Parameterized safety verification of round-based shared-memory systems

Nicolas Waldburger
Nathalie Bertrand, Nicolas Markey, Ocan Sankur

1Univ Rennes, Inria, CNRS, IRISA, France

RP22, 17th October 2022
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 considered 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:
### 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.
A motivating example: Aspnes’ consensus algorithm

int $k := 0$, bool $p \in \{0, 1\}$, $(rg_b[r])_{b \in \{0,1\}, r \in \mathbb{N}}$ all initialized to no;
while true do
    read from $rg_0[k]$ and $rg_1[k]$
    if $rg_0[k] = \text{yes}$ and $rg_1[k] = \text{no}$ then $p := 0$
    else if $rg_0[k] = \text{no}$ and $rg_1[k] = \text{yes}$ then $p := 1$
    write yes to $rg_p[k]$
    if $k > 0$ then
        read from $rg_{1-p}[k-1]$
        if $rg_{1-p}[k-1] = \text{no}$ then return $p$
        $k := k+1$

Algorithm 1: Aspnes’ consensus algorithm\(^1\).

---

An example of execution of Aspnes’ consensus algorithm

\begin{align*}
&\begin{array}{ccc}
0 & 1 & 0 \\
A & B & C \\
\end{array} & \begin{array}{ll}
rg_0[k] & rg_1[k] \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\end{array}
\end{align*}

All processes getting to round 3 will take preference 1.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Round</th>
<th>Process</th>
<th>( \text{rg}_0[k] )</th>
<th>( \text{rg}_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Round</th>
<th>Process</th>
<th>rg&lt;sub&gt;0&lt;/sub&gt;[k]</th>
<th>rg&lt;sub&gt;1&lt;/sub&gt;[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
### An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process A</th>
<th>Process B</th>
<th>Process C</th>
<th>$rg_0[k]$</th>
<th>$rg_1[k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>$rg_0[k]$</th>
<th>$rg_1[k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

process A wins the race
All processes getting to round 3 will take preference 1
An example of execution of Aspnes’ consensus algorithm

\[
\begin{array}{c|cc}
0 & no & yes \\
1 & no & yes \\
2 & no & no \\
3 & no & no \\
\end{array}
\]
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

B wants to write on $rg_1[k]$
Non-atomic:
A may move before B writes

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

All processes getting to round 3 will take preference 1

Non-atomic:
A may move before B writes
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>( rg_0[k] )</th>
<th>( rg_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>B</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Process B wins the race.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>rg_0[k]</th>
<th>rg_1[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

0 1 0
A B C

reads

All process getting to round 3 will take preference 1.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( r_{g_0}[k] )</th>
<th>( r_{g_1}[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

A    B    C
\[\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
\end{array}\]
\[\begin{array}{c}
rg_0[k] \\
gr_1[k] \\
no \\
o \\
\end{array}\]
\[\begin{array}{c}
writes \\
reads \\
no \\
no \\
\end{array}\]
\[\begin{array}{c}
writes \\
reads \\
no \\
no \\
\end{array}\]
\[\begin{array}{c}
writes \\
reads \\
no \\
no \\
\end{array}\]
\[\begin{array}{c}
writes \\
reads \\
no \\
no \\
\end{array}\]

no preference wins on this round
### An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Round</th>
<th>Process 0</th>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

- **rg\(_0\)[k]**  
  - ROUND 0: Process 0 (A)  
  - ROUND 1: Process 1 (B)  
  - ROUND 2: Process 0 (A)  
  - ROUND 3: Process 1 (B)

- **rg\(_1\)[k]**  
  - ROUND 0: Process 0 (A)  
  - ROUND 1: Process 1 (B)  
  - ROUND 2: Process 0 (A)  
  - ROUND 3: Process 1 (B)
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cc}
\text{reads} & \text{writes} \\
\text{no} & \text{writes} \\
\text{no} & \text{reads} \\
\text{yes} & \text{reads} \\
\text{yes} & \text{writes} \\
\text{no} & \text{reads} \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\end{array}
\]

\[
\begin{array}{cc}
\text{rg}_0[k] & \text{rg}_1[k] \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{yes} & \text{yes} \\
\text{yes} & \text{yes} \\
\end{array}
\]
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>( \text{rg}_0[k] )</th>
<th>( \text{rg}_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

A process wins the race if all processes get to round 3.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \begin{array}{cccc}
\begin{array}{c}
\text{writes} \\
\text{no} \\
\text{no} \\
\text{yes} \\
\text{yes} \\
\text{yes} \\
\end{array}
& \begin{array}{c}
\text{rg}_0[k] \\
\text{no} \\
\text{no} \\
\text{yes} \\
\text{yes} \\
\text{yes} \\
\end{array}
& \begin{array}{c}
\text{rg}_1[k] \\
\text{no} \\
\text{yes} \\
\text{yes} \\
\text{yes} \\
\text{yes} \\
\end{array}
\end{array} \]
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rg₀[k]</th>
<th>rg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>
| 0      |   |   | yes
| 1      |   | yes | yes
| 2      | no | yes | yes
| 3      | no | no  | yes

rg₀[k]    | yes | yes |
rg₁[k]    | yes | yes |
An example of execution of Aspnes’ consensus algorithm

A
B
C

reads

\[ \text{rg}_0[k] \]
\[ \text{rg}_1[k] \]

1

no
no

2

no
yes

1

yes
yes

0

yes
yes

A

B

C

process B wins the race

All processes getting to round 3 will take preference 1.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>rg_0[k]</th>
<th>rg_1[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>B</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

All processes agree on a value in round 3.
An example of execution of Aspnes’ consensus algorithm

 writes  \( \text{rg}_0[k] \)  \( \text{rg}_1[k] \)

0  yes  yes
1  yes  yes
2  yes  yes
3  no  yes

process B wins the race
All process getting to round 3 will take preference 1
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( \text{rg}_0[k] )</th>
<th>( \text{rg}_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

- Process B wins the race
- All processes getting to round 3 will take preference 1

\begin{itemize}
  \item \textbf{rg}_0[k] = \text{no}
  \item \textbf{rg}_1[k] = \text{yes}
\end{itemize}
A model: round-based register protocols

Inspired by models for shared-memory systems without rounds\(^2\).
A model: round-based register protocols

Inspired by models for shared-memory systems without rounds\textsuperscript{23}. 
• One model for all processes: a finite automaton

---

\textsuperscript{2} Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. \textit{CAV’13}

\textsuperscript{3} Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. \textit{ICALP’16}
A model: round-based register protocols

Inspired by models for shared-memory systems without rounds\textsuperscript{23}.

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

---

\textsuperscript{2} Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. \textit{CAV’13}

\textsuperscript{3} Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. Reachability in networks of register protocols under stochastic schedulers. \textit{ICALP’16}
A model: round-based register protocols

Inspired by models for shared-memory systems without rounds\(^2\).

- 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 limited visibility range

\[ k + 1 \]
\[ k \]
\[ k - 1 \]
\[ \vdots \]
\[ k - v \]
\[ k - v - 1 \]

\[ \text{can be written to} \]
\[ \text{can be read from} \]

\[ v \text{ given in unary} \]
Semantics of the model

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$p \times 3$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$q \times 1$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

processes are undistinguished

rounds processes   registers
Semantics of the model

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

<table>
<thead>
<tr>
<th></th>
<th>$q$ $\times 1$</th>
<th>$d_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$p$ $\times 3$</td>
<td>$a$</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>$d_0$</td>
</tr>
</tbody>
</table>

$((q, \text{write}(b), r), 3)$

<table>
<thead>
<tr>
<th></th>
<th>$r$ $\times 1$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$p$ $\times 3$</td>
<td>$a$</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>$d_0$</td>
</tr>
</tbody>
</table>
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 considered algorithms
Our model
The safety problem
Results
Conclusion

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 $q_{\text{err}}$ is avoided?

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

\[ q_0 \xrightarrow{\text{Inc}} q_1 \xrightarrow{\text{write}(a)} q_2 \xrightarrow{\text{Inc}} \]

\[ q_3 \xrightarrow{\text{write}(a)} q_4 \xrightarrow{\text{write}(b)} \]

\[ q_5 \xrightarrow{\text{read}^{-1}(a)} q_6 \xrightarrow{\text{read}^{0}(d_0)} q_{\text{err}} \xrightarrow{\text{read}^{0}(b)} \]

Initial state: \( q_0 \)
Increment round: \( q_2 \)
Read initial symbol \( d_0 \): \( q_4 \)

\[ v = 1 \] (processes can read one round back)
A small example

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

\[
\begin{align*}
1 & \quad d_0 \\
0 & \quad q_0, d_0
\end{align*}
\]
A small example

State $q_4$ can be covered from the initial configuration with one process:
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:
State $q_6$ can be covered from the initial configuration with two processes:

1

\[ \begin{align*}
& 1 & d_0 \\
& 0 & q_0 \times 2 & d_0
\end{align*} \]
A small example

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

1. $d_0$
2. Writes a
3. $q_0$
4. $q_1$
5. a
A small example

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

1. \( (q_0, 0) \rightarrow (q_1, a) \rightarrow (q_2, d_0) \rightarrow (q_3, write(a)) \rightarrow (q_4, read^{-1}(d_0)) \rightarrow (q_5, read^{-1}(a)) \rightarrow (q_6, read^0(d_0)) \rightarrow (q_{\text{err}}, read^0(b)) \rightarrow (q_{\text{err}}, q_{\text{err}}) \)

2. \( (q_0, 1) \rightarrow (q_2, d_0) \rightarrow (q_3, write(a)) \rightarrow (q_4, read^{-1}(d_0)) \rightarrow (q_5, read^{-1}(a)) \rightarrow (q_6, read^0(d_0)) \rightarrow (q_{\text{err}}, read^0(b)) \rightarrow (q_{\text{err}}, q_{\text{err}}) \)
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 $rg[k]$ while $rg[k-1]$ still has value $d_0$;
- To cover $(q_6, k)$, one must write to $rg[k-1]$ while $rg[k]$ still has value $d_0$. 
Claim: the system is safe.

Observe that $q_{\text{err}}$ 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 $d_0$;
- To cover $(q_6, k)$, one must write to $rg[k-1]$ while $rg[k]$ still has value $d_0$.

This is the only source of “incompatibility”!
Main contribution

Parameterized safety in round-based register protocols is PSPACE-complete\(^4\).

\(^4\) Nathalie Bertrand, Nicolas Markey, Ocan Sankur, W. Parameterized safety verification of round-based shared-memory systems. ICALP’22
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 \textit{PSPACE-hard}.

By reduction from Quantified Boolean Formula.
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)
The considered algorithms
Our model
The safety problem
Results
Conclusion

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
The considered algorithms
Our model
The safety problem
Results
Conclusion

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

moves: \[\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\]

rounds: \[1, 4, 3, 2, 0, 1, 4\]
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:

Number of relevant rounds at a given time may be exponential...

storable in polynomial space?
Instead: sliding window along the rounds non-deterministically guessing the execution

not too wide in the abstract semantics

→ storable in polynomial space
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...

Number of relevant rounds at a given time may be exponential... storable in polynomial space. In the abstract semantics, the sliding window of \( \theta_4 \) is forgotten, and \( \theta_3 \) is inserted between \( \theta_0 \) and \( \theta_5 \).
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
- Generalisation to other reachability problems (e.g. TARGET)
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

- Generalisation to other reachability problems (e.g. TARGET)
- Almost-sure reachability/cube reachability in round-based register protocols (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

- Generalisation to other reachability problems (e.g. TARGET)
- Almost-sure reachability/cube reachability in round-based register protocols (termination of Aspnes’ algorithm)
- Weak memory
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
- Generalisation to other reachability problems (e.g. TARGET)
- Almost-sure reachability/cube reachability in round-based register protocols (termination of Aspnes’ algorithm)
- Weak memory

Thank you!
Classical notions of fairness are not satisfactory

$q_{\text{err}}$ 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 $q_{\text{err}}$ is never enabled.