Parameterized safety verification of round-based shared-memory systems

Nicolas Waldburger [1]
Nathalie Bertrand [1], Nicolas Markey [1], Ocan Sankur [1]

[1]Univ Rennes, Inria, CNRS, IRISA, France

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**
- **Asynchrony**: some processes might be faster than others

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**
- **Asynchrony**: some processes might be faster than others
- **Non-atomic** read & write combinations, no fault

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**
- **Asynchrony**: some processes might be faster than others
- **Non-atomic** read & write combinations, no fault
- **Round-based**: Fresh copy of registers at each round, processes can be on different rounds

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**
- **Asynchrony**: some processes might be faster than others
- **Non-atomic** read & write combinations, no fault
- **Round-based**: Fresh copy of registers at each round, processes can be on different rounds

The binary consensus problem

Make all processes agree on a common value, each process having an initial preference p.
Desired properties of consensus algorithms:

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**
- **Asynchrony**: some processes might be faster than others
- **Non-atomic** read & write combinations, no fault
- **Round-based**: Fresh copy of registers at each round, processes can be on different rounds

The binary consensus problem

Make all processes agree on a common value, each process having an initial preference p.
Desired properties of consensus algorithms:

Validity : If a process decides value p, some process started with preference p.

Agreement : Two processes that decide decide of the same value.

Termination : All processes eventually decide of a value.

Round-based shared-memory algorithms

The distributed systems considered

- **Parallel**, **identical** processes communicating via **shared memory**
- **Asynchrony**: some processes might be faster than others
- **Non-atomic** read & write combinations, no fault
- **Round-based**: Fresh copy of registers at each round, processes can be on different rounds

The binary consensus problem

Make all processes agree on a common value, each process having an initial preference p.
Desired properties of consensus algorithms:

Validity : If a process decides value p, some process started with preference p.

Agreement : Two processes that decide decide of the same value.

Termination : All processes eventually decide of a value.

Consensus with shared memory is difficult: there is no wait-free consensus protocol with shared memory and two processes.

A motivating example: Aspnes' consensus algorithm

int $k := 0$, bool $p \in \{0, 1\}$, $(\mathrm{rg}_b[r])_{b\in\{0,1\}, r\in\mathbb{N}}$ all initialized to no;
while true **do**
  read from $\mathrm{rg}_0[k]$ and $\mathrm{rg}_1[k]$;
  if $\mathrm{rg}_0[k] =$ yes and $\mathrm{rg}_1[k] =$ no **then** $p := 0$;
  else if $\mathrm{rg}_0[k] =$ no and $\mathrm{rg}_1[k] =$ yes **then** $p := 1$;
  write yes to $\mathrm{rg}_p[k]$;
  if $k > 0$ **then**
    read from $\mathrm{rg}_{1-p}[k-1]$;
    if $\mathrm{rg}_{1-p}[k-1] =$ no **then** return p;
  $k := k+1$;

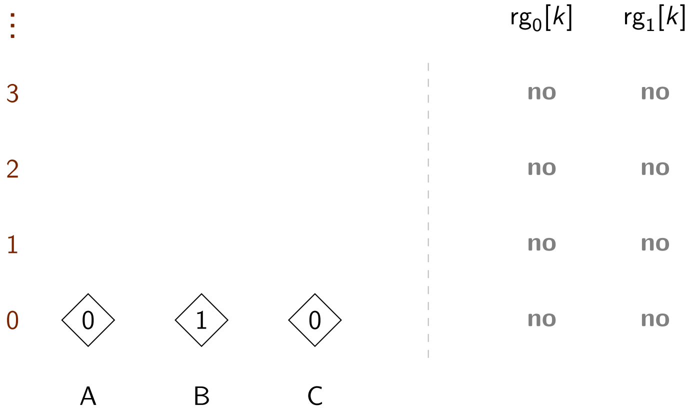
read from registers of rounds k and $k-1$

write to registers of round k

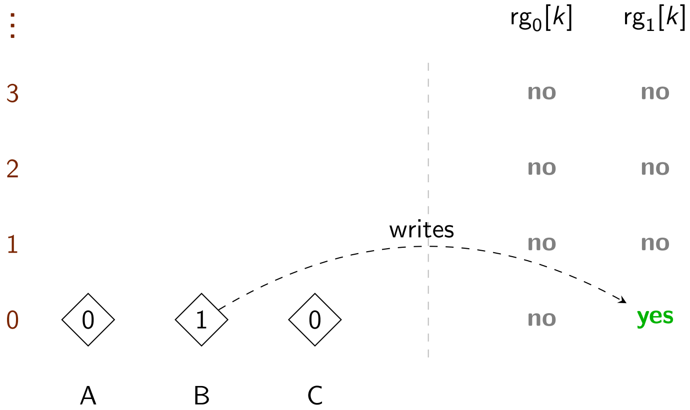
Algorithm 1: Aspnes' consensus algorithm[1].

[1] James Aspnes, Fast deterministic consensus in a noisy environment, *Journal of Algorithms*, 2002.

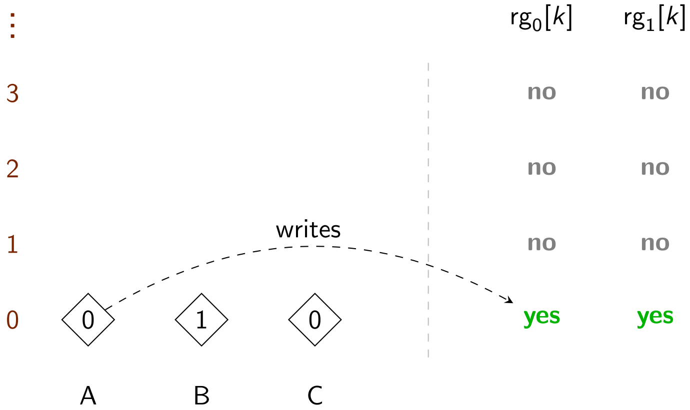
An example of execution of Aspnes' consensus algorithm



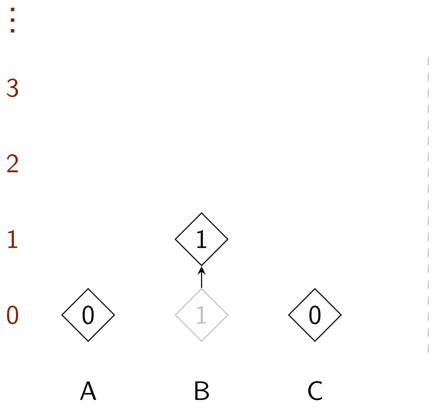
An example of execution of Aspnes' consensus algorithm



An example of execution of Aspnes' consensus algorithm

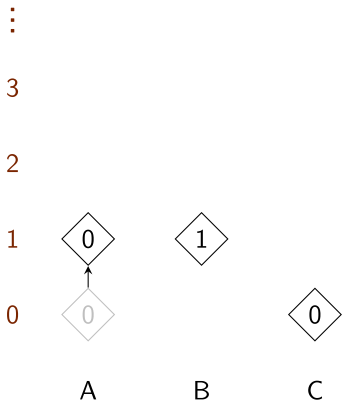


An example of execution of Aspnes' consensus algorithm



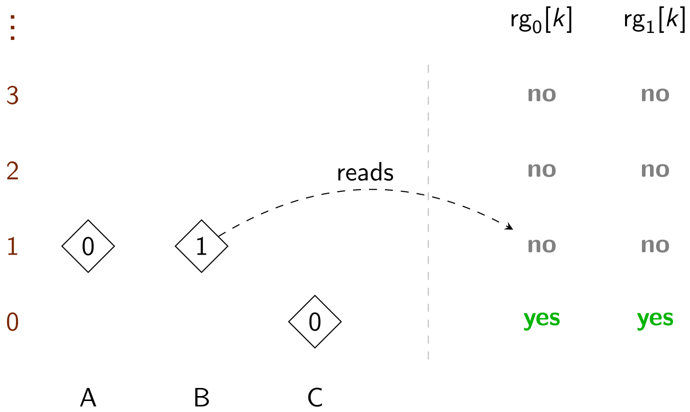
$rg_0[k]$	$rg_1[k]$
no	**no**
no	**no**
no	**no**
yes	**yes**

An example of execution of Aspnes' consensus algorithm

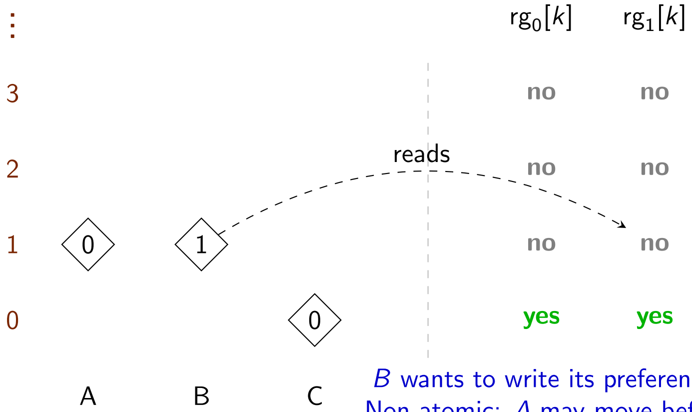


k	$rg_0[k]$	$rg_1[k]$
3	no	no
2	no	no
1	no	no
0	yes	yes

An example of execution of Aspnes' consensus algorithm



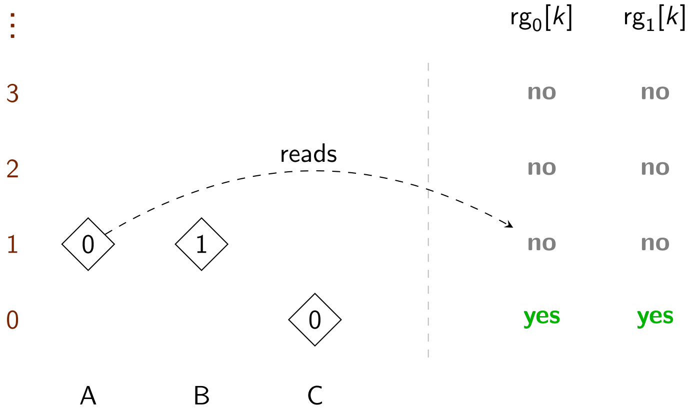
An example of execution of Aspnes' consensus algorithm



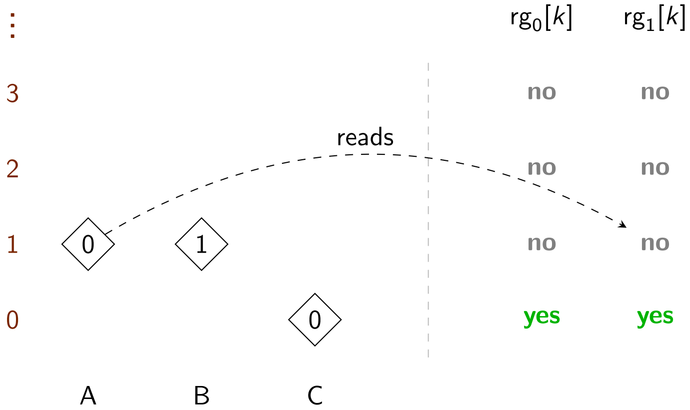
B wants to write its preference on $rg_1[k]$

Non-atomic: A may move before B writes

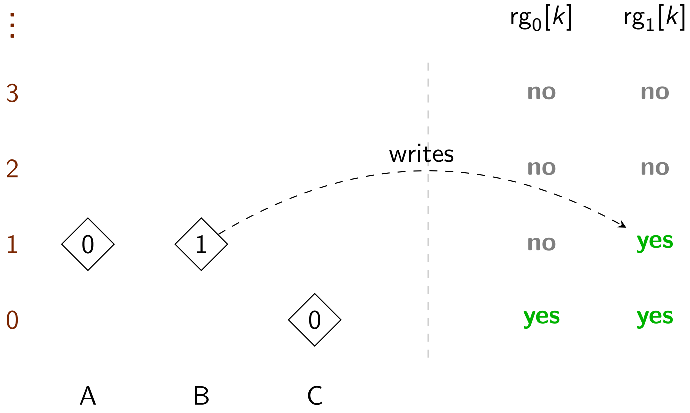
An example of execution of Aspnes' consensus algorithm



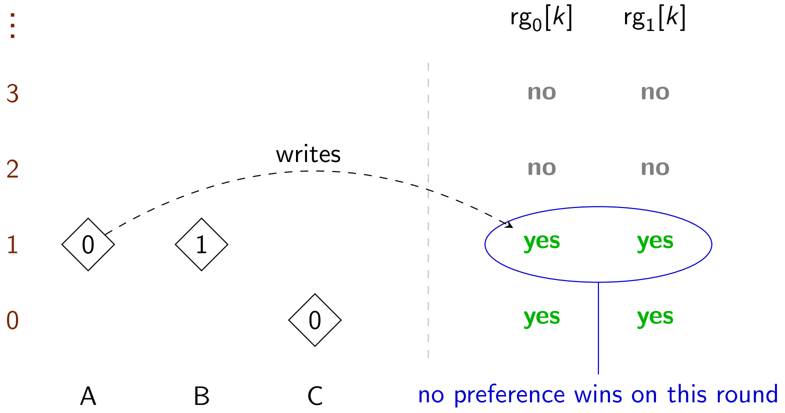
An example of execution of Aspnes' consensus algorithm



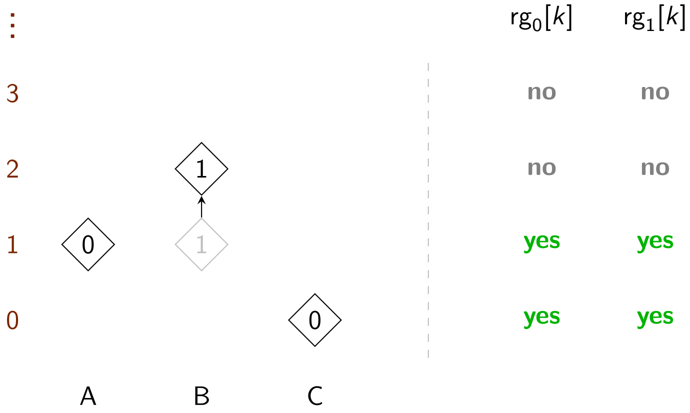
An example of execution of Aspnes' consensus algorithm



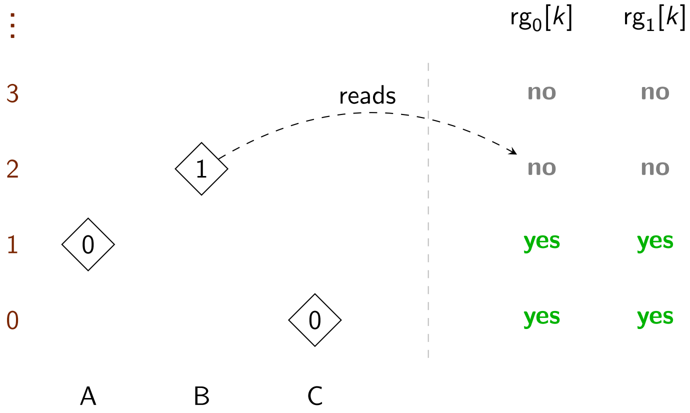
An example of execution of Aspnes' consensus algorithm



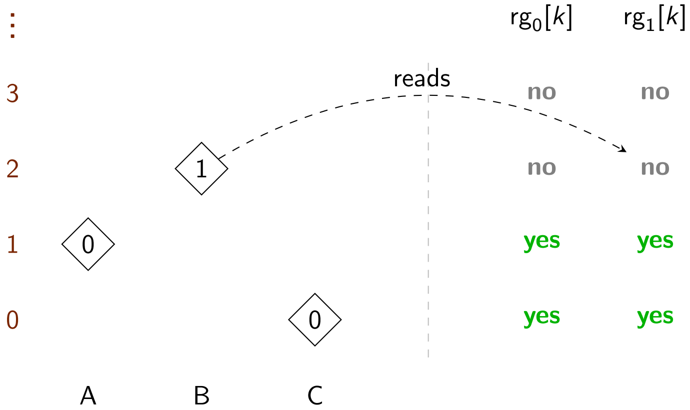
An example of execution of Aspnes' consensus algorithm



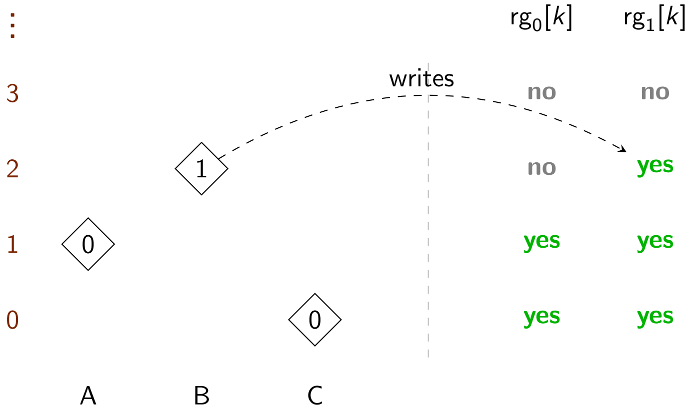
An example of execution of Aspnes' consensus algorithm



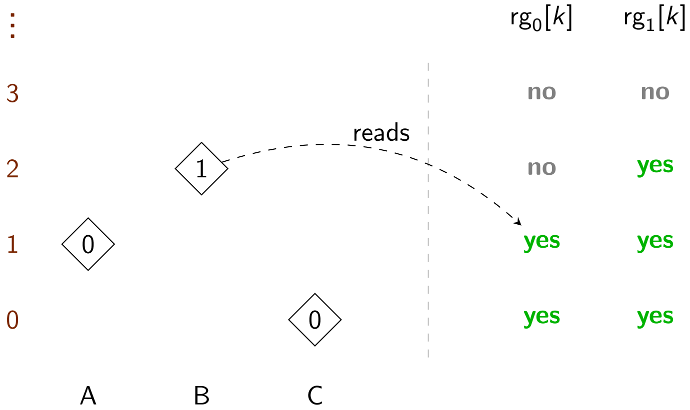
An example of execution of Aspnes' consensus algorithm



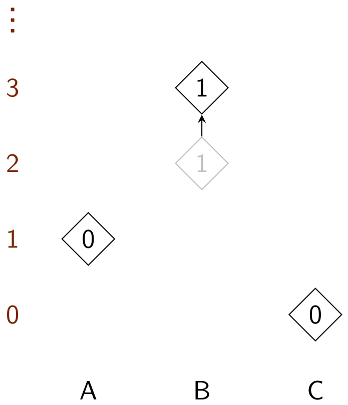
An example of execution of Aspnes' consensus algorithm



An example of execution of Aspnes' consensus algorithm

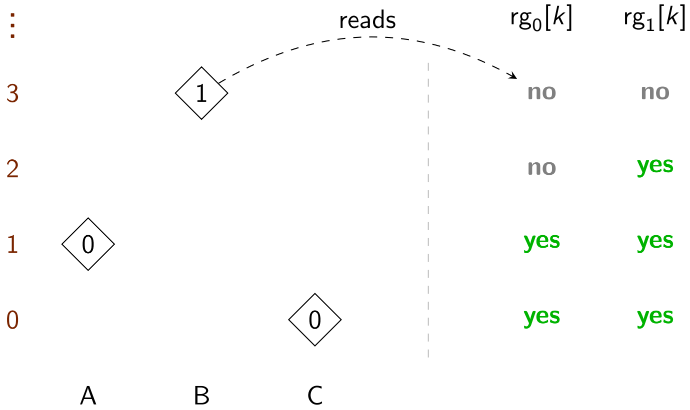


An example of execution of Aspnes' consensus algorithm

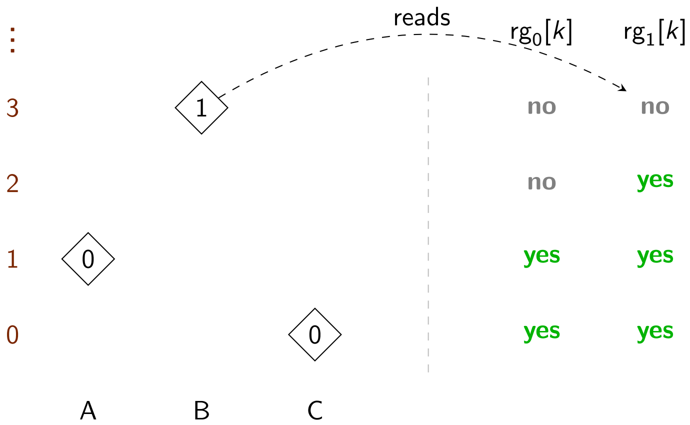


k	$\mathrm{rg}_0[k]$	$\mathrm{rg}_1[k]$
3	no	no
2	no	yes
1	yes	yes
0	yes	yes

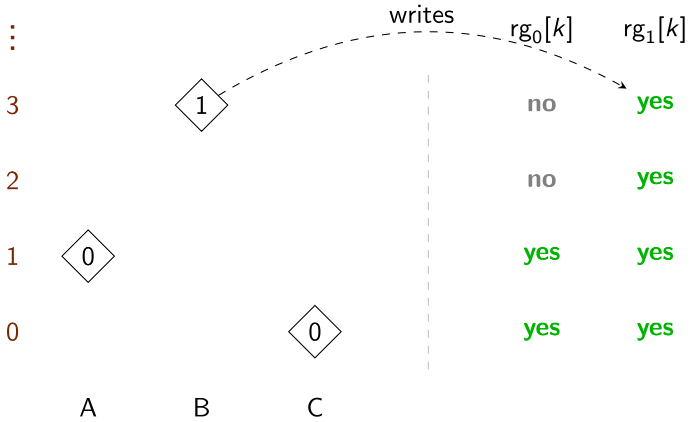
An example of execution of Aspnes' consensus algorithm



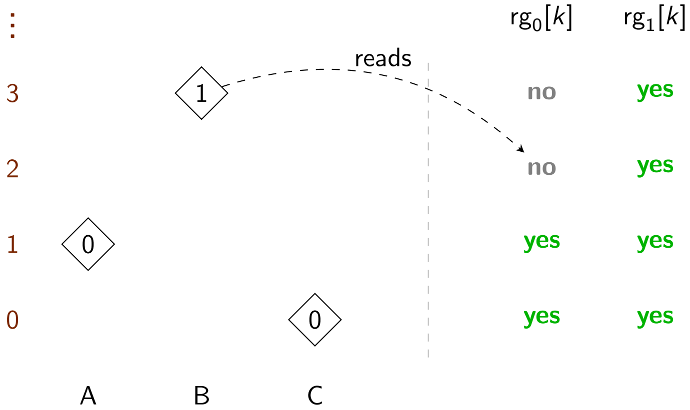
An example of execution of Aspnes' consensus algorithm



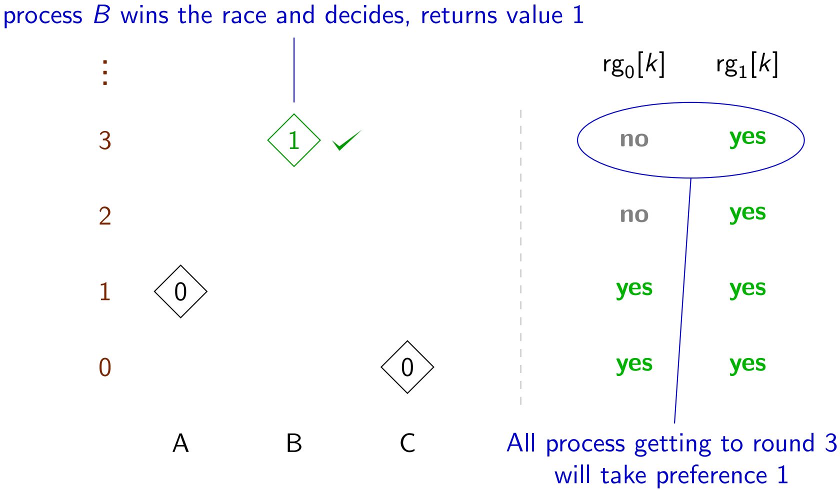
An example of execution of Aspnes' consensus algorithm



An example of execution of Aspnes' consensus algorithm



An example of execution of Aspnes' consensus algorithm



A model: round-based register protocols

Inspired by models for shared-memory systems without rounds[2,3].

---

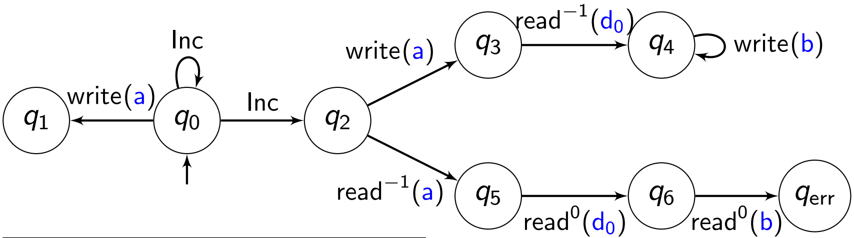
[2] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *CAV'13*

[3] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *ICALP'16*

A model: round-based register protocols

Inspired by models for shared-memory systems without rounds[2][3].

- One model for all processes: a finite automaton



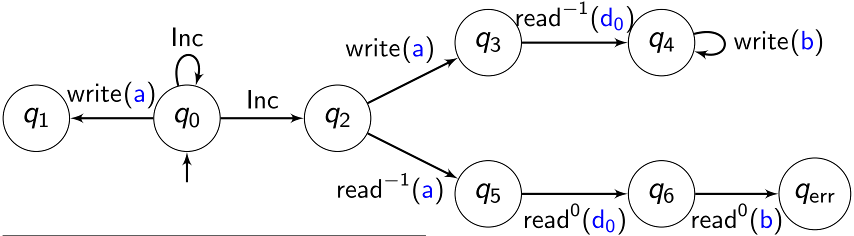
[2] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *CAV'13*

[3] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *ICALP'16*

A model: round-based register protocols

Inspired by models for shared-memory systems without rounds[2,3].

- One model for all processes: a finite automaton
- Transitions are read actions, write actions and round increments



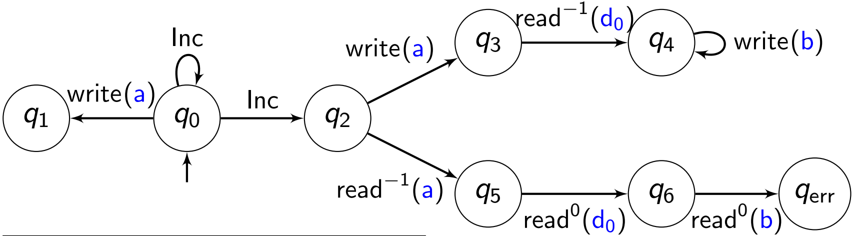
[2] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *CAV'13*

[3] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *ICALP'16*

A model: round-based register protocols

Inspired by models for shared-memory systems without rounds[2][3].

- One model for all processes: a finite automaton
- Transitions are read actions, write actions and round increments
- Processes can be on different rounds, the round number of a process may never decrease



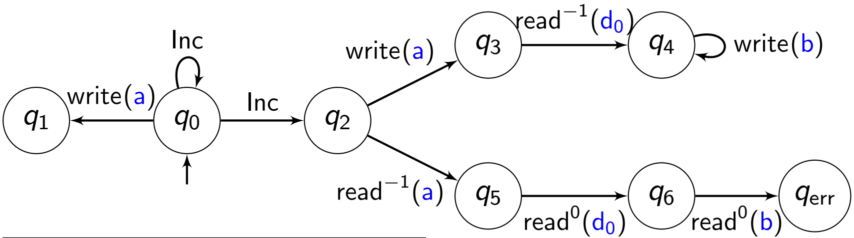
[2] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *CAV'13*

[3] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *ICALP'16*

A model: round-based register protocols

Inspired by models for shared-memory systems without rounds[2,3].

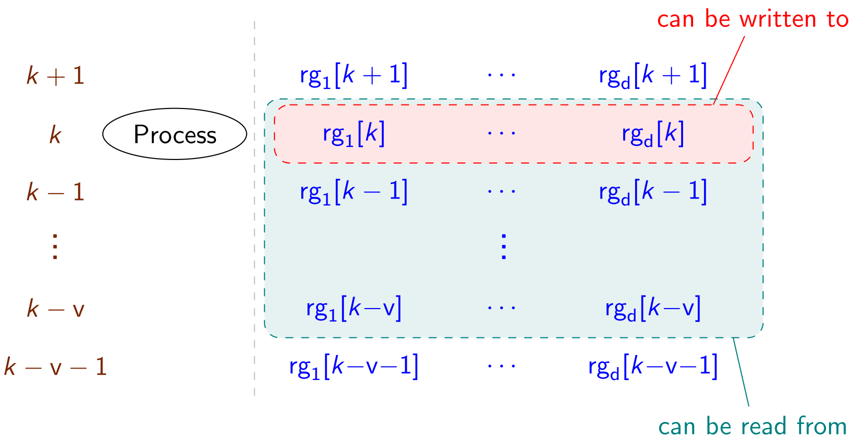
- One model for all processes: a finite automaton
- Transitions are read actions, write actions and round increments
- Processes can be on different rounds, the round number of a process may never decrease
- A fixed number d of registers per round (the total number of registers is hence unbounded)



[2] Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. *CAV'13*

[3] Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. *ICALP'16*

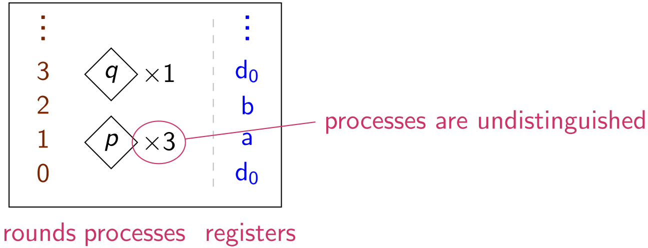
A limited visilibity range



v given in unary

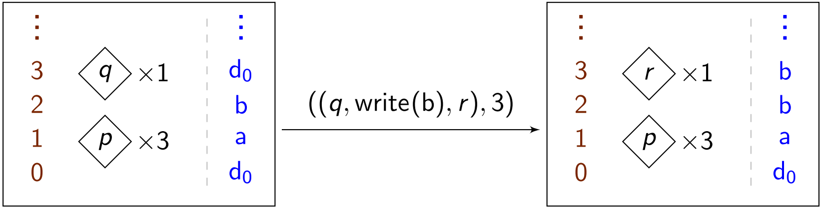
Semantics of the model

From now on, let $d = 1$: one register per round.



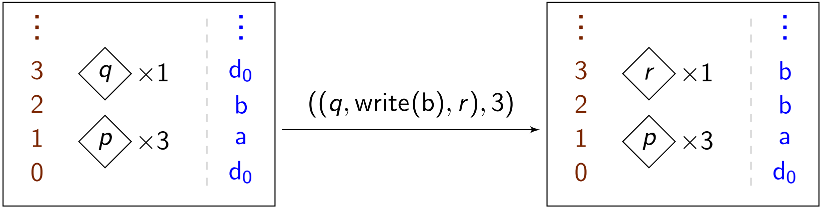
Semantics of the model

From now on, let $\mathrm{d} = 1$: one register per round.

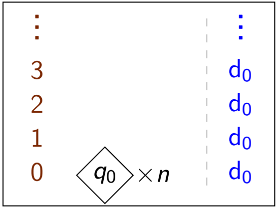


Semantics of the model

From now on, let d = 1: one register per round.



Initial configuration of size n:



The safety problem

The (parameterized) safety problem

Is it true that, for all numbers of processes n and all executions from the initial configuration of size n, an error state q_{err} is avoided?

The safety problem

The (parameterized) safety problem

Is it true that, for all numbers of processes n and all executions from the initial configuration of size n, an error state q_{err} is avoided?

Dual problem: look for an execution *covering* the error.

The safety problem

The (parameterized) safety problem

Is it true that, for all numbers of processes n and all executions from the initial configuration of size n, an error state q_{err} is avoided?

Dual problem: look for an execution *covering* the error.

If the error state cannot be covered, the system is safe.

The safety problem

The (parameterized) safety problem

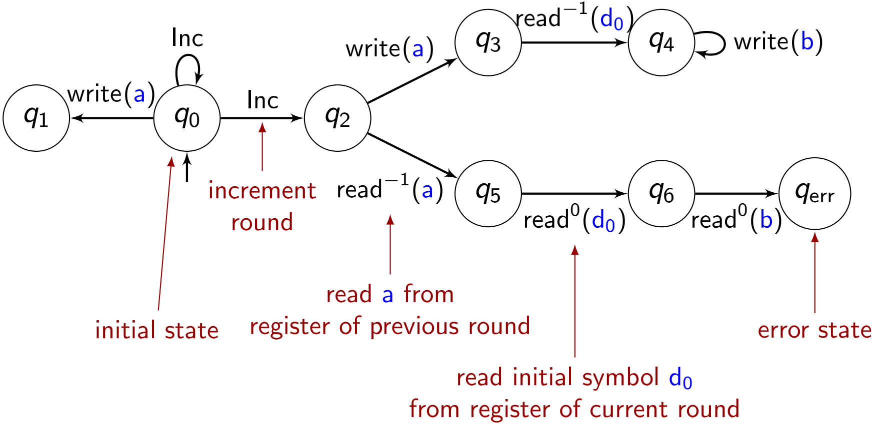
Is it true that, for all numbers of processes n and all executions from the initial configuration of size n, an error state q_{err} is avoided?

Dual problem: look for an execution *covering* the error.

If the error state cannot be covered, the system is safe.

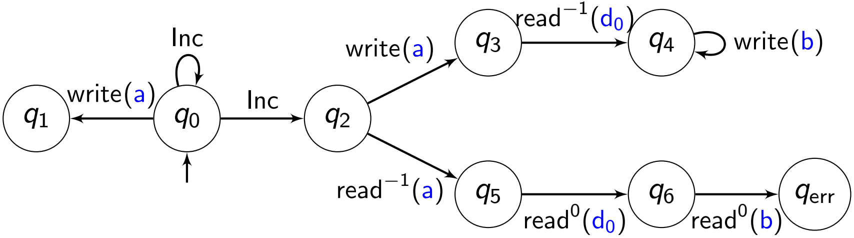
Agreement and Validity of Aspnes' consensus algorithm can be encoded as safety properties.

A small example

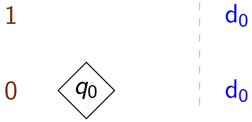


v = 1 (processes can read one round back)

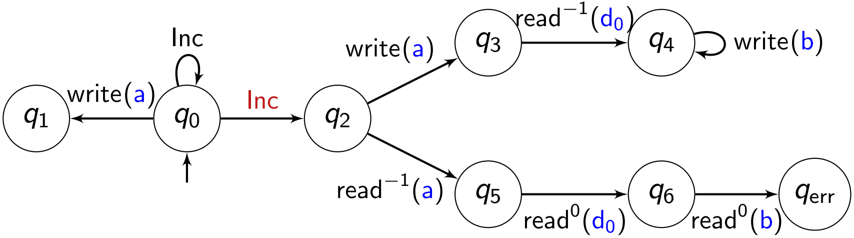
A small example



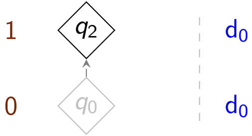
State q_4 can be covered from the initial configuration with one process:



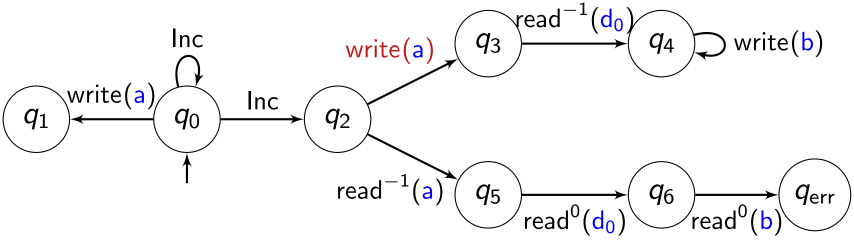
A small example



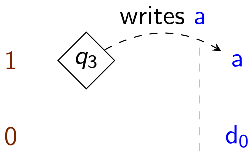
State q_4 can be covered from the initial configuration with one process:



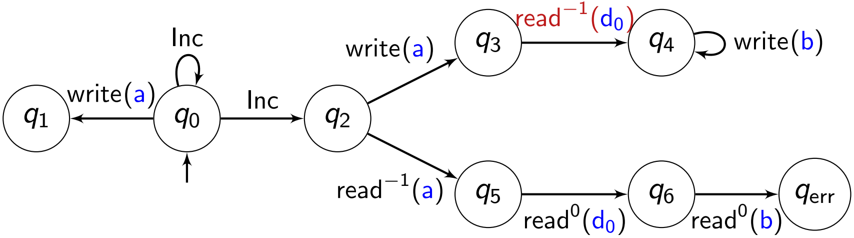
A small example



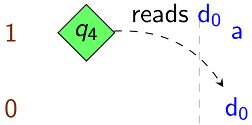
State q_4 can be covered from the initial configuration with one process:



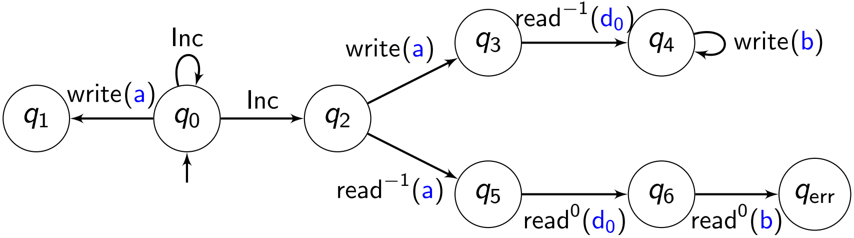
A small example



State q_4 can be covered from the initial configuration with one process:

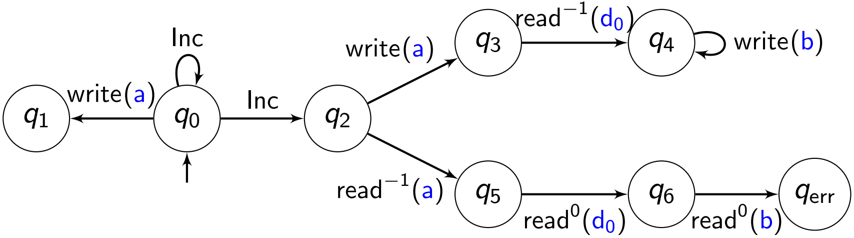


A small example

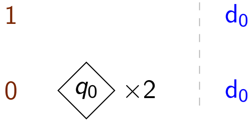


State q_6 can be covered from the initial configuration with two processes:

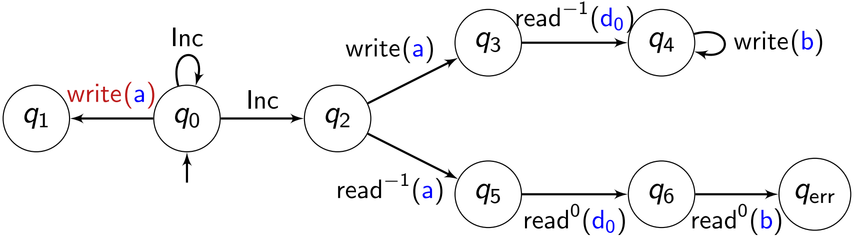
A small example



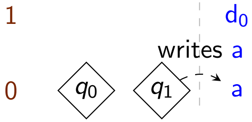
State q_6 can be covered from the initial configuration with two processes:



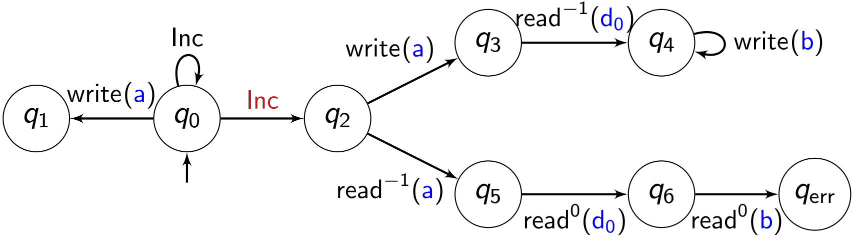
A small example



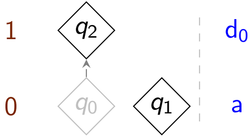
State q_6 can be covered from the initial configuration with two processes:



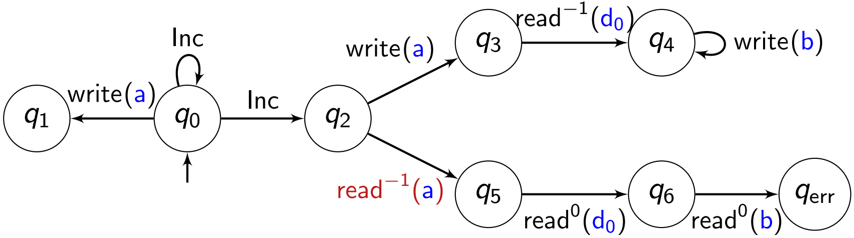
A small example



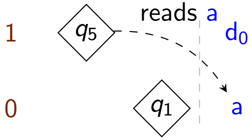
State q_6 can be covered from the initial configuration with two processes:



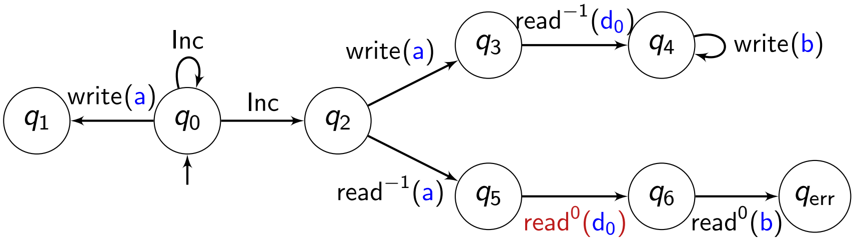
A small example



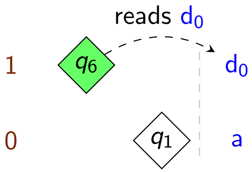
State q_6 can be covered from the initial configuration with two processes:



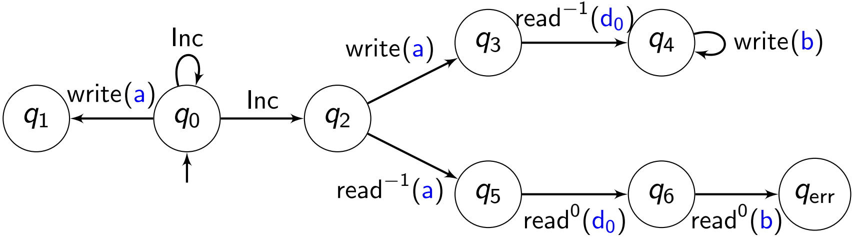
A small example



State q_6 can be covered from the initial configuration with two processes:

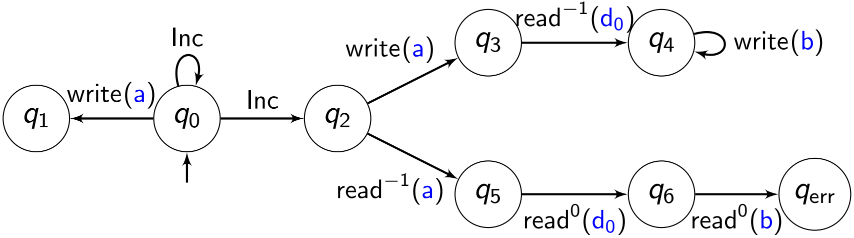


A small example



Claim: the system is safe.

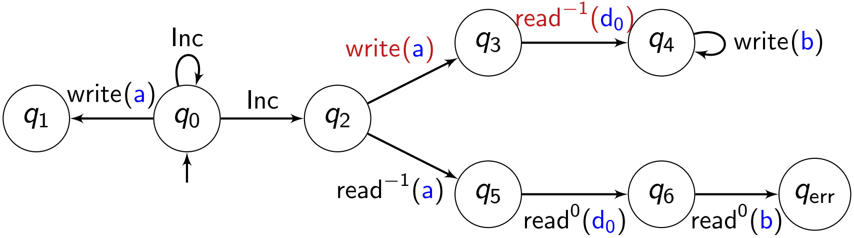
A small example



Claim: the system is safe.

Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. But:

A small example

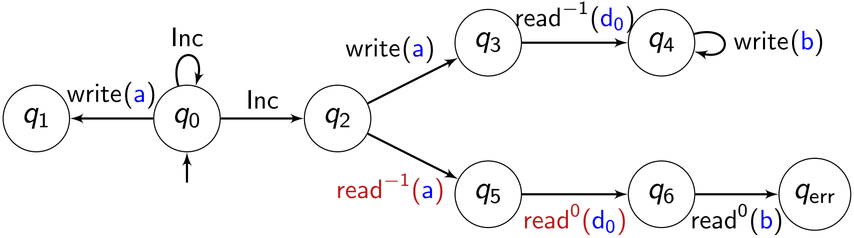


Claim: the system is safe.

Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. But:

- To cover (q_4, k), one must write to rg$[k]$ while rg$[k-1]$ still has value d_0;

A small example

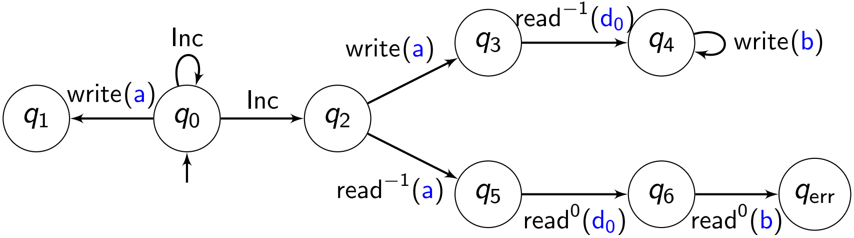


Claim: the system is safe.

Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. But:

- To cover (q_4, k), one must write to $\mathrm{rg}[k]$ while $\mathrm{rg}[k-1]$ still has value d_0;
- To cover (q_6, k), one must write to $\mathrm{rg}[k-1]$ while $\mathrm{rg}[k]$ still has value d_0.

A small example



Claim: the system is safe.

Observe that q_{err} can be covered if and only if, for some round k, (q_4, k) and (q_6, k) can be covered in the same execution. But:

- To cover (q_4, k), one must write to $\mathrm{rg}[k]$ while $\mathrm{rg}[k-1]$ still has value d_0;
- To cover (q_6, k), one must write to $\mathrm{rg}[k-1]$ while $\mathrm{rg}[k]$ still has value d_0.

This is the only source of “incompatibility”!

Main contribution

Parameterized safety in round-based register protocols is PSPACE-complete.

Lower bounds

Exponential lower bounds

In order to reach an error state, one might need at least:

- An exponential number of processes,

Lower bounds

Exponential lower bounds

In order to reach an error state, one might need at least:

- An exponential number of processes,
- spreading across an exponential number of rounds at the same time.

Lower bounds

Exponential lower bounds

In order to reach an error state, one might need at least:

- An exponential number of processes,
- spreading across an exponential number of rounds at the same time.

Theorem

The safety problem is PSPACE-hard.

By reduction from Quantified Boolean Formula.

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)
- Thanks to copycat, define an abstraction where one only remembers which pairs (state,round) are populated by at least one process

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)
- Thanks to copycat, define an abstraction where one only remembers which pairs (state,round) are populated by at least one process
- Exploit limited visibility range: reads and writes are local with respect to the round

PSPACE-membership

Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.

The execution cannot be guessed move by move in polynomial space: too many relevant rounds at the same time!

Ingredients of the algorithm

- Copycat property (thanks to non-atomicity)
- Thanks to copycat, define an abstraction where one only remembers which pairs (state,round) are populated by at least one process
- Exploit limited visibility range: reads and writes are local with respect to the round
- Rely on a sliding window along the rounds

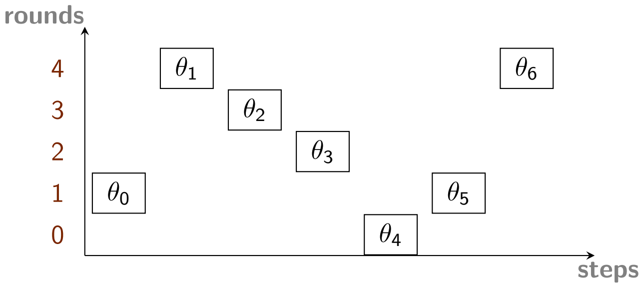
A visual display for executions

Execution: $\sigma_0 \xrightarrow{\theta_0} \sigma_1 \xrightarrow{\theta_1} \sigma_2 \xrightarrow{\theta_2} \sigma_3 \xrightarrow{\theta_3} \sigma_4 \xrightarrow{\theta_4} \sigma_5 \xrightarrow{\theta_5} \sigma_6 \xrightarrow{\theta_6} \sigma_7$

$\Downarrow$

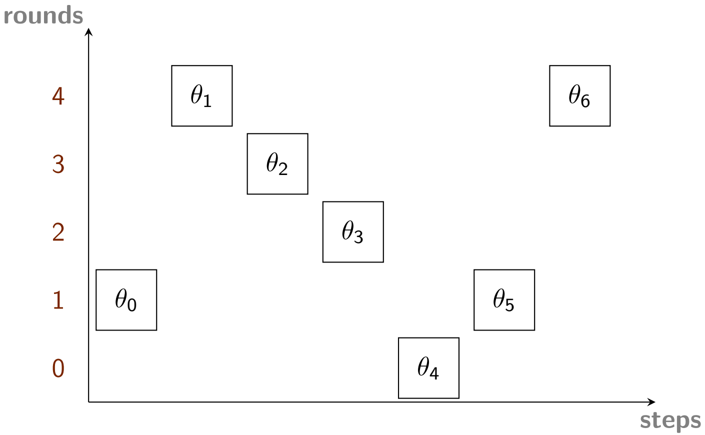
moves:	$\boxed{\theta_0}$	$\boxed{\theta_1}$	$\boxed{\theta_2}$	$\boxed{\theta_3}$	$\boxed{\theta_4}$	$\boxed{\theta_5}$	$\boxed{\theta_6}$
rounds:	1	4	3	2	0	1	4

$\Downarrow$



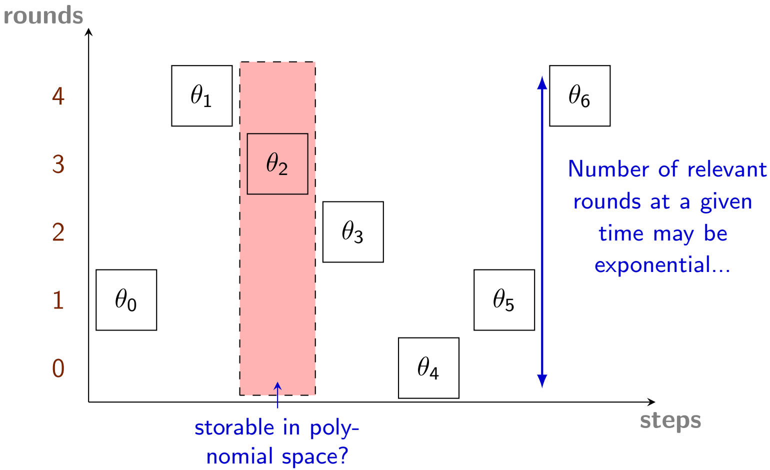
The sliding window

Here v = 1: processes at round k can read from rounds k and $k-1$



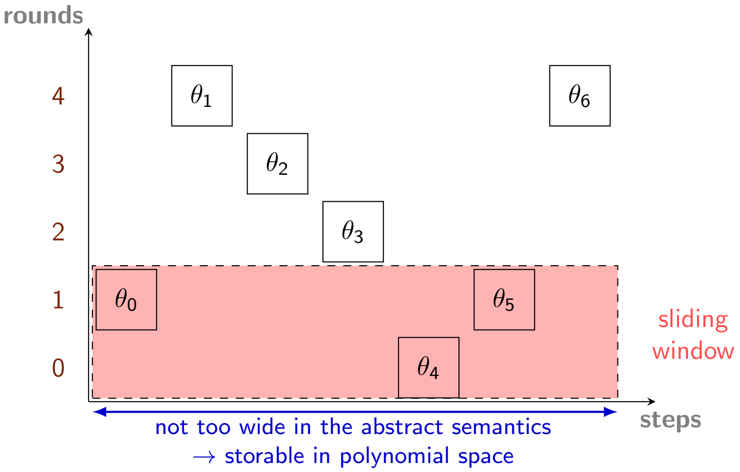
The sliding window

Intuitive idea of proceeding move by move is not working:



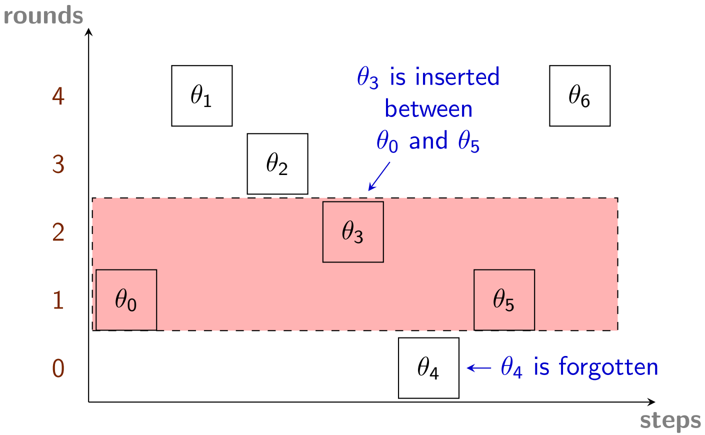
The sliding window

Instead: sliding window along the rounds non-deterministically guessing the execution



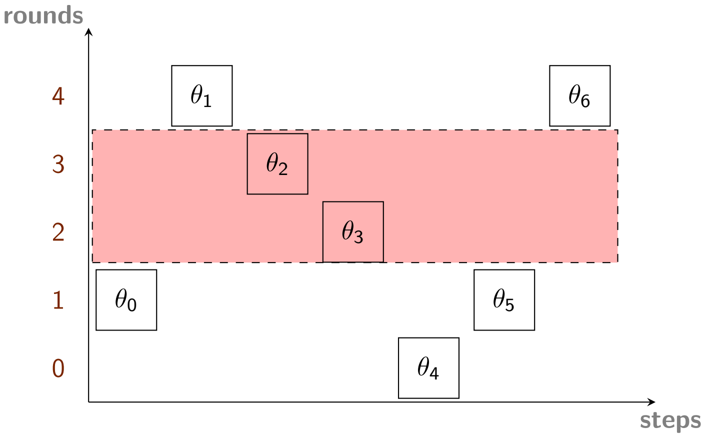
The sliding window

Checking that a move is valid only depends on what happens locally.



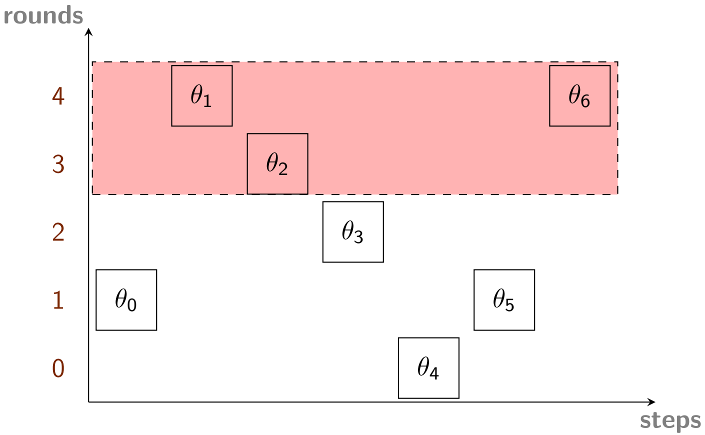
The sliding window

And so on...



The sliding window

And so on...



Exponential upper bounds

Termination of the safety algorithm

The algorithm returns that the system is not safe if a local configuration reached contains q_{err}.

Exponential upper bounds

Termination of the safety algorithm

The algorithm returns that the system is not safe if a local configuration reached contains q_{err}.

After an exponential number of iterations, the information has looped and the algorithm stops.

Exponential upper bounds

Termination of the safety algorithm

The algorithm returns that the system is not safe if a local configuration reached contains q_{err}.

After an exponential number of iterations, the information has looped and the algorithm stops.

From the algorithm, we derive exponential upper bounds matching the lower bounds:

Exponential upper bounds

Termination of the safety algorithm

The algorithm returns that the system is not safe if a local configuration reached contains q_{err}.

After an exponential number of iterations, the information has looped and the algorithm stops.

From the algorithm, we derive exponential upper bounds matching the lower bounds:

Exponential upper bound on cutoff

There exists an exponential upper bound on the number of processes needed to cover q_{err}.

Exponential upper bounds

Termination of the safety algorithm

The algorithm returns that the system is not safe if a local configuration reached contains q_{err}.

After an exponential number of iterations, the information has looped and the algorithm stops.

From the algorithm, we derive exponential upper bounds matching the lower bounds:

Exponential upper bound on cutoff

There exists an exponential upper bound on the number of processes needed to cover q_{err}.

Exponential upper bound on the number of rounds

There exists an exponential upper bound on the number of rounds needed to cover q_{err}.

Conclusion

Summary

- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm

Conclusion

Summary

- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm
- Parameterized safety is PSPACE-complete

Conclusion

Summary

- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm
- Parameterized safety is PSPACE-complete
- The poly-space algorithm relies on a sliding window along the rounds

Conclusion

Summary

- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm
- Parameterized safety is PSPACE-complete
- The poly-space algorithm relies on a sliding window along the rounds

Future work

- Other problems on our model: parameterized TARGET & INEVITABILITY

Conclusion

Summary

- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm
- Parameterized safety is PSPACE-complete
- The poly-space algorithm relies on a sliding window along the rounds

Future work

- Other problems on our model: parameterized TARGET & INEVITABILITY
- Almost-sure reachability in round-based register protocols with stochastic schedulers (termination of Aspnes' algorithm)

Conclusion

Summary

- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm
- Parameterized safety is PSPACE-complete
- The poly-space algorithm relies on a sliding window along the rounds

Future work

- Other problems on our model: parameterized TARGET & INEVITABILITY
- Almost-sure reachability in round-based register protocols with stochastic schedulers (termination of Aspnes' algorithm)
- Links with classical notions of fairness

Conclusion

Summary

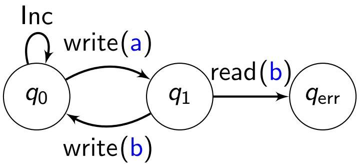
- Round-based register protocols are a model for round-based shared-memory algorithms such as Aspnes' consensus algorithm
- Parameterized safety is PSPACE-complete
- The poly-space algorithm relies on a sliding window along the rounds

Future work

- Other problems on our model: parameterized TARGET & INEVITABILITY
- Almost-sure reachability in round-based register protocols with stochastic schedulers (termination of Aspnes' algorithm)
- Links with classical notions of fairness

Thank you!

Classical notions of fairness are not satisfactory



q_{err} is reached with probability 1 with a stochastic scheduler with two processes.

Consider the execution with two processes where one process goes to q_1 and back to q_0 on every round, while the other process stays on q_0 forever.

This execution is fair with respect to:

- Fairness on moves: no move is available infinitely often because k increases
- Fairness on transitions: transition from q_1 to q_{err} is never enabled.