Distributed algorithms: a challenging playground for model checking

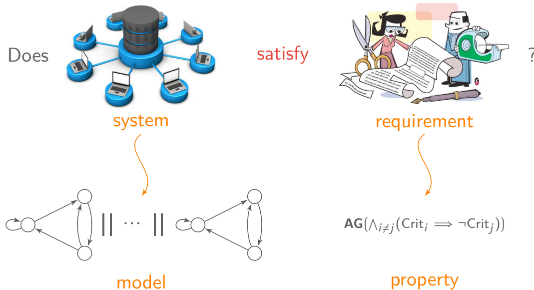
Nathalie Bertrand Inria

OPODIS - December 13th 2021

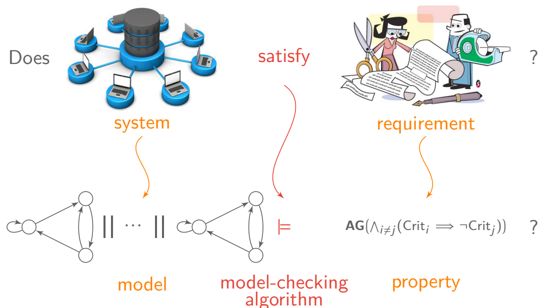
based on joint work with Igor Konnov, Marijana Lazić, Bastien Thomas and Josef Widder

Model checking in a nutshell

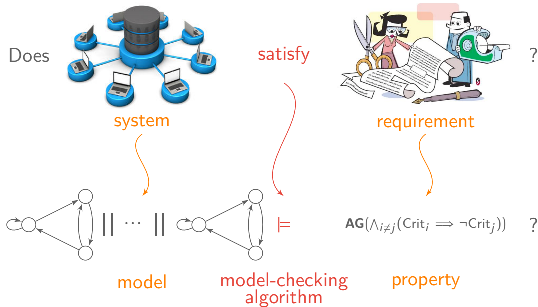
Model checking in a nutshell



Model checking in a nutshell



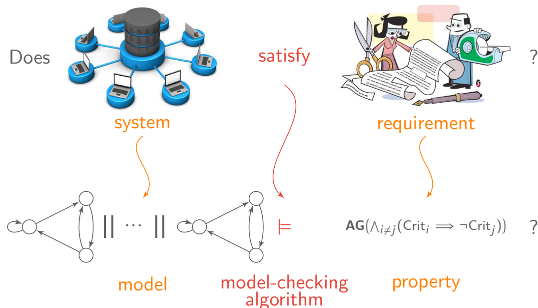
Model checking in a nutshell



⊕ generic, successfully applied to hardware/software verification
embedded softwares, real-time systems, controllers in avionics, telecommunications, planning, etc.

⊖ undecidable in general, scalability issues

Model checking in a nutshell



⊕ generic, successfully applied to hardware/software verification
embedded softwares, real-time systems, controllers in avionics, telecommunications, planning, etc.

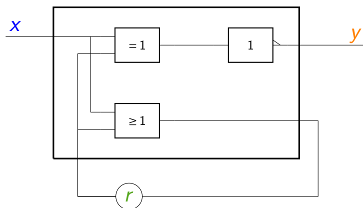
⊖ undecidable in general, scalability issues

2 Turing awards

- Pnueli, 1996: temporal logic; program and systems verification
- Clarke, Emerson and Sifakis, 2007: model checking as highly effective verification technology

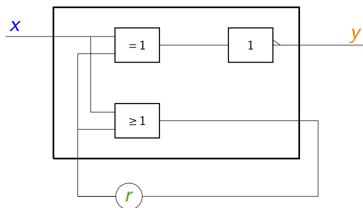
Hardware model checking: a basic example

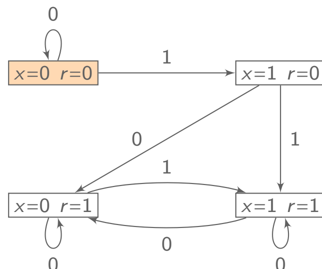
output control function $\lambda_y = \neg(x \oplus r)$
register evaluation $\delta_r = x \vee r$



Hardware model checking: a basic example

output control function $\lambda_y = \neg(x \oplus r)$
register evaluation $\delta_r = x \vee r$

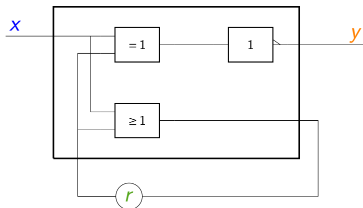


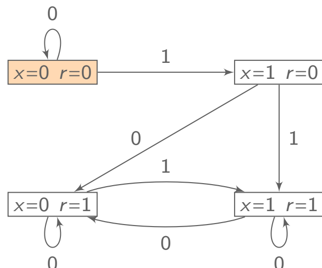


automaton model

Hardware model checking: a basic example

output control function $\lambda_y = \neg(x \oplus r)$
register evaluation $\delta_r = x \vee r$



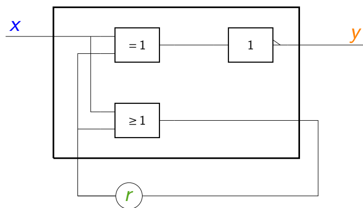


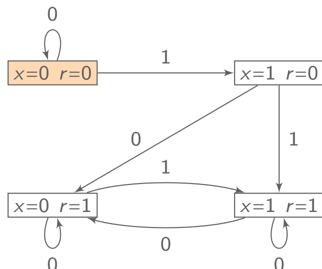
automaton model

safety property: when register value is 1, output is 0
reduces to no state with $r = 1$ and $y = 1$ is reachable
→ counterexample: finite path reaching error states

Hardware model checking: a basic example

output control function $\lambda_y = \neg(x \oplus r)$
register evaluation $\delta_r = x \vee r$





automaton model

safety property: when register value is 1, output is 0
reduces to no state with $r = 1$ and $y = 1$ is reachable
→ counterexample: finite path reaching error states

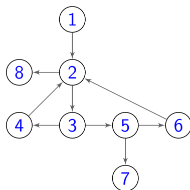
automaton is exponential in number of variables

Software model checking: a basic example

```
0  binsearch(x,t) /* t sorted int array of size n */
1  low:=0; high:=n-1;
2  while low <= high {mid := (low + high)/2;
3    if x < t[mid]
4       then high:=mid-1;
5       else if x > t[mid]
6            then low:=mid+1;
7            else return mid;}
8  return -1;
```

Software model checking: a basic example

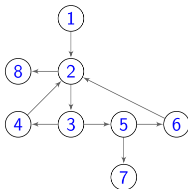
```
0  binsearch(x,t) /* t sorted int array of size n */
1  low:=0; high:=n−1;
2  while low <= high {mid := (low + high)/2;
3    if x < t[mid]
4        then high:=mid−1;
5        else if x > t[mid]
6             then low:=mid+1;
7             else return mid;}
8  return −1;
```



control flow graph

Software model checking: a basic example

```
0 binsearch(x,t) /* t sorted int array of size n */
1 low:=0; high:=n−1;
2 while low <= high {mid := (low + high)/2;
3   if x < t[mid]
4      then high:=mid−1;
5      else if x > t[mid]
6            then low:=mid+1;
7            else return mid;}
8 return −1;
```



control flow graph

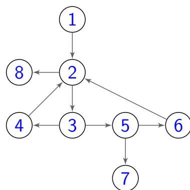
liveness property: every execution terminates

reduces to maximal paths in CFG reach 7 or 8

→ CFG model is not sufficient to prove termination

Software model checking: a basic example

```
0 binsearch(x,t) /* t sorted int array of size n */
1 low:=0; high:=n-1;
2 while low <= high {mid := (low + high)/2;
3   if x < t[mid]
4      then high:=mid-1;
5      else if x > t[mid]
6           then low:=mid+1;
7           else return mid;}
8 return -1;
```



control flow graph

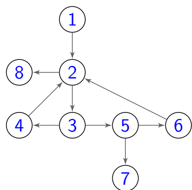
liveness property: every execution terminates
reduces to maximal paths in CFG reach 7 or 8
→ CFG model is not sufficient to prove termination

examples of richer model classes

- ⊕ counters: `int` manipulation
- ⊕ FIFO queues: communication channels
- ⊕ pushdown automata: recursion stack

Software model checking: a basic example

```
0 binsearch(x,t) /* t sorted int array of size n */
1 low:=0; high:=n-1;
2 while low <= high {mid := (low + high)/2;
3   if x < t[mid]
4      then high:=mid-1;
5      else if x > t[mid]
6           then low:=mid+1;
7           else return mid;}
8 return -1;
```



control flow graph

liveness property: every execution terminates
reduces to maximal paths in CFG reach 7 or 8

→ CFG model is not sufficient to prove termination

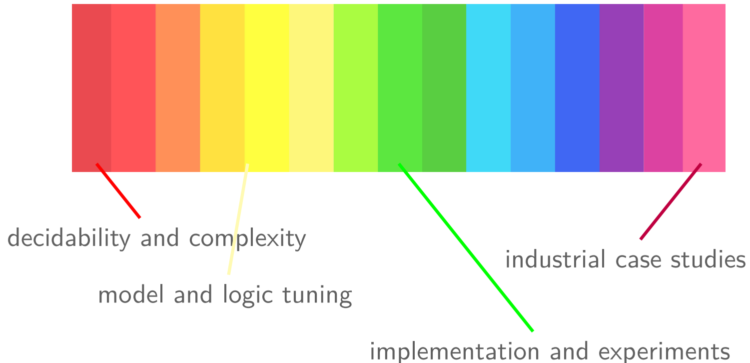
examples of richer model classes

- ⊕ counters: `int` manipulation
- ⊕ FIFO queues: communication channels
- ⊕ pushdown automata: recursion stack

the more expressive the model, the more complex the verification

Model checking community

Wide spectrum of activities: from theory to applications



Outline

Introduction to model checking

Parameterized verification for distributed algorithms

Two frameworks for threshold-based fault-tolerant algorithms
- Threshold automata (with random choices)
- Predicate abstraction

Conclusion

Peterson's mutual exclusion algorithm

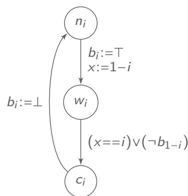
[Peterson Information Processing Letters 1981]

```
loop forever;
.
.   /* non-critical actions */
.
b_i := ⊤ ; x := 1 − i;
wait until (x = i) ∨ (¬b_{1−i}); /* request */
do critical section od;
b_i = ⊥; /* release */
.
.
.
end loop
```

Peterson's mutual exclusion algorithm

[Peterson Information Processing Letters 1981]

```
loop forever;
.
.   /* non-critical actions */
.
b_i := ⊤ ; x := 1 - i;
wait until (x = i) ∨ (¬b_{1-i}); /* request */
do critical section od;
b_i = ⊥; /* release */
.
.
.
end loop
```



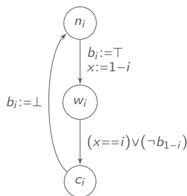
Correctness expressed as a **safety** property:

the processes are not in their critical section simultaneously

Peterson's mutual exclusion algorithm

[Peterson Information Processing Letters 1981]

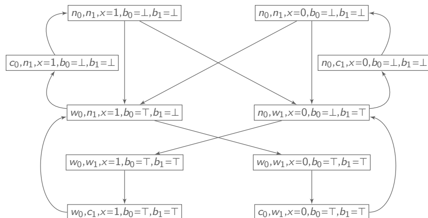
```
loop forever;
.
.   /* non-critical actions */
.
b_i := ⊤ ; x := 1 - i;
wait until (x = i) ∨ (¬b_{1-i}); /* request */
do critical section od;
b_i = ⊥; /* release */
.
.
.
end loop
```



Correctness expressed as a **safety** property:

the processes are not in their critical section simultaneously

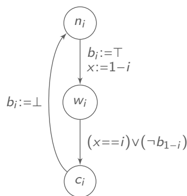
Product transition system representing all possible interleavings



Peterson's mutual exclusion algorithm

[Peterson Information Processing Letters 1981]

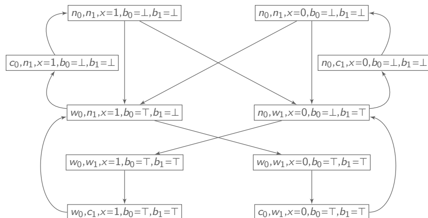
```
loop forever;
.
.    /* non-critical actions */
.
b_i := ⊤ ; x := 1 - i;
wait until (x = i) ∨ (¬b_{1-i}); /* request */
do critical section od;
b_i = ⊥; /* release */
.
.
.
end loop
```



Correctness expressed as a **safety** property:

the processes are not in their critical section simultaneously

Product transition system representing all possible interleavings



checking correctness reduces to

no state $(c_0, c_1, _, _, _)$ is reachable

→ exhaustive exploration of executions

Limitations of standard model checking techniques

⚠ **state-space explosion**: product transition system is exponential in number of processes, and of variables
→ tools hardly scale to large number of processes or real-life examples

Limitations of standard model checking techniques

⚠ **state-space explosion**: product transition system is exponential in number of processes, and of variables

→ tools hardly scale to large number of processes or real-life examples

partial solutions to improve scalability

- BDD encodings
- POR techniques
- bounded model-checking
- CEGAR approaches

Limitations of standard model checking techniques

⚠ **state-space explosion**: product transition system is exponential in number of processes, and of variables

→ tools hardly scale to large number of processes or real-life examples

partial solutions to improve scalability

- BDD encodings
- POR techniques
- bounded model-checking
- CEGAR approaches

⚠ models with **fixed number of processes**

e.g. randomized consensus for 10 participants with model checker PRISM

→ correctness should be proven for arbitrary number of processes

Parameterized verification: to infinity and beyond!

Parameterized verification: to infinity and beyond!

- correctness should hold **for every number of clients**

$$\forall n \quad \underbrace{C \,||\cdots||\, C}_{n \text{ times}} \,||\, S \quad \models \quad \varphi$$

- more generally: for all number of participants, for all network topologies, for all potential failures, for all *parameter valuations*

Parameterized verification: to infinity and beyond!

- correctness should hold **for every number of clients**

$$\forall n \quad \underbrace{C \mid\mid \cdots \mid\mid C}_{n \text{ times}} \mid\mid S \quad \models \quad \varphi$$

- more generally: for all number of participants, for all network topologies, for all potential failures, for all *parameter valuations*

model checking infinitely many instances at once

Parameterized verification: to infinity and beyond!

- correctness should hold **for every number of clients**

$$\forall n \quad \underbrace{C \parallel \cdots \parallel C}_{n \text{ times}} \parallel S \quad \models \quad \varphi$$

- more generally: for all number of participants, for all network topologies, for all potential failures, for all *parameter valuations*

⚠ model checking infinitely many instances at once

Good news! it may be computationally easier to prove correctness "for all n" than "for n a large fixed value".

Parameterized verification for distributed algorithms

Starting from...

- algorithm pseudo-code

```
bool v := input_value({0, 1});
int  r := 1 ;
while (true) do
 send (R,r,v) to all;
 wait for n − t messages (R,r,∗);
 if received (n + t) / 2 messages (R,r,w)
 then send (P,r,w,D) to all;
 else send (P,r,?) to all;
 wait for n − t messages (P,r,∗);
 if received at least t + 1
    messages (P,r,w,D) then {
  v := w;
/* enough support −> update estimate */
  if received at least (n + t) / 2
   messages (P,r,w,D)
  then decide w;
/* strong majority −> decide */
 } else  v := random(0, 1) ;
/* unclear −> coin toss      */
 r := r + 1 ;
od
```

- requirements
  validity, agreement
  (a.s.) termination

Parameterized verification for distributed algorithms

Starting from...

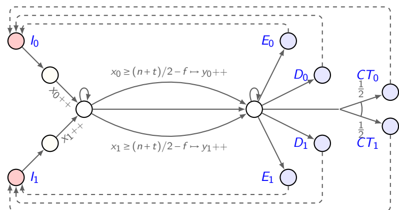
- algorithm pseudo-code

```
bool v := input_value({0, 1});
int r := 1 ;
while (true) do
 send (R,r,v) to all;
 wait for n − t messages (R,r,∗);
 if received (n + t) / 2 messages (R,r,w)
 then send (P,r,w,D) to all;
 else send (P,r,?) to all;
 wait for n − t messages (P,r,∗);
 if received at least t + 1
    messages (P,r,w,D) then {
  v := w;
/∗ enough support −> update estimate ∗/
  if received at least (n + t) / 2
  messages (P,r,w,D)
  then decide w;
/∗ strong majority −> decide ∗/
 } else v := random(0, 1) ;
/∗ unclear −> coin toss       ∗/
 r := r + 1 ;
od
```

- requirements
  validity, agreement
  (a.s.) termination

... derive

- model



- formulas

$\forall k, k' \in \mathbb{N}, \quad \mathbf{A}\left(\mathbf{F}\,\boldsymbol{\kappa}[D_V, k] > 0 \rightarrow \mathbf{G}\,\boldsymbol{\kappa}[D_{1-V}, k'] = 0\right)$

$\forall k \in \mathbb{N}\, \mathbf{A}\left(\mathbf{F}\,\boldsymbol{\kappa}[I_V, 0] = 0 \quad \rightarrow \quad \mathbf{G}\,\boldsymbol{\kappa}[D_V, k] = 0\right)$

$\mathbb{P}_{\mathfrak{a}}\left(\bigvee_{k \in \mathbb{N}} \bigvee_{v \in \{0,1\}} \mathbf{G} \bigwedge_{\ell \in \mathcal{L} \setminus \{D_V\}} \boldsymbol{\kappa}[\ell, k] = 0\right) = 1$

Parameterized verification for distributed algorithms

Starting from...

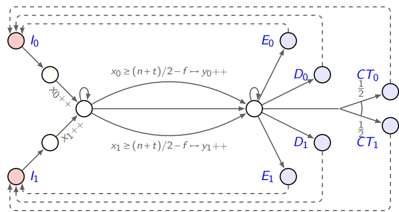
- algorithm pseudo-code

```
bool v := input_value({0, 1});
int  r := 1 ;
while (true) do
 send (R,r,v) to all;
 wait for n − t messages (R,r,∗);
 if received (n + t) / 2 messages (R,r,w)
 then send (P,r,w,D) to all;
 else send (P,r,?) to all;
 wait for n − t messages (P,r,∗);
 if received at least t + 1
    messages (P,r,w,D) then {
  v := w;
/∗ enough support −> update estimate ∗/
  if received at least (n + t) / 2
   messages (P,r,w,D)
  then decide w;
/∗ strong majority −> decide ∗/
 } else  v := random(0, 1) ;
/∗ unclear −> coin toss        ∗/
 r := r + 1 ;
od
```

- requirements
  validity, agreement
  (a.s.) termination

... derive

- model



- formulas

$\forall k, k' \in \mathbb{N}, \quad \mathbf{A}\left(\mathbf{F}\,\boldsymbol{\kappa}[D_V, k] > 0 \rightarrow \mathbf{G}\,\boldsymbol{\kappa}[D_{1-V}, k'] = 0\right)$

$\forall k \in \mathbb{N}\, \mathbf{A}\left(\mathbf{F}\,\boldsymbol{\kappa}[I_V, 0] = 0 \quad \rightarrow \quad \mathbf{G}\,\boldsymbol{\kappa}[D_V, k] = 0\right)$

$\mathbb{P}_{\mathfrak{a}}\left(\bigvee_{k \in \mathbb{N}} \bigvee_{V \in \{0,1\}} \mathbf{G} \bigwedge_{\ell \in \mathscr{L} \setminus \{D_V\}} \boldsymbol{\kappa}[\ell, k] = 0\right) = 1$

- **model checking algorithms**
- **prototype implementation**

Which distributed algorithms?

A variety of settings to explore

- **timing model**: asynchronous, synchronous, etc.
- **communication paradigm**: shared variable, broadcast, etc.
- **failure model**: no failures, crash, Byzantine processes
- **addressed problem**: consensus, leader election, DB consistency, etc.

Which distributed algorithms?

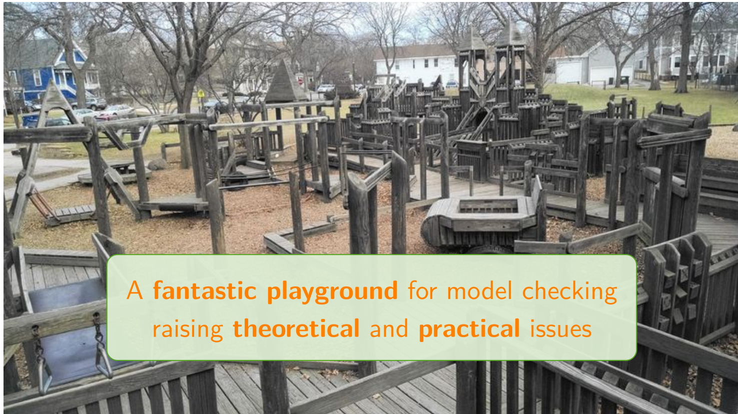
A variety of settings to explore

- **timing model**: asynchronous, synchronous, etc.
- **communication paradigm**: shared variable, broadcast, etc.
- **failure model**: no failures, crash, Byzantine processes
- **addressed problem**: consensus, leader election, DB consistency, etc.

Which distributed algorithms?

A variety of settings to explore

- **timing model**: asynchronous, synchronous, etc.
- **communication paradigm**: shared variable, broadcast, etc.
- **failure model**: no failures, crash, Byzantine processes
- **addressed problem**: consensus, leader election, DB consistency, etc.



A **fantastic playground** for model checking raising **theoretical** and **practical** issues

Outline

Introduction to model checking

Parameterized verification for distributed algorithms

Two frameworks for threshold-based fault-tolerant algorithms
- Threshold automata (with random choices)
- Predicate abstraction

Conclusion

Threshold automata

for fault-tolerant threshold-based distributed algorithms

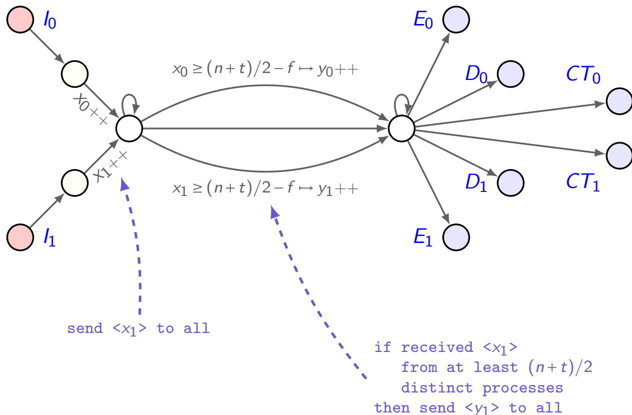
- locations represent algorithm control points
- shared variables count sent messages of each type
- guards as linear constraints on variables and parameters

[Konnov Veith Widder CAV'15, Konnov Lazić Veith Widder POPL'17]

Threshold automata

for fault-tolerant threshold-based distributed algorithms

- locations represent algorithm control points
- shared variables count sent messages of each type
- guards as linear constraints on variables and parameters



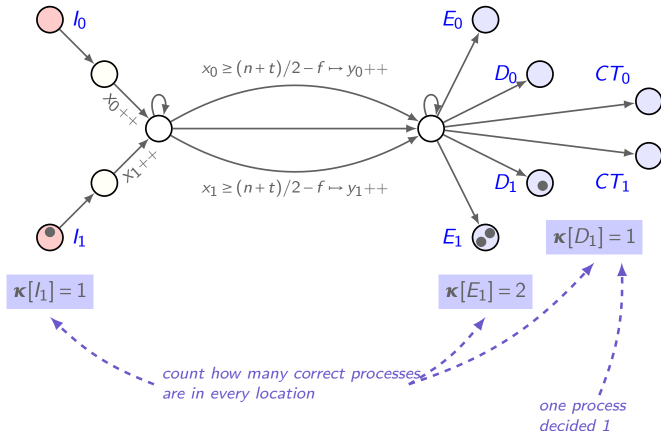
[Konnov Veith Widder CAV'15, Konnov Lazić Veith Widder POPL'17]

Semantics of threshold automata

- infinite counter system
  - finitely many `int` counters: 1 per location of the TA
  - unbounded counter values because of parameters

Semantics of threshold automata

- infinite counter system
  - finitely many `int` counters: 1 per location of the TA
  - unbounded counter values because of parameters



Specifying and verifying correctness

- Linear-time Temporal Logic (LTL) fragment without Next, and with counters
- atomic propositions: whether counter value is 0 or not

[Konnov Veith Widder CAV'15, Konnov Lazić Veith Widder POPL'17]

Specifying and verifying correctness

- Linear-time Temporal Logic (LTL) fragment without Next, and with counters
- atomic propositions: whether counter value is 0 or not

Agreement: No two correct processes decide differently (safety)

$$\mathbf{F}\ \boldsymbol{\kappa}[D_v] > 0 \quad \rightarrow \quad \mathbf{G}\ \boldsymbol{\kappa}[D_{1-v}] = 0$$

Termination: Eventually all correct processes decide (liveness)

$$\mathbf{F} \bigwedge_{\ell \in \mathscr{L} \setminus \{D_0, D_1\}} \boldsymbol{\kappa}[\ell] = 0$$

[Konnov Veith Widder CAV'15, Konnov Lazić Veith Widder POPL'17]

Specifying and verifying correctness

- Linear-time Temporal Logic (LTL) fragment without Next, and with counters
- atomic propositions: whether counter value is 0 or not

Agreement: No two correct processes decide differently (safety)

$$\mathbf{F}\ \boldsymbol{\kappa}[D_v] > 0 \quad \rightarrow \quad \mathbf{G}\ \boldsymbol{\kappa}[D_{1-v}] = 0$$

Termination: Eventually all correct processes decide (liveness)

$$\mathbf{F} \bigwedge_{\ell \in \mathscr{L} \setminus \{D_0, D_1\}} \boldsymbol{\kappa}[\ell] = 0$$

Given a threshold automaton TA, a specification φ in $\mathsf{ELTL}_{\mathsf{FT}}$, and a resilience condition RC
one can decide whether for all parameters satisfying RC, $\mathrm{Sys(TA)} \models \varphi$

Tool support: ByMC at `forsyte.at/software/bymc/`

[Konnov Veith Widder CAV'15, Konnov Lazić Veith Widder POPL'17]

How to handle randomization?

Ben Or's randomized algorithm for consensus [Ben Or PODC'83]

```
bool v := input_value({0, 1});
int r := 1 ;
while (true) do
 send (R,r,v) to all;
 wait for n − t messages (R,r,∗);
 if received (n + t) / 2 messages (R,r,w)
 then send (P,r,w,D) to all;
 else send (P,r,?) to all;
 wait for n − t messages (P,r,∗);
 if received at least t + 1
    messages (P,r,w,D) then {
  v := w;                      /∗ enough support −> update estimate ∗/
  if received at least (n + t) / 2
   messages (P,r,w,D)
  then decide w;                        /∗ strong majority −> decide ∗/
 } else v := random(0, 1) ;        /∗ unclear −> coin toss       ∗/
 r := r + 1 ;
od
```

How to handle randomization?

Ben Or's randomized algorithm for consensus [Ben Or PODC'83]

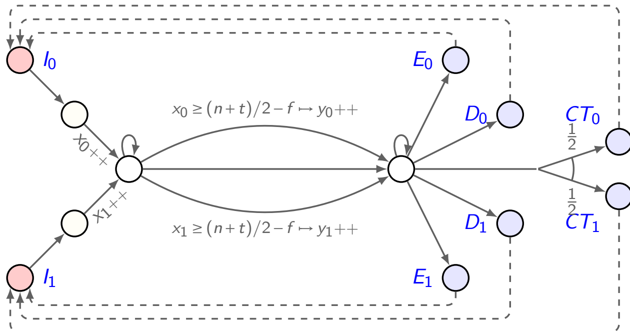
```
bool v := input_value({0, 1});
int r := 1 ;
while (true) do
 send (R,r,v) to all;
 wait for n − t messages (R,r,∗);
 if received (n + t) / 2 messages (R,r,w)
 then send (P,r,w,D) to all;
 else send (P,r,?) to all;
 wait for n − t messages (P,r,∗);
 if received at least t + 1
    messages (P,r,w,D) then {
  v := w;                  /∗ enough support −> update estimate ∗/
  if received at least (n + t) / 2
   messages (P,r,w,D)
  then decide w;                     /∗ strong majority −> decide ∗/
 } else v := random(0, 1) ;      /∗ unclear −> coin toss      ∗/
 r := r + 1 ;
od
```

Modeling challenges

- unboundedly many rounds
- probabilistic choices for local/global coin tosses

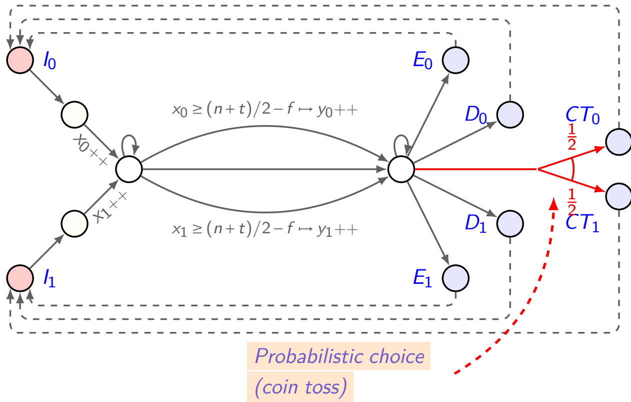
Probabilistic threshold automata

Ben Or's randomized consensus algorithm



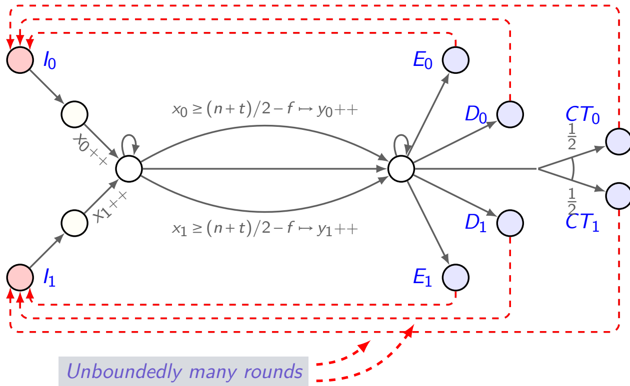
Probabilistic threshold automata

Ben Or's randomized consensus algorithm



Probabilistic threshold automata

Ben Or's randomized consensus algorithm



Specifying correctness

Agreement: No two correct processes decide differently (safety)

$$(\forall k \in \mathbb{N}_0)\,(\forall k' \in \mathbb{N}_0)\ \mathbf{A}\left(\mathbf{F}\ \boldsymbol{\kappa}[D_v, k] > 0 \quad \rightarrow \quad \mathbf{G}\ \boldsymbol{\kappa}[D_{1-v}, k'] = 0\right)$$

Validity: Any decided value was proposed initially (safety)

$$(\forall k \in \mathbb{N}_0)\ \mathbf{A}\left(\mathbf{F}\ \boldsymbol{\kappa}[I_{1-v}, 0] = 0 \quad \rightarrow \quad \mathbf{G}\ \boldsymbol{\kappa}[D_{1-v}, k] = 0\right)$$

Almost sure termination: under every adversary, with probability 1 every correct process eventually decides (prob. liveness)

$$\mathbb{P}_{\mathtt{a}}\left(\ \bigvee_{k\in\mathbb{N}_0} \bigvee_{v\in\{0,1\}} \mathbf{G} \bigwedge_{\ell\in\mathscr{L}\setminus\{D_v\}} \boldsymbol{\kappa}[\ell, k] = 0\right) = 1$$

Specifying correctness

Agreement: No two correct processes decide differently (safety)

$$(\forall k \in \mathbb{N}_0)\,(\forall k' \in \mathbb{N}_0)\ \mathbf{A}\ \big(\mathbf{F}\ \boldsymbol{\kappa}[D_v, k] > 0 \quad \rightarrow \quad \mathbf{G}\ \boldsymbol{\kappa}[D_{1-v}, k'] = 0\big)$$

Validity: Any decided value was proposed initially (safety)

$$(\forall k \in \mathbb{N}_0)\ \mathbf{A}\ \big(\mathbf{F}\ \boldsymbol{\kappa}[I_{1-v}, 0] = 0 \quad \rightarrow \quad \mathbf{G}\ \boldsymbol{\kappa}[D_{1-v}, k] = 0\big)$$

Almost sure termination: under every adversary, with probability 1 every correct process eventually decides (prob. liveness)

$$\mathbb{P}_{\mathsf{a}}\Big(\bigvee_{k \in \mathbb{N}_0} \bigvee_{v \in \{0,1\}} \mathbf{G} \bigwedge_{\ell \in \mathscr{L} \setminus \{D_v\}} \boldsymbol{\kappa}[\ell, k] = 0\Big) = 1$$

Verification challenges

- specifications over multiple rounds
- probabilistic guarantees

Verifying correctness

Safety properties

- must hold on **all** executions
  - probabilistic choices can be transformed into non-determinism
  - reduction to non-probabilistic threshold automata
- communication-closure
  - reduction to single round threshold automaton by reordering actions
- round switch invariants
  - reduction to single round specification

[**B.** Konnov Lazić Widder Concur'19]

Verifying correctness

Safety properties

- must hold on **all** executions
  - probabilistic choices can be transformed into non-determinism
  - reduction to non-probabilistic threshold automata
- communication-closure
  - reduction to single round threshold automaton by reordering actions
- round switch invariants
  - reduction to single round specification

Liveness property

- must hold on **almost all** executions
  - probability values do matter!
  - restriction to *round-rigid* adversaries to reorder actions within rounds
  - sufficient conditions on single round threshold automaton that imply almost sure termination

[**B.** Konnov Lazić Widder Concur'19]

Verifying correctness

Safety properties

- must hold on **all** executions
  - probabilistic choices can be transformed into non-determinism
  - reduction to non-probabilistic threshold automata
- communication-closure
  - reduction to single round threshold automaton by reordering actions
- round switch invariants
  - reduction to single round specification

Liveness property

- must hold on **almost all** executions
  - probability values do matter!
  - restriction to *round-rigid* adversaries to reorder actions within rounds
  - sufficient conditions on single round threshold automaton that imply almost sure termination

sufficient conditions only, approach mimicks manual proof

[**B.** Konnov Lazić Widder Concur'19]

Experimental evaluation

- 6 classical randomized distributed algorithms
- several one-round safety and liveness properties for each

Algorithm	Verif time per property
- Ben-Or's Byzantine random. consensus	≤ 1 sec
- Ben-Or's crash random. consensus	≤ 1 sec
- Ben-Or's clean crash random. consensus	≤ 1 sec
- Bracha's randomized consensus	≤ 1 sec
- Raynal's k-set agreement	3–40 sec
- Song's and van Renesse's BOSCO	3 hours on a cluster

Outline

Introduction to model checking

Parameterized verification for distributed algorithms

Two frameworks for threshold-based fault-tolerant algorithms
- Threshold automata (with random choices)
- Predicate abstraction

Conclusion

Threshold-based round-based algorithms

Phase King algorithm [Berman Garay, Mathematical Systems Theory 1993]

```
int id := identifier({0 ... n−1});
bool v := input_value({0, 1});
for r=0 to t do
    broadcast (r,id,v);
    receive all (r,_,_);
    if # of (r,_,0) received > n/2 + t /* majority of 0 */
       v := 0;
       else if # of (r,_,1) received > n/2 + t /* majority of 1 */
            v := 1;
            else v := v' where (r,r,v') received; /* new value is king value */
```

Threshold-based round-based algorithms

Phase King algorithm [Berman Garay, Mathematical Systems Theory 1993]

```
int id := identifier({0 ... n−1});
bool v := input_value({0, 1});
for r=0 to t do
    broadcast (r,id,v);
    receive all (r,_,_);
    if # of (r,_,0) received > n/2 + t /* majority of 0 */
       v := 0;
       else if # of (r,_,1) received > n/2 + t /* majority of 1 */
            v := 1;
            else v := v' where (r,r,v') received; /* new value is king value */
```

- threshold automaton with states arranged in **layers**/rounds
- processes send their local state as message contents
- unbounded number of rounds (parameter t)
- synchronous or asynchronous algorithms

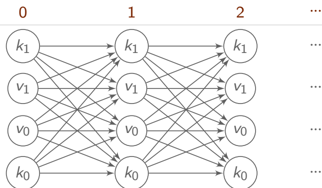
Threshold-based round-based algorithms

Phase King algorithm

[Berman Garay, Mathematical Systems Theory 1993]

```
int id := identifier({0 ... n−1});
bool v := input_value({0, 1});
for r=0 to t do
    broadcast (r,id,v);
    receive all (r,_,_);
    if # of (r,_,0) received > n/2 + t /* majority of 0 */
       v := 0;
       else if # of (r,_,1) received > n/2 + t /* majority of 1 */
            v := 1;
            else v := v' where (r,r,v') received; /* new value is king value */
```

- threshold automaton with states arranged in **layers**/rounds
- processes send their local state as message contents
- unbounded number of rounds (parameter t)
- synchronous or asynchronous algorithms



e.g. guard from v_1 to v_0

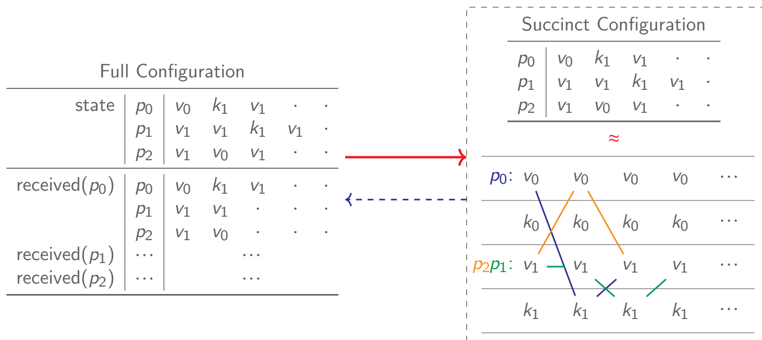
$v_0 + f > n/2 + t \vee (k_0 > 0 \wedge v_1 \leq n/2 + t)$

Abstraction steps

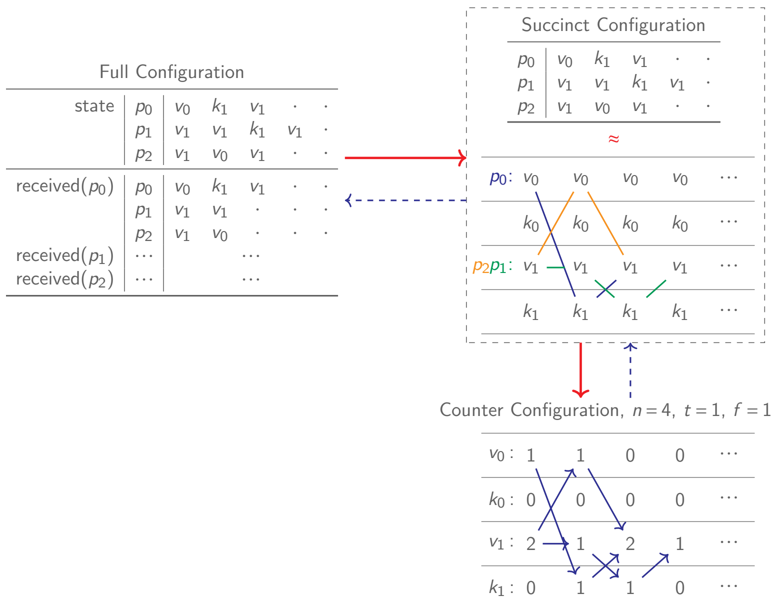
Full Configuration

state	p_0	v_0	k_1	v_1	·	·
	p_1	v_1	v_1	k_1	v_1	·
	p_2	v_1	v_0	v_1	·	·
received(p_0)	p_0	v_0	k_1	v_1	·	·
	p_1	v_1	v_1	·	·	·
	p_2	v_1	v_0	·	·	·
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		

Abstraction steps



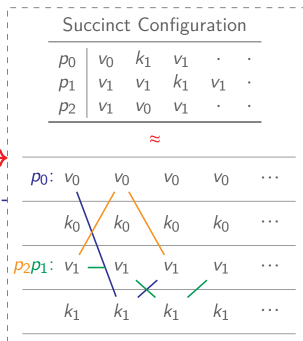
Abstraction steps



Abstraction steps

Full Configuration

state	p_0	v_0	k_1	v_1	·	·
	p_1	v_1	v_1	k_1	v_1	·
	p_2	v_1	v_0	v_1	·	·
received(p_0)	p_0	v_0	k_1	v_1	·	·
	p_1	v_1	v_1	·	·	·
	p_2	v_1	v_0	·	·	·
received(p_1)	⋯			⋯		
received(p_2)	⋯			⋯		



Guard Configuration

$v_0 > 0$	T	T	F	F	⋯
$k_0 > 0$	F	F	F	F	⋯
$v_1 > 0$	T	T	T	T	⋯
$k_1 > 0$	F	T	T	F	⋯
$2(v_0 + k_0 + f) > n + 2t$	F	F	F	F	⋯
$2(v_1 + k_1 + f) > n + 2t$	F	F	T	F	⋯
$2(v_0 + k_0) > n + 2t$	F	F	F	F	⋯
$2(v_1 + k_1) > n + 2t$	F	F	F	F	⋯
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	F	⋯

Counter Configuration, $n = 4$, $t = 1$, $f = 1$

v_0:	1	1	0	0	⋯
k_0:	0	0	0	0	⋯
v_1:	2	1	2	1	⋯
k_1:	0	1	1	0	⋯

Guard automaton

$v_0 > 0$	T	T	F	F	F	$\cdots$
$k_0 > 0$	F	F	F	F	F	$\cdots$
$v_1 > 0$	T	T	T	T	F	$\cdots$
$k_1 > 0$	F	T	T	F	F	$\cdots$
$2(v_0+k_0+f) > n+2t$	F	F	F	F	F	$\cdots$
$2(v_1+k_1+f) > n+2t$	F	F	T	F	F	$\cdots$
$2(v_0+k_0) > n+2t$	F	F	F	F	F	$\cdots$
$2(v_1+k_1) > n+2t$	F	F	F	F	F	$\cdots$
$v_0+k_0+v_1+k_1+f \geq n$	T	T	T	F	F	$\cdots$

guard configuration

[**B.** Thomas Widder Concur'20]

Guard automaton

	γ_0	γ_1	γ_2	γ_3	γ_4	
$v_0 > 0$	T	T	F	F	F	$\cdots$
$k_0 > 0$	F	F	F	F	F	$\cdots$
$v_1 > 0$	T	T	T	T	F	$\cdots$
$k_1 > 0$	F	T	T	F	F	$\cdots$
$2(v_0 + k_0 + f) > n + 2t$	F	F	F	F	F	$\cdots$
$2(v_1 + k_1 + f) > n + 2t$	F	F	T	F	F	$\cdots$
$2(v_0 + k_0) > n + 2t$	F	F	F	F	F	$\cdots$
$2(v_1 + k_1) > n + 2t$	F	F	F	F	F	$\cdots$
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	F	F	$\cdots$

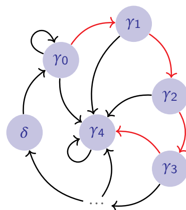
guard configuration

[**B.** Thomas Widder Concur'20]

Guard automaton

	γ_0	γ_1	γ_2	γ_3	γ_4	$\cdots$
$v_0 > 0$	T	T	F	F	F	$\cdots$
$k_0 > 0$	F	F	F	F	F	$\cdots$
$v_1 > 0$	T	T	T	T	F	$\cdots$
$k_1 > 0$	F	T	T	F	F	$\cdots$
$2(v_0 + k_0 + f) > n + 2t$	F	F	F	F	F	$\cdots$
$2(v_1 + k_1 + f) > n + 2t$	F	F	T	F	F	$\cdots$
$2(v_0 + k_0) > n + 2t$	F	F	F	F	F	$\cdots$
$2(v_1 + k_1) > n + 2t$	F	F	F	F	F	$\cdots$
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	F	F	$\cdots$

guard configuration



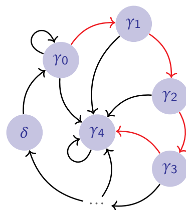
guard automaton

[**B.** Thomas Widder Concur'20]

Guard automaton

	γ_0	γ_1	γ_2	γ_3	γ_4	
$v_0 > 0$	T	T	F	F	F	⋯
$k_0 > 0$	F	F	F	F	F	⋯
$v_1 > 0$	T	T	T	T	F	⋯
$k_1 > 0$	F	T	T	F	F	⋯
$2(v_0 + k_0 + f) > n + 2t$	F	F	F	F	F	⋯
$2(v_1 + k_1 + f) > n + 2t$	F	F	T	F	F	⋯
$2(v_0 + k_0) > n + 2t$	F	F	F	F	F	⋯
$2(v_1 + k_1) > n + 2t$	F	F	F	F	F	⋯
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	F	F	⋯

guard configuration



guard automaton

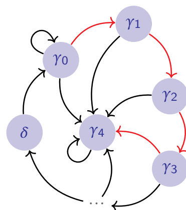
- fixed set of predicates
- transitions in guard automaton via predicate abstraction (SMT solver)

[**B.** Thomas Widder Concur'20]

Guard automaton

	γ_0	γ_1	γ_2	γ_3	γ_4	⋯
$v_0 > 0$	T	T	F	F	F	⋯
$k_0 > 0$	F	F	F	F	F	⋯
$v_1 > 0$	T	T	T	T	F	⋯
$k_1 > 0$	F	T	T	F	F	⋯
$2(v_0 + k_0 + f) > n + 2t$	F	F	F	F	F	⋯
$2(v_1 + k_1 + f) > n + 2t$	F	F	T	F	F	⋯
$2(v_0 + k_0) > n + 2t$	F	F	F	F	F	⋯
$2(v_1 + k_1) > n + 2t$	F	F	F	F	F	⋯
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	F	F	⋯

guard configuration



guard automaton

- fixed set of predicates
- transitions in guard automaton via predicate abstraction (SMT solver)

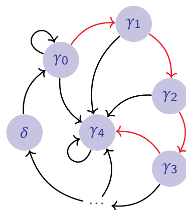
The language of the guard automaton **overapproximates** the set of all executions of the layered threshold automaton.

[**B.** Thomas Widder Concur'20]

Guard automaton

	γ_0	γ_1	γ_2	γ_3	γ_4	
$v_0 > 0$	T	T	F	F	F	⋯
$k_0 > 0$	F	F	F	F	F	⋯
$v_1 > 0$	T	T	T	T	F	⋯
$k_1 > 0$	F	T	T	F	F	⋯
$2(v_0 + k_0 + f) > n + 2t$	F	F	F	F	F	⋯
$2(v_1 + k_1 + f) > n + 2t$	F	F	T	F	F	⋯
$2(v_0 + k_0) > n + 2t$	F	F	F	F	F	⋯
$2(v_1 + k_1) > n + 2t$	F	F	F	F	F	⋯
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	F	F	⋯

guard configuration



guard automaton

- fixed set of predicates
- transitions in guard automaton via predicate abstraction (SMT solver)

The language of the guard automaton **overapproximates** the set of all executions of the layered threshold automaton.

⚠ Incomplete method yet **sufficient to prove correctness** of Phase King algorithm + **analysis can be refined** by adding more predicates

[B. Thomas Widder Concur'20]

How good are guard automata and predicate abstraction?

Drawbacks and advantages compared to threshold automata approach

- ⊖ layered assumption on algorithm structure
- ⊖ formalization using domain theory concepts
- ⊖ randomization not supported

How good are guard automata and predicate abstraction?

Drawbacks and advantages compared to threshold automata approach

- ⊖ layered assumption on algorithm structure
- ⊖ formalization using domain theory concepts
- ⊖ randomization not supported

- ⊕ formal relation between distributed algorithms and model semantics
- ⊕ allows to check for liveness properties on infinite executions
- ⊕ enables refinement
- ⊕ stand alone implementation

Outline

Introduction to model checking

Parameterized verification for distributed algorithms

Two frameworks for threshold-based fault-tolerant algorithms
- Threshold automata (with random choices)
- Predicate abstraction

Conclusion

Parameterized verification of distributed algorithms

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation
- are **generic**
  one model-checking algorithm for a class of distributed algorithms

Parameterized verification of distributed algorithms

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation
- are **generic**
  one model-checking algorithm for a class of distributed algorithms

Half a dozen of frameworks so far

- broadcast protocols [Esparza Finkel Mayr LICS'99] [Delzanno Sangnier Zavattaro Concur'10]
- shared-memory models [Esparza Ganty Majumdar JACM 2016] [Bouyer Markey Randour Sangnier Stan ICALP'16]
- randomized algorithms on rings [Lin Rümmer CAV'16]
- synchronous algorithms on rings [Aiswarya Bollig Gastin I&C 2018]
- population protocols [Esparza Ganty Leroux Majumdar Acta Inf. 2017]

Parameterized verification of distributed algorithms

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation
- are **generic**
  one model-checking algorithm for a class of distributed algorithms

Half a dozen of frameworks so far

- broadcast protocols [Esparza Finkel Mayr LICS'99] [Delzanno Sangnier Zavattaro Concur'10]
- shared-memory models [Esparza Ganty Majumdar JACM 2016] [Bouyer Markey Randour Sangnier Stan ICALP'16]
- randomized algorithms on rings [Lin Rümmer CAV'16]
- synchronous algorithms on rings [Aiswarya Bollig Gastin I&C 2018]
- population protocols [Esparza Ganty Leroux Majumdar Acta Inf. 2017]

Two available tools to try out: ByMC, Peregrine

Still many challenges for model checking

Current interests

- compositional analysis of threshold automata to improve scalability
- shared-memory consensus algorithms [Aspnes JACM 2002]

Mid- to long-term objectives

- model extraction from pseudo-code
- quantitative analysis of randomized consensus algorithms
- complexity measures computation (*e.g.* convergence time)
- automated synthesis of correct-by design distributed algorithms

Still many challenges for model checking

Current interests

- compositional analysis of threshold automata to improve scalability
- shared-memory consensus algorithms [Aspnes JACM 2002]

Mid- to long-term objectives

- model extraction from pseudo-code
- quantitative analysis of randomized consensus algorithms
- complexity measures computation (*e.g.* convergence time)
- automated synthesis of correct-by design distributed algorithms

Thanks for your attention!