger

Industrial distributed algorithms in Tendermint blockchain

I read that paper about **Byzantine Model Checker**

Model the algorithm as a threshold automaton

Verify safety and liveness for all $n, t, f : n > 3t \wedge t \geq f \geq 0$

I have heard this talk by Leslie Lamport

Let's write it in TLA^{+}

Run the **TLC model checker** for fixed parameters

TLC takes forever...

Run **APALACHE** for fixed parameters

Industrial distributed algorithms in Tendermint blockchain

I read that paper about **Byzantine Model Checker**

Model the algorithm as a threshold automaton

Verify safety and liveness for all $n, t, f : n > 3t \wedge t \geq f \geq 0$

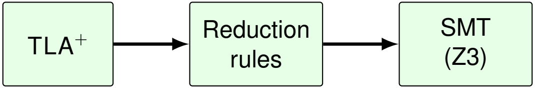
I have heard this talk by Leslie Lamport

Let's write it in TLA$^+$

Run the **TLC model checker** for fixed parameters

TLC takes forever...

Run **APALACHE** for fixed parameters



Focus on distributed algorithms

✓ Invariants	+ Fixed parameters, bounded executions
✓ Inductive invariants	+ Fixed parameters

[github.com/konnov/apalache]

What we were doing in the last months...

Specifying several Tendermint protocols in TLA^{+}:

- fast synchronization
- light client
- consensus, tuned for fork detection

`[github.com/informalsystems/verification]`

Conclusions

Reasoning about fault-tolerant algorithms is hard

…but fun!

Practical algorithms are even harder

Threshold guards are everywhere

Specialized tools for narrow classes, e.g., ByMC

vs.

General tools for broader classes, e.g., Apalache