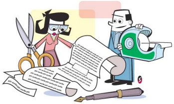
Parameterized verification of round-based distributed algorithms

Nathalie Bertrand
Inria Rennes & IRISA

ETAPS 2022 - Munich

based on joint work with
Bastien Thomas, Josef Widder
Nicolas Markey, Ocan Sankur, Nicolas Waldburger

Model checking in a nutshell

Does  satisfy  ?

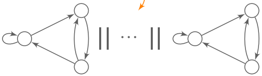
system requirement

Model checking in a nutshell

Does  satisfy  ?

system

requirement



$\mathbf{AG}(\bigwedge_{i \neq j}(\mathsf{Crit}_i \implies \neg \mathsf{Crit}_j))$

model

property

Model checking in a nutshell

Does  satisfy  ?

system

requirement

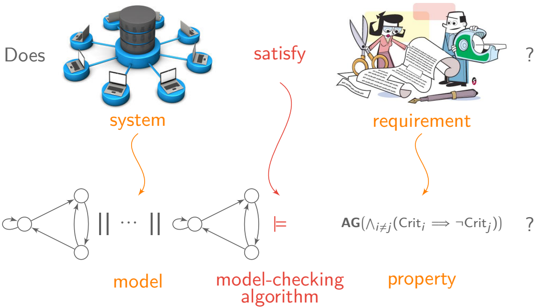
 $\models$ $\mathbf{AG}(\wedge_{i\neq j}(\mathsf{Crit}_i \implies \neg\mathsf{Crit}_j))$?

model

model-checking algorithm

property

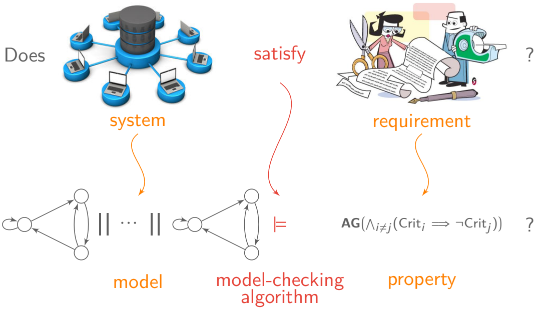
Model checking in a nutshell



⊕ 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

Standard model checking for distributed algorithms

Peterson's algorithm [Peterson IPL 1981]

- mutual exclusion
- processes P_0 and P_1
- shared variables x, b_0 and b_1 (b_i read-only to P_{i-1})

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

Standard model checking for distributed algorithms

Peterson's algorithm [Peterson IPL 1981]

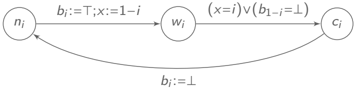
- mutual exclusion
- processes P_0 and P_1
- shared variables x, b_0 and b_1 (b_i read-only to P_{i-1})

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

Correctness: *processes are not in their critical section simultaneously*

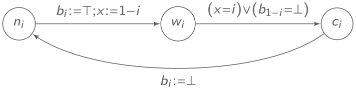
Modelling and verifying Peterson's algorithm

[Baier Katoen MIT Press 2008]

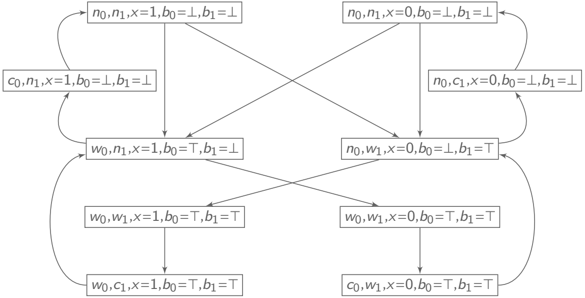


Modelling and verifying Peterson's algorithm

[Baier Katoen MIT Press 2008]

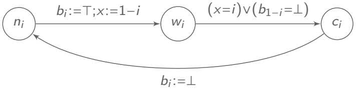


Product transition system representing all interleavings

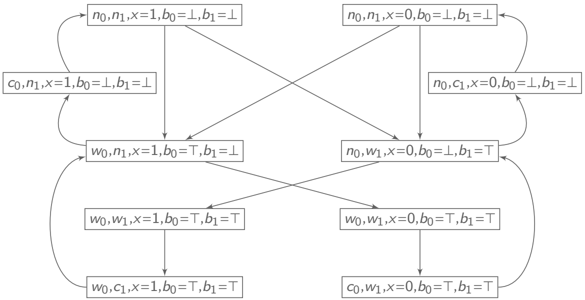


Modelling and verifying Peterson's algorithm

[Baier Katoen MIT Press 2008]



Product transition system representing all interleavings



correctness reduces to: no state $(c_0, c_1, \_, \_, \_)$ is reachable

Limitations of standard model checking techniques for the verification of distributed algorithms

⚠ **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 for the verification of distributed algorithms

⚠ **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 for the verification of distributed algorithms

⚠ **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**
→ correctness should be proven for arbitrary number of processes

Parameterized verification: to infinity and beyond!

from 

to 

Parameterized verification: to infinity and beyond!

from  to 

- 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 every number of participants, for every network topologies, for every potential failures, for every *parameter valuations*

Parameterized verification: to infinity and beyond!

from  to 

- 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 every number of participants, for every network topologies, for every potential failures, for every *parameter valuations*

⚠ model checking infinitely many instances at once

Parameterized verification for distributed algorithms

From... algorithm pseudo-code and requirements

```
bool v := input_value({0, 1});
int r := 1 ;
a0 := [1,0,0...]; a1 := [1,0,0,...];
while (true) do
 read a0[r] and a1[r];
 if ∃b, ab[r] = 1 and a1−b[r] = 0
 then v:= b; fi
 write 1 in av[r]
 read a1−v[r−1];
 if a1−v[r−1] = 0
 then return v;
 else r := r+1 ; fi od
```

- correctness for all n

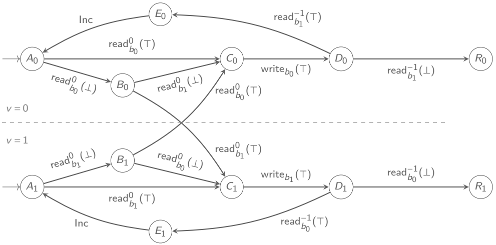
Parameterized verification for distributed algorithms

From... algorithm pseudo-code and requirements

```
bool v := input_value({0, 1});
int  r := 1 ;
a0 := [1,0,0...]; a1 := [1,0,0,...];
while (true) do
 read a0[r] and a1[r];
 if ∃b, ab[r] = 1 and a1−b[r] = 0
 then v:= b; fi
 write 1 in av[r]
 read a1−v[r−1];
 if a1−v[r−1] = 0
 then return v;
 else r := r+1 ; fi od
```

- correctness for all n

... derive model, formulas



- $\varphi_1 \wedge \cdots \wedge \varphi_k$ (independent of n)

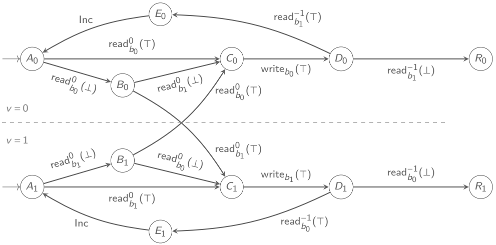
Parameterized verification for distributed algorithms

From... algorithm pseudo-code and requirements

```
bool v := input_value({0, 1});
int  r := 1 ;
a0 := [1,0,0...]; a1 := [1,0,0,...];
while (true) do
 read a0[r] and a1[r];
 if ∃b, ab[r] = 1 and a1−b[r] = 0
 then v:= b; fi
 write 1 in av[r]
 read a1−v[r−1];
 if a1−v[r−1] = 0
 then return v;
 else  r := r+1 ;  fi od
```

- correctness for all n

... derive model, formulas, **model checking algorithms and tools**



- $\varphi_1 \wedge \cdots \wedge \varphi_k$ (independent of n)

Which distributed algorithms?

A variety of settings to explore

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

Which distributed algorithms?

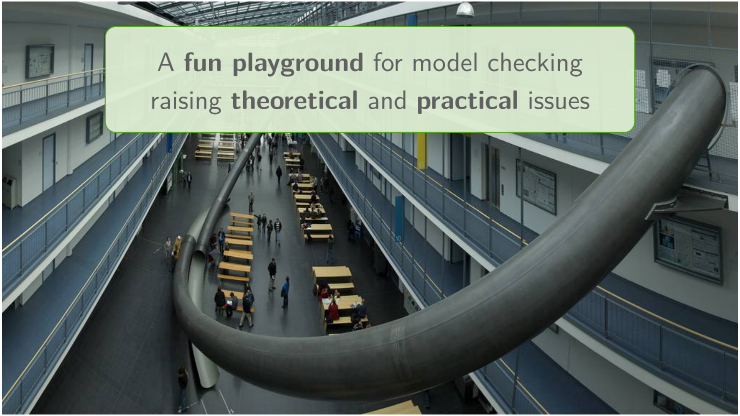
A variety of settings to explore

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

Which distributed algorithms?

A variety of settings to explore

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

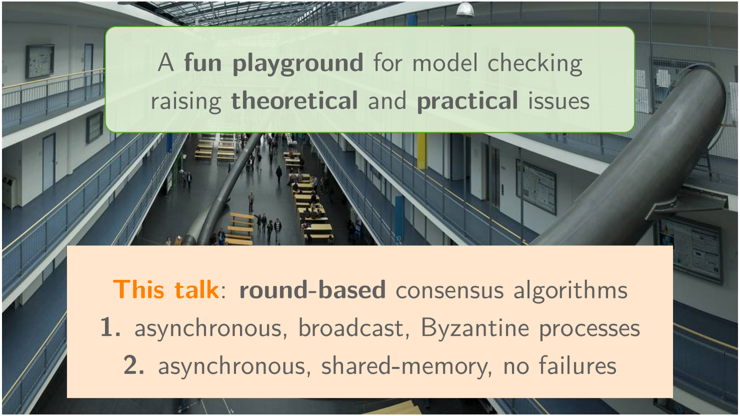


A **fun playground** for model checking
raising **theoretical** and **practical** issues

Which distributed algorithms?

A variety of settings to explore

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



Round-based algorithms for consensus

Consensus

- fundamental problem in distributed computing
- processes each with an initial value must agree on a common value
- difficult problem under asynchrony and/or failures

[Fischer Lynch Paterson JACM 1985]

Round-based algorithms for consensus

Consensus

- fundamental problem in distributed computing
- processes each with an initial value must agree on a common value
- difficult problem under asynchrony and/or failures

[Fischer Lynch Paterson JACM 1985]

Requirements for consensus algorithms

agreement	all correct processes decide the same value
validity	values decided by correct processes must be initial ones
termination	eventually all correct processes decide

Round-based algorithms for consensus

Consensus

- fundamental problem in distributed computing
- processes each with an initial value must agree on a common value
- difficult problem under asynchrony and/or failures

[Fischer Lynch Paterson JACM 1985]

Requirements for consensus algorithms

agreement	all correct processes decide the same value
validity	values decided by correct processes must be initial ones
termination	eventually all correct processes decide

Rounds are useful:

- for a correct process to be once the leader [Berman Garay MST 1993]
- to eventually sample a common value in randomized algorithms [BenOr PODC'85]
- for asynchrony to help a correct process to decide [Aspnes JA 1992]

Part 1: Broadcast fault-tolerant algorithms

Threshold-based round-based fault-tolerant algorithms

Phase King algorithm [Berman Garay MST 1993]

- binary consensus
- n processes communicate by **broadcasts** in **synchronous rounds**
- t is a known upper bound on unknown number of faulty processes f

Threshold-based round-based fault-tolerant algorithms

Phase King algorithm

[Berman Garay MST 1993]

- binary consensus
- n processes communicate by **broadcasts** in **synchronous rounds**
- t is a known upper bound on unknown number of faulty processes f

```
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; /* adopt value 0 */
       else if # of (r,_,1) received > n/2 + t /* majority of 1 */
            v := 1; /* adopt value 1 */
            else v := v' where (r,r,v') received; /* adopt king value */
```

- local variable v stores current value
- at round r, process with id r is the King
- if majority is unclear, processes adopt King's value for next round

Modelling Phase King algorithm

Layered threshold automata

variant of threshold automata [Konnov Veith Widder CAV'15]

capture asynchronous or synchronous communications

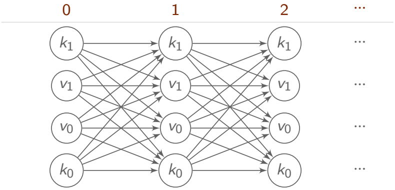
Modelling Phase King algorithm

Layered threshold automata

variant of threshold automata [Konnov Veith Widder CAV'15]

capture asynchronous or synchronous communications

- one model for all processes — identifiers abstracted away
- automaton with states arranged in layers (finer than rounds in general)
  k_b: King and value b; v_b not King and value b
- unbounded number of rounds (parameter t)



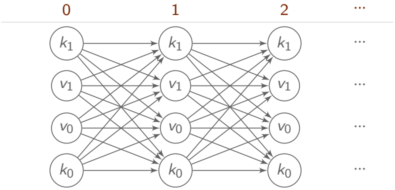
Modelling Phase King algorithm

Layered threshold automata

variant of threshold automata [Konnov Veith Widder CAV'15]

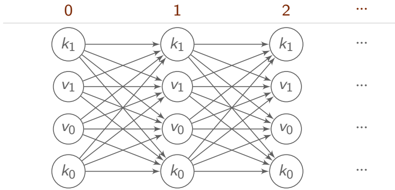
capture asynchronous or synchronous communications

- one model for all processes — identifiers abstracted away
- automaton with states arranged in layers (finer than rounds in general)
  k_b: King and value b; v_b not King and value b
- unbounded number of rounds (parameter t)



- processes broadcast their local state
- threshold guards on transitions ⚠ constraining current layer only
  e.g. $\mathbf{g}(v_0^r, v_1^{r+1}) = v_1^r + f > n/2 + t \;\vee\; \left(v_0^r \leq n/2 + t \;\wedge\; k_1^r > 0\right)$

Semantics of layered threshold automata



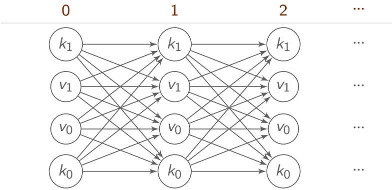
$$\mathbf{g}(v_0, v_1) = v_1 + f > n/2 + t \ \vee \\ (k_1 > 0 \ \wedge \ v_0 \leq n/2 + t)$$

Full configuration stores for each process
history of local states, and received messages from every process

Example with $n = 4$, $f = t = 1$

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

Semantics of layered threshold automata



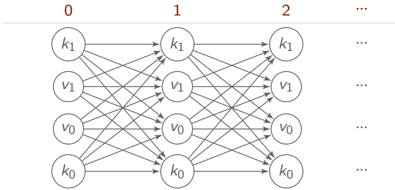
$$\mathbf{g}(v_0, v_1) = v_1 + f > n/2 + t \ \vee \ (k_1 > 0 \ \wedge \ v_0 \leq n/2 + t)$$

Full configuration stores for each process
history of local states, and received messages from every process

Example with $n = 4, \ f = t = 1$

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

Semantics of layered threshold automata



$$\mathbf{g}(v_0, v_1) = v_1 + f > n/2 + t \ \vee \ (k_1 > 0 \ \wedge \ v_0 \leq n/2 + t)$$

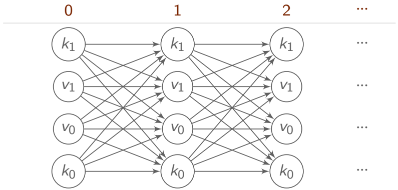
Full configuration stores for each process history of local states, and received messages from every process

Example with $n = 4,\ f = t = 1$

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

$$\xrightarrow{k_1 > 0 \ \wedge \ v_0 \leq n/2 + t}$$

Semantics of layered threshold automata



$$\mathbf{g}(v_0, v_1) = v_1 + f > n/2 + t \ \vee \ (k_1 > 0 \ \wedge \ v_0 \le n/2 + t)$$

Full configuration stores for each process
history of local states, and received messages from every process

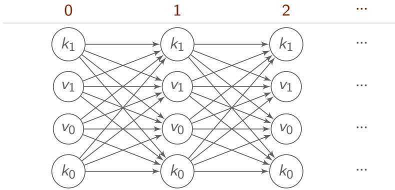
Example with $n = 4,\ f = t = 1$

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

$$\xrightarrow{k_1 > 0 \ \wedge \ v_0 \le n/2 + t}$$

v_1	v_0	v_1	·	·
v_1	v_1	k_1	v_1	·
v_0	k_1	v_1	·	·
v_1	v_0	·	·	·
v_1	v_1	k_1	·	·
v_0	k_1	·	·	·
		···		
		···		

Semantics of layered threshold automata



$$\mathbf{g}(v_0, v_1) = v_1 + f > n/2 + t \ \vee \ (k_1 > 0 \ \wedge \ v_0 \le n/2 + t)$$

Full configuration stores for each process
history of local states, and received messages from every process

Example with $n = 4$, $f = t = 1$

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

$$\xrightarrow{k_1 > 0 \ \wedge \ v_0 \le n/2 + t}$$

v_1	v_0	v_1	·	·
v_1	v_1	k_1	v_1	·
v_0	k_1	v_1	·	·
v_1	v_0	·	·	·
v_1	v_1	k_1	v_1	·
v_0	k_1	·	·	·
		⋯		
		⋯		

Model checking layered threshold automata

The parameterized model checking of layered threshold automata is **undecidable**, for **safety properties** already.

Model checking layered threshold automata

The parameterized model checking of layered threshold automata is **undecidable**, for **safety properties** already.

Our approach: incomplete yet refinable method

1. successive abstractions of semantics: removal of received messages (thanks to layered assumption); **counting** abstraction
2. **overapproximation** of sets of behaviours by a *guard automaton* using **predicate abstraction**; enabling **refinement** by adding more predicates

[**B.** Thomas Widder Concur'21]

Abstraction steps

Full Configuration

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

Abstraction steps

Full Configuration

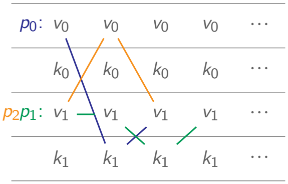
state	p_0	v_0	k_1	v_1	$\cdot$	$\cdot$
	p_1	v_1	v_1	k_1	v_1	$\cdot$
	p_2	v_1	v_0	v_1	$\cdot$	$\cdot$
received(p_0)	p_0	v_0	k_1	v_1	$\cdot$	$\cdot$
	p_1	v_1	v_1	$\cdot$	$\cdot$	$\cdot$
	p_2	v_1	v_0	$\cdot$	$\cdot$	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		

layered hyp.

Succinct Configuration

p_0	v_0	k_1	v_1	$\cdot$	$\cdot$
p_1	v_1	v_1	k_1	v_1	$\cdot$
p_2	v_1	v_0	v_1	$\cdot$	$\cdot$

$\approx$



Abstraction steps

Full Configuration

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

layered hyp.

Succinct Configuration

p_0	v_0	k_1	v_1	$\cdot$	$\cdot$
p_1	v_1	v_1	k_1	v_1	$\cdot$
p_2	v_1	v_0	v_1	$\cdot$	$\cdot$

$\approx$

p_0:	v_0	v_0	v_0	v_0	$\cdots$
	k_0	k_0	k_0	k_0	$\cdots$
$p_2 p_1$:	v_1	v_1	v_1	v_1	$\cdots$
	k_1	k_1	k_1	k_1	$\cdots$

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

v_0 :	1	1	0	0	$\cdots$
k_0 :	0	0	0	0	$\cdots$
v_1 :	2	1	2	1	$\cdots$
k_1 :	0	1	1	0	$\cdots$

Abstraction steps

Full Configuration

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

layered hyp.

Succinct Configuration

p_0	v_0	k_1	v_1	$\cdot$	$\cdot$
p_1	v_1	v_1	k_1	v_1	$\cdot$
p_2	v_1	v_0	v_1	$\cdot$	$\cdot$

$\approx$

p_0:	v_0	v_0	v_0	v_0	$\cdots$
	k_0	k_0	k_0	k_0	$\cdots$
$p_2 p_1$:	v_1	v_1	v_1	v_1	$\cdots$
	k_1	k_1	k_1	k_1	$\cdots$

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

v_0 :	1	1	0	0	$\cdots$
k_0 :	0	0	0	0	$\cdots$
v_1 :	2	1	2	1	$\cdots$
k_1 :	0	1	1	0	$\cdots$

Guard Configuration

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

Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

$v_0 > 0$	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

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

	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

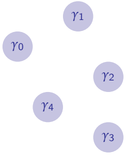
Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

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

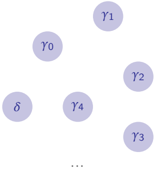
Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

	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



guard automaton

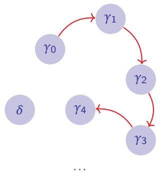
Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

	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

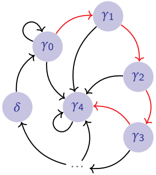
Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

	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



guard automaton

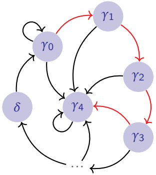
Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

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

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

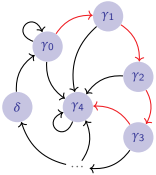
Guard automaton

Finite set of predicates: taken from formula and transition guards

- states = valuations of predicates
- transitions obtained via predicate abstraction; automated with SMT solver

	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



guard automaton

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

⚠ Incomplete method yet
sufficient to **prove correctness** of Phase King (safety and liveness)
possible **refinement** by adding predicates

Part 2: Shared-memory algorithms

Shared-memory round-based algorithms

Aspnes' algorithm [Aspnes JA 1992]

- binary consensus in noisy environment
- n processes **asynchronously** write to and read from **shared registers**

Shared-memory round-based algorithms

Aspnes' algorithm

[Aspnes JA 1992]

- binary consensus in noisy environment
- n processes **asynchronously** write to and read from **shared registers**

```
bool v := input_value({0, 1});
int r := 1 ;
b0 := [⊤,⊥,⊥...]; b1 := [⊤,⊥,⊥,...]; /* 2 registers per round */
while (true) do
 read b0[r] and b1[r]; /* reading current round registers */
 if ∃w, bw[r] = ⊤ and b1−w[r] = ⊥
 then v:= w; fi
 write ⊤ in bv[r] /* proposing value v by setting register to ⊤ */
 read b1−v[r−1];
 if b1−v[r−1] = ⊥ /* checking noone proposed 1−v in previous round */
 then return v;
 else r := r+1 ; fi od
```

- local variable v stores current value
- a process at round r can read from registers of rounds $r-1$ and r, and write to round r registers
- value v is returned if no process already proposed opposite value $1-v$ in last and current round

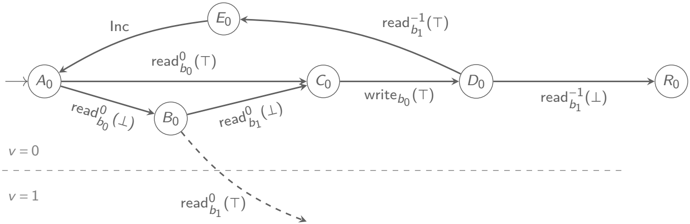
Modelling Aspnes' algorithm

Shared-memory protocols with rounds
extend shared memory protocols [Esparza Ganty Majumdar JACM 2016]

Modelling Aspnes' algorithm

Shared-memory protocols with rounds
extend shared memory protocols [Esparza Ganty Majumdar JACM 2016]

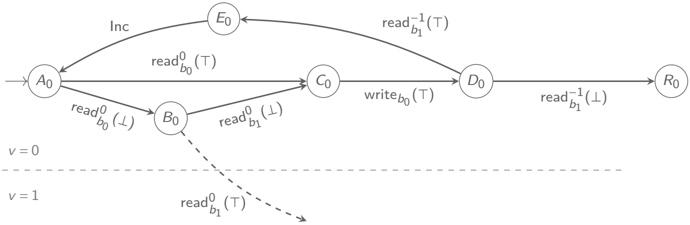
- one model for all processes
- unbounded number of rounds
- **d** shared registers per round (unboundedly many in total)



Modelling Aspnes' algorithm

Shared-memory protocols with rounds
extend shared memory protocols [Esparza Ganty Majumdar JACM 2016]

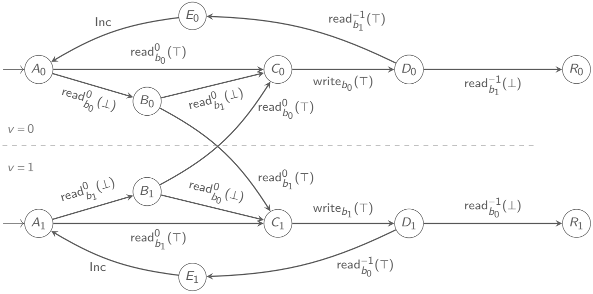
- one model for all processes
- unbounded number of rounds
- **d** shared registers per round (unboundedly many in total)



- actions: read from current and previous registers within window **w**, write to current registers, round increment

$$\mathbf{d} = 2,\ \mathbf{w} = 1,\ \mathrm{read}^{0}_{b_0}(\bot),\ \mathrm{read}^{-1}_{b_1}(\top),\ \mathrm{write}_{b_0}(\top),\ \mathrm{Inc}$$

Semantics of shared-memory models with rounds

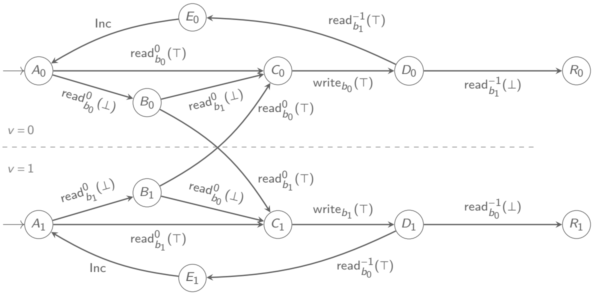


Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n = 3$

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \perp$	$E_0, 0$
round 1	$b_0 : \perp$	$C_1, 0$
	$b_1 : \perp$	
	$\vdots$	

Semantics of shared-memory models with rounds

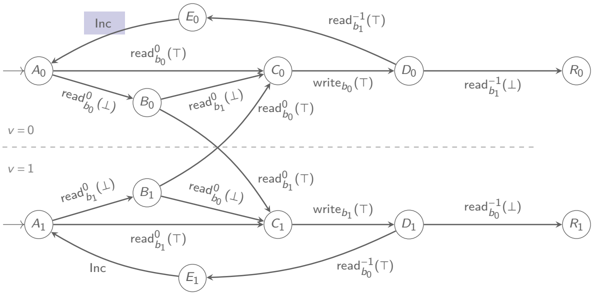


Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n = 3$

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$E_0, 0$
round 1	$b_0 : \bot$	$C_1, 0$
	$b_1 : \bot$	
	$\vdots$	

Semantics of shared-memory models with rounds



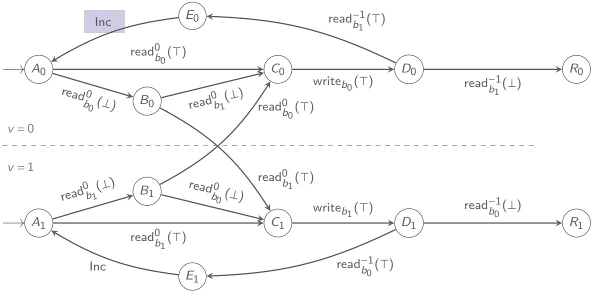
Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n = 3$

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$\boxed{E_0, 0}$
round 1	$b_0 : \bot$	$C_1, 0$
	$b_1 : \bot$	
	$\vdots$	

$\xrightarrow{\text{Inc}}$

Semantics of shared-memory models with rounds



Concrete configuration stores
values of shared registers, and for each process its local state and round

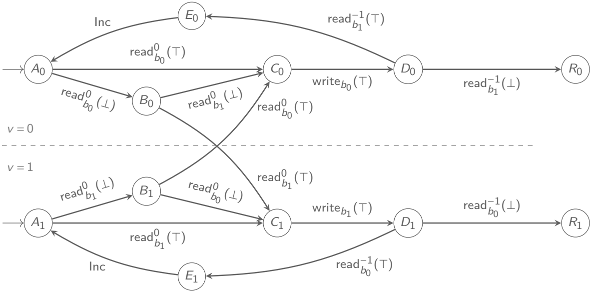
Example with $n = 3$

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$E_0, 0$
round 1	$b_0 : \bot$	$C_1, 0$
	$b_1 : \bot$	
	$\vdots$	

$\xrightarrow{\text{Inc}}$

$\top$	$C_0, 1$
$\bot$	$A_0, 1$
$\bot$	$C_1, 0$
$\bot$	
$\vdots$	

Semantics of shared-memory models with rounds



Concrete configuration stores
values of shared registers, and for each process its local state and round

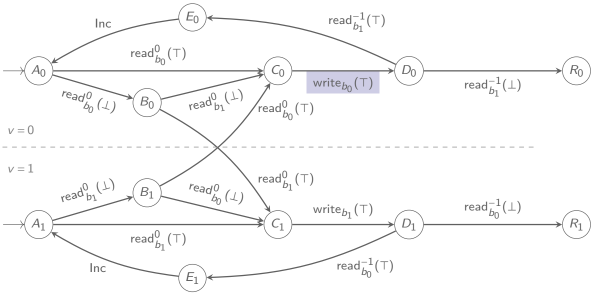
Example with $n = 3$

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$E_0, 0$
round 1	$b_0 : \bot$	$C_1, 0$
	$b_1 : \bot$	
	$\vdots$	

$\xrightarrow{\text{Inc}}$

$\top$	$C_0, 1$
$\bot$	$A_0, 1$
$\bot$	$C_1, 0$
$\bot$	
$\vdots$	

Semantics of shared-memory models with rounds



Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n=3$

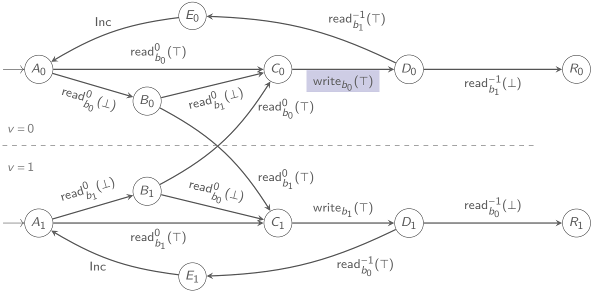
round 0	$b_0:\top$	$C_0,1$
	$b_1:\bot$	$E_0,0$
round 1	$b_0:\bot$	$C_1,0$
	$b_1:\bot$	
	$\vdots$	

$\xrightarrow{\text{Inc}}$

$\top$	$C_0,1$
$\bot$	$A_0,1$
$\bot$	$C_1,0$
$\bot$	
$\vdots$	

$\xrightarrow{\text{write}_{b_0}(\top)}$

Semantics of shared-memory models with rounds

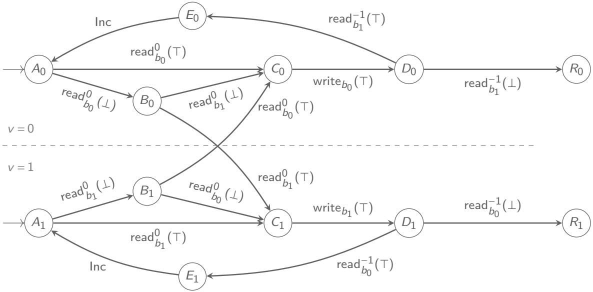


Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n = 3$

$$
\begin{array}{ll|l}
\text{round } 0 & b_0 : \top & C_0, 1 \\
 & b_1 : \bot & E_0, 0 \\
\hline
\text{round } 1 & b_0 : \bot & C_1, 0 \\
 & b_1 : \bot & \\
\hline
 & \vdots &
\end{array}
\xrightarrow{\text{Inc}}
\begin{array}{c|l}
\top & C_0, 1 \\
\bot & A_0, 1 \\
\hline
\bot & C_1, 0 \\
\bot & \\
\hline
\vdots &
\end{array}
\xrightarrow{\mathrm{write}_{b_0}(\top)}
\begin{array}{c|l}
\top & \boxed{D_0, 1} \\
\bot & A_0, 1 \\
\hline
\boxed{\top} & C_1, 0 \\
\bot & \\
\hline
\vdots &
\end{array}
$$

Semantics of shared-memory models with rounds



Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n = 3$

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$E_0, 0$
round 1	$b_0 : \bot$	$C_1, 0$
	$b_1 : \bot$	
	$\vdots$	

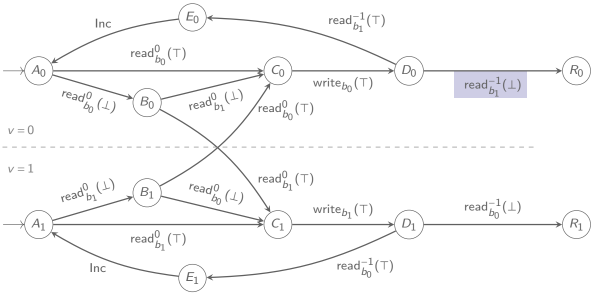
$\xrightarrow{\mathsf{Inc}}$

$\top$	$C_0, 1$
$\bot$	$A_0, 1$
$\bot$	$C_1, 0$
$\bot$	
$\vdots$	

$\xrightarrow{\mathsf{write}_{b_0}(\top)}$

$\top$	$D_0, 1$
$\bot$	$A_0, 1$
$\top$	$C_1, 0$
$\bot$	
$\vdots$	

Semantics of shared-memory models with rounds

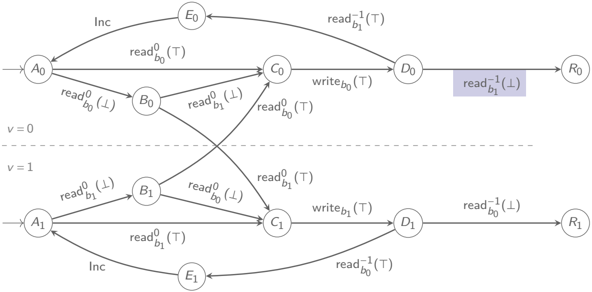


Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n=3$

$$\begin{array}{l r|l} \text{round } 0 & b_0:\top & C_0,1 \\ & b_1:\bot & E_0,0 \\ \hline \text{round } 1 & b_0:\bot & C_1,0 \\ & b_1:\bot & \\ \hline & \vdots & \end{array} \xrightarrow{\text{Inc}} \begin{array}{r|l} \top & C_0,1 \\ \bot & A_0,1 \\ \hline \bot & C_1,0 \\ \bot & \\ \hline \vdots & \end{array} \xrightarrow{\text{write}_{b_0}(\top)} \begin{array}{r|l} \top & D_0,1 \\ \bot & A_0,1 \\ \hline \top & C_1,0 \\ \bot & \\ \hline \vdots & \end{array} \xrightarrow{\text{read}^{-1}_{b_1}(\bot)}$$

Semantics of shared-memory models with rounds



Concrete configuration stores
values of shared registers, and for each process its local state and round

Example with $n = 3$

$$\begin{array}{lr|l} \text{round 0} & b_0 : \top & C_0, 1 \\ & b_1 : \bot & E_0, 0 \\ \hline \text{round 1} & b_0 : \bot & C_1, 0 \\ & b_1 : \bot & \\ \hline & \vdots & \end{array} \xrightarrow{\text{Inc}} \begin{array}{r|l} \top & C_0, 1 \\ \bot & A_0, 1 \\ \hline \bot & C_1, 0 \\ \bot & \\ \hline \vdots & \end{array} \xrightarrow{\text{write}_{b_0}(\top)} \begin{array}{r|l} \top & D_0, 1 \\ \bot & A_0, 1 \\ \hline \top & C_1, 0 \\ \bot & \\ \hline \vdots & \end{array} \xrightarrow{\text{read}_{b_1}^{-1}(\bot)} \begin{array}{r|l} \top & R_0, 1 \\ \bot & A_0, 1 \\ \hline \top & C_1, 0 \\ \bot & \\ \hline \vdots & \end{array}$$

Model checking shared-memory protocols with rounds

[**B.** Markey Sankur Waldburger, submitted]

The parameterized model checking of **safety properties** for shared-memory protocols with rounds is **PSPACE-complete**.

Model checking shared-memory protocols with rounds

[**B.** Markey Sankur Waldburger, submitted]

The parameterized model checking of **safety properties** for shared-memory protocols with rounds is **PSPACE-complete**.

Objective: prove $\forall n,\ \forall \gamma_0 \in \Gamma_{\mathsf{init}}(n),\ \forall \gamma_0 \xrightarrow{*} \gamma :\ q_{\mathsf{err}} \notin \gamma$

Model checking shared-memory protocols with rounds

[**B.** Markey Sankur Waldburger, submitted]

The parameterized model checking of **safety properties** for shared-memory protocols with rounds is **PSPACE-complete**.

Objective: prove $\forall n,\ \forall \gamma_0 \in \Gamma_{\mathsf{init}}(n),\ \forall \gamma_0 \xrightarrow{*} \gamma :\ q_{\mathsf{err}} \notin \gamma$

dually, look for a counterexample: $\exists n,\ \exists \gamma_0 \in \Gamma_{\mathsf{init}}(n),\ \exists \gamma_0 \xrightarrow{*} \gamma :\ q_{\mathsf{err}} \in \gamma$

Model checking shared-memory protocols with rounds

[**B.** Markey Sankur Waldburger, submitted]

The parameterized model checking of **safety properties** for shared-memory protocols with rounds is **PSPACE-complete**.

Objective: prove $\forall n,\ \forall \gamma_0 \in \Gamma_{\mathsf{init}}(n),\ \forall \gamma_0 \xrightarrow{*} \gamma :\ q_{\mathsf{err}} \notin \gamma$

dually, look for a counterexample: $\exists n,\ \exists \gamma_0 \in \Gamma_{\mathsf{init}}(n),\ \exists \gamma_0 \xrightarrow{*} \gamma :\ q_{\mathsf{err}} \in \gamma$

Challenges: exponential lower bounds everywhere!

- minimum round at which q_{err} is reached;
- number of processes needed to reach q_{err};
- number of required active rounds on executions reaching q_{err}

all may be **exponential** in the protocol size

Exploiting monotonicity

Copycat property on states and written values

- if a state can be populated by a process, it can be populated by an arbitrary number of them;
- if a value can be written to a register once, it can be written arbitrarily many times

Exploiting monotonicity

Copycat property on states and written values

- if a state can be populated by a process, it can be populated by an arbitrary number of them;
- if a value can be written to a register once, it can be written arbitrarily many times

Abstract configuration stores
which states are populated, and which registers have been written to

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$A_0, 1$
round 1	$b_0 : \top$	$C_1, 0$
	$b_1 : \bot$	$C_1, 0$
		$A_0, 1$
	$\vdots$	

full config.

$\mathbf{reg}(b_0, 0)$	$C_0, 1$
$\mathbf{reg}(b_0, 1)$	$A_0, 1$
	$C_1, 0$

abstract config.

Exploiting monotonicity

Copycat property on states and written values

- if a state can be populated by a process, it can be populated by an arbitrary number of them;
- if a value can be written to a register once, it can be written arbitrarily many times

Abstract configuration stores
which states are populated, and which registers have been written to

round 0	$b_0 : \top$	$C_0, 1$
	$b_1 : \bot$	$A_0, 1$
round 1	$b_0 : \top$	$C_1, 0$
	$b_1 : \bot$	$C_1, 0$
		$A_0, 1$
	$\vdots$	

full config.

$\mathbf{reg}(b_0, 0)$	$C_0, 1$
$\mathbf{reg}(b_0, 1)$	$A_0, 1$
	$C_1, 0$

abstract config.

⚠ Limited monotonicity: two reachable states may not be mutually reachable

Checking parameterized safety in PSPACE

Proof high-level idea:

- guess a feasible sequence of moves leading to an error state

$\langle A_0 \xrightarrow{\mathsf{read}^0_{b_0}(\bot)} B_0, \mathbf{0}\rangle$ $\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{0}\rangle$ $\langle B_0 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} C_0, \mathbf{1}\rangle$ $\langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{0}\rangle$

$\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle$ $\langle B_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} C_0, \mathbf{2}\rangle$ $\langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{2}\rangle$ $\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle$

$\langle A_1 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} B_1, \mathbf{0}\rangle$ $\langle C_1 \xrightarrow{\mathsf{write}_{b_1}(\top)} D_1, \mathbf{0}\rangle$ $\langle D_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} R_0, \mathbf{2}\rangle$

- while maintaining abstract configuration

Checking parameterized safety in PSPACE

Proof high-level idea: **iteratively on rounds**

- guess a feasible sequence of moves leading to an error state

$\langle A_0 \xrightarrow{\mathsf{read}^0_{b_0}(\bot)} B_0, \mathbf{0}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{0}\rangle \quad \langle B_0 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} C_0, \mathbf{1}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{0}\rangle$

$\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle \quad \langle B_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} C_0, \mathbf{2}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{2}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle$

$\langle A_1 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} B_1, \mathbf{0}\rangle \quad \langle C_1 \xrightarrow{\mathsf{write}_{b_1}(\top)} D_1, \mathbf{0}\rangle \quad \langle D_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} R_0, \mathbf{2}\rangle$

- while maintaining abstract configuration

Checking parameterized safety in PSPACE

Proof high-level idea: **iteratively on rounds**

- guess a feasible sequence of moves leading to an error state

$\langle A_0 \xrightarrow{\mathsf{read}^0_{b_0}(\bot)} B_0, \mathbf{0}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{0}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{0}\rangle$

$\langle A_1 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} B_1, \mathbf{0}\rangle \quad \langle C_1 \xrightarrow{\mathsf{write}_{b_1}(\top)} D_1, \mathbf{0}\rangle$

- while maintaining abstract configuration

Checking parameterized safety in PSPACE

Proof high-level idea: **iteratively on rounds**

- guess a feasible sequence of moves leading to an error state

$\langle A_0 \xrightarrow{\mathrm{read}^0_{b_0}(\bot)} B_0, \mathbf{0}\rangle \quad \langle E_0 \xrightarrow{\mathrm{Inc}} A_0, \mathbf{0}\rangle \quad \langle B_0 \xrightarrow{\mathrm{read}^0_{b_1}(\bot)} C_0, \mathbf{1}\rangle \quad \langle C_0 \xrightarrow{\mathrm{write}_{b_0}(\top)} D_0, \mathbf{0}\rangle$

$\langle E_0 \xrightarrow{\mathrm{Inc}} A_0, \mathbf{1}\rangle \qquad \langle E_0 \xrightarrow{\mathrm{Inc}} A_0, \mathbf{1}\rangle$

$\langle A_1 \xrightarrow{\mathrm{read}^0_{b_1}(\bot)} B_1, \mathbf{0}\rangle \quad \langle C_1 \xrightarrow{\mathrm{write}_{b_1}(\top)} D_1, \mathbf{0}\rangle$

- while maintaining abstract configuration

Checking parameterized safety in PSPACE

Proof high-level idea: **iterativelyon rounds**

- guess a feasible sequence of moves leading to an error state

$\langle A_0 \xrightarrow{\text{read}^0_{b_0}(\perp)} B_0, \mathbf{0}\rangle \quad \langle E_0 \xrightarrow{\text{Inc}} A_0, \mathbf{0}\rangle \quad \langle B_0 \xrightarrow{\text{read}^0_{b_1}(\perp)} C_0, \mathbf{1}\rangle \quad \langle C_0 \xrightarrow{\text{write}_{b_0}(\top)} D_0, \mathbf{0}\rangle$

$\langle E_0 \xrightarrow{\text{Inc}} A_0, \mathbf{1}\rangle \quad \langle B_0 \xrightarrow{\text{read}^{-1}_{b_1}(\perp)} C_0, \mathbf{2}\rangle \quad \langle C_0 \xrightarrow{\text{write}_{b_0}(\top)} D_0, \mathbf{2}\rangle \quad \langle E_0 \xrightarrow{\text{Inc}} A_0, \mathbf{1}\rangle$

$\langle A_1 \xrightarrow{\text{read}^0_{b_1}(\perp)} B_1, \mathbf{0}\rangle \quad \langle C_1 \xrightarrow{\text{write}_{b_1}(\top)} D_1, \mathbf{0}\rangle \quad \langle D_0 \xrightarrow{\text{read}^{-1}_{b_1}(\perp)} R_0, \mathbf{2}\rangle$

- while maintaining abstract configuration

Checking parameterized safety in PSPACE

Proof high-level idea: **iteratively on rounds**

- guess a feasible sequence of moves leading to an error state

$\langle A_0 \xrightarrow{\mathsf{read}^0_{b_0}(\bot)} B_0, \mathbf{0}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{0}\rangle \quad \langle B_0 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} C_0, \mathbf{1}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{0}\rangle$

$\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle \quad \langle B_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} C_0, \mathbf{2}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{2}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle$

$\langle A_1 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} B_1, \mathbf{0}\rangle \quad \langle C_1 \xrightarrow{\mathsf{write}_{b_1}(\top)} D_1, \mathbf{0}\rangle \quad \langle D_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} R_0, \mathbf{2}\rangle$

- while maintaining abstract configuration

Polynomial space suffices thanks to visibility window $\mathbf{w}$

- information propagation when inserting moves of round k and forgetting moves of rounds $k - \mathbf{w} - 1$

Checking parameterized safety in PSPACE

Proof high-level idea: **iteratively on rounds**

- guess a feasible sequence of moves leading to an error state

$$\langle B_0 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} C_0, \mathbf{1}\rangle$$

$$\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle \quad \langle B_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} C_0, \mathbf{2}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{2}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle$$

$$\langle D_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} R_0, \mathbf{2}\rangle$$

- while maintaining abstract configuration

Polynomial space suffices thanks to visibility window **w**

- information propagation when inserting moves of round k and forgetting moves of rounds $k - \mathbf{w} - 1$

Checking parameterized safety in PSPACE

Proof high-level idea: **iteratively on rounds**

- guess a feasible sequence of moves leading to an error state

$$\langle B_0 \xrightarrow{\mathsf{read}^0_{b_1}(\bot)} C_0, \mathbf{1}\rangle$$

$$\langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle \quad \langle B_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} C_0, \mathbf{2}\rangle \quad \langle C_0 \xrightarrow{\mathsf{write}_{b_0}(\top)} D_0, \mathbf{2}\rangle \quad \langle E_0 \xrightarrow{\mathsf{Inc}} A_0, \mathbf{1}\rangle$$

$$\langle D_0 \xrightarrow{\mathsf{read}^{-1}_{b_1}(\bot)} R_0, \mathbf{2}\rangle$$

- while maintaining abstract configuration

Polynomial space suffices thanks to visibility window **w**

- information propagation when inserting moves of round k and forgetting moves of rounds $k - \mathbf{w} - 1$

applies to **prove safety** (agreement and validity) of Aspnes' algorithm

Summary

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation

Summary

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation

- This talk: **round-based** algorithms
  1. fault-tolerant broadcast algorithms
     [**B.** Thomas Widder Concur'21]
     - layered threshold automata
     - undecidable in general
     - predicate abstraction: incomplete yet refinable analysis
  2. shared-memory algorithms
     [**B.** Markey Sankur Waldburger, submitted]
     - shared-registers automata
     - safety verification is PSPACE-complete
     - exponential cutoff, minimal covering length, and drift

Other parameterized verification frameworks for distributed algorithms

- threshold automata [Konnov Lazić Veith Widder POPL'17]
- broadcast protocols [Esparza Finkel Mayr LICS'99] [Delzanno Sangnier Zavattaro Concur'10]
- global sync. protocols [Jaber Jacobs Wagner Kulkarni Samanta CAV'20]
- shared-memory models [Esparza Ganty Majumdar JACM 2016] [Bouyer Markey Randour Sangnier Stan ICALP'16]
- token-passing algorithms on lines/rings [Lin Rümmer CAV'16]
- population protocols [Esparza Ganty Leroux Majumdar Acta Inf. 2017]
- synchronous algorithms on rings [Aiswarya Bollig Gastin I&C 2018]

Special thanks to

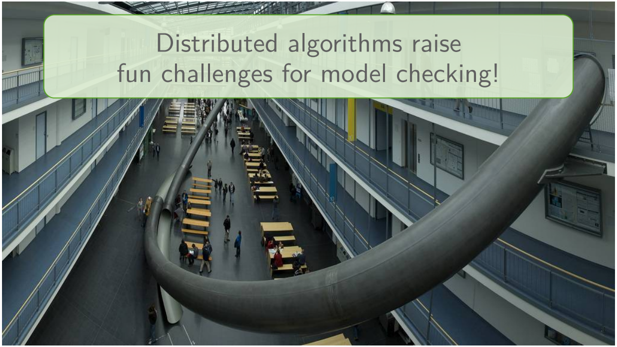
Arnaud Sangnier

Josef Widder

Special thanks to

Arnaud Sangnier

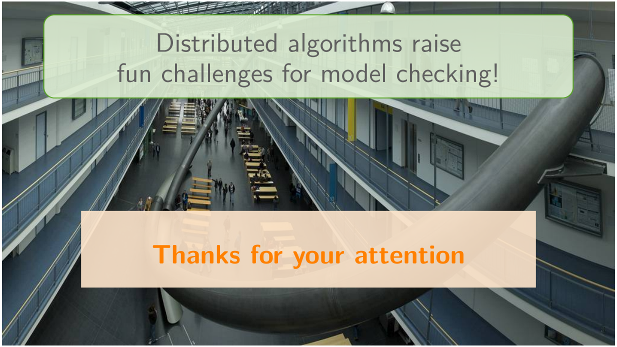
Josef Widder



Distributed algorithms raise
fun challenges for model checking!

Special thanks to

Arnaud Sangnier   Josef Widder



Distributed algorithms raise
fun challenges for model checking!

Thanks for your attention