Parameterized safety verification of round-based shared-memory systems

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

¹Univ 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 considered algorithms

Our model

The safety problem

Results

Conclusion

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
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 $p \in \{0, 1\}$, $(rg_b[r])_{b \in \{0, 1\}, r \in \mathbb{N}}$ all initialized to no;

\[
\text{while} \quad \text{true} \quad \text{do} \\
\quad \text{read from } rg_0[k] \text{ and } rg_1[k] \; ; \\
\quad \text{if } rg_0[k] = \text{yes and } rg_1[k] = \text{no then } p := 0; \\
\quad \text{else if } rg_0[k] = \text{no and } rg_1[k] = \text{yes then } p := 1; \\
\quad \text{write yes to } rg_p[k] \; ; \\
\quad \text{if } k > 0 \text{ then} \\
\quad \quad \text{read from } rg_{1-p}[k-1] \; , \\
\quad \quad \text{if } rg_{1-p}[k-1] = \text{no then return } p; \\
\quad k := k+1; \\
\]

\textbf{Algorithm 1:} Aspnes’ consensus algorithm$^1$.

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>0</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Round</th>
<th>Process</th>
<th>writes</th>
<th>( rg_0[k] )</th>
<th>( rg_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>B</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>C</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

\( \ast \): Initialization

\( [k] \): Request phase

A process getting to round 3 will take preference 1
An example of execution of Aspnes' consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>B</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

\[ rg_0[k] \quad rg_1[k] \]

- no
- yes
- writes

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>( r_{g0}[k] )</th>
<th>( r_{g1}[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

A wins the race and decides, returns value 1.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Step</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>A</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

A wins the race and decides, returns value 1

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

<table>
<thead>
<tr>
<th></th>
<th>rg₀[k]</th>
<th>rg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Process B wins the race and decides, returns value 1.
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

<table>
<thead>
<tr>
<th>Process</th>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>B</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>C</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>

All processes reaching round 3 with preference 1.
An example of execution of Aspnes’ consensus algorithm

The considered algorithms
Our model
The safety problem
Results
Conclusion

An example of execution of Aspnes’ consensus algorithm

A  B  C

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

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

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>rg0[k]</th>
<th>rg1[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

All processes 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

rg0[k]  | no | no |
rg1[k]  | no | no |
reads   | yes| yes|
writes  |    |    |
races
All processes getting to round 3 will take preference 1
```

A wins the race and decides, returns value 1

No preference wins on this round

reads
reads
reads
reads
writes
writes
reads
reads
reads
writes
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>Value</th>
<th>rg₀[k]</th>
<th>rg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Process B wins the race and decides, returns value 1.
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>rg₀[k]</th>
<th>rg₁[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>

Process B wins the race and decides, returns value 1. 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, yes</td>
<td>yes</td>
</tr>
<tr>
<td>( C )</td>
<td>yes, yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
An example of execution of Aspnes’ consensus algorithm

<table>
<thead>
<tr>
<th>Process</th>
<th>Writes</th>
<th>Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

At round 3, process 1 writes 1, process 0 reads 1, and process 2 reads 1. Process 1 wins the race and decides, returning value 1. 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>Reads</th>
<th>$\text{rg}_0[k]$</th>
<th>$\text{rg}_1[k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>B</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>A</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

A wins the race and decides, returns value 1.

All processes 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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

The considered algorithms

Our model

The safety problem

Results

Conclusion
An example of execution of Aspnes’ consensus algorithm

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

Process A writes a 0 to memory. Process C reads 0. Process B reads the value of memory and decides on 1. Now processes A and B read the memories and decide on 1. Now all processes go to round 3.
An example of execution of Aspnes’ consensus algorithm

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

process B wins the race and decides, returns value 1

All process getting to round 3 will take preference 1
A model: round-based register protocols

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

\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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{model_diagram}
\end{figure}

\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\(^{23}\).

- 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\textsuperscript{23}.

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

\begin{itemize}
  \item $q_0$
  \item $q_1$
  \item $q_2$
  \item $q_3$
  \item $q_4$
  \item $q_5$
  \item $q_6$
  \item $q_{err}$
\end{itemize}

\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
- 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 visibility range

\begin{align*}
k + 1 & \quad \cdots \quad k + 1 \\
k & \quad \text{Process} \\
k - 1 & \quad \cdots \quad k - 1 \\
\vdots & \quad \cdots \quad \vdots \\
k - v & \quad \cdots \quad k - v \\
k - v - 1 & \quad \cdots \quad k - v - 1
\end{align*}

rg\_1[k + 1] \quad \cdots \quad rg\_d[k + 1]
\begin{array}{ccc}
rg\_1[k] & \cdots & rg\_d[k] \\
rg\_1[k - 1] & \cdots & rg\_d[k - 1] \\
\vdots & & \vdots \\
rg\_1[k - v] & \cdots & rg\_d[k - v] \\
rg\_1[k - v - 1] & \cdots & rg\_d[k - v - 1]
\end{array}

can be written to

can be read from

v given in \textit{unary}
From now on, let $d = 1$: one register per round.

\[
\begin{array}{c|c|c|c}
3 & q \times 1 & d_0 \\
2 & b \\
1 & p \times 3 & a \\
0 & d_0 \\
\end{array}
\]

processes are undistinguished

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

\[
\begin{array}{c}
\vdots \\
3 & q \times 1 & d_0 \\
2 & b \\
1 & p \times 3 & a \\
0 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
\vdots \\
3 & r \times 1 & b \\
2 & b \\
1 & p \times 3 & a \\
0 & d_0 \\
\end{array}
\]

$((q, \text{write}(b), r), 3)$
Semantics of the model

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

Initial configuration of size $n$: 

$\begin{array}{c}
q \times 1 \\
q \\
3 \\
2 \\
1 \\
0 \\
\hline
\end{array}$

$\begin{array}{c}
p \times 3 \\
p \\
1 \\
0 \\
\hline
\end{array}$

$\begin{array}{c}
\text{[((}q, \text{write}(b), r\text{)), 3)} \\
\hline
\end{array}$
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 (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 (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

\[ q_0 \xrightarrow{\text{write}(a)} q_1 \]
\[ q_0 \xrightarrow{\text{Inc}} q_2 \]
\[ q_2 \xrightarrow{\text{read}^{-1}(a)} q_5 \]
\[ q_3 \xrightarrow{\text{read}^{-1}(d_0)} q_4 \]
\[ q_4 \xrightarrow{\text{write}(b)} q_1 \]
\[ q_5 \xrightarrow{\text{read}^0(d_0)} q_6 \]
\[ q_6 \xrightarrow{\text{read}^0(b)} q_{\text{err}} \]

- Initial state: \( q_0 \)
- Error state: \( q_{\text{err}} \)
- Initial symbol: \( d_0 \)
- Read initial symbol from register of current round
- Read a from register of previous round
- Increment round
- \( v = 1 \) (processes can read one round back)
A small example

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

<table>
<thead>
<tr>
<th>1</th>
<th>$d_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$q_0$</td>
</tr>
</tbody>
</table>
A small example

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

| 0 | $q_0$ | $d_0$ |
| 1 | $q_2$ | $d_0$ |
A small example

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

write(a)
A small example

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

\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:
A small example

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

$$
\begin{align*}
1 & \quad q_2 & d_0 \\
0 & \quad q_0 & \quad q_1 & a
\end{align*}
$$
A small example

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

1. $q_1$ writes $a$
2. $q_0$ reads $d_0$
3. $q_2$ writes $a$
4. $q_3$ reads $d_0$
5. $q_4$ writes $b$
6. $q_5$ reads $a$
7. $q_6$ reads $d_0$
8. $q_{err}$ reads $b$
A small example

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

reads $d_0$

1 $q_6$ $d_0$

0 $q_1$ a
**Claim**: the system is safe.
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:
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 \(d_0\).
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 $d_0$.

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.
Theorem

There exists a (non-deterministic) polynomial-space algorithm solving the (dual of the) parameterized safety problem.
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
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
- Rely on a sliding window along the rounds
A visual display for executions

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

Moves: $\theta_0$, $\theta_1$, $\theta_2$, $\theta_3$, $\theta_4$, $\theta_5$, $\theta_6$

Rounds: 1, 4, 3, 2, 0, 1, 4
Here $v = 1$: processes at round $k$ can read from rounds $k$ and $k-1$.
Intuitive idea of proceeding move by move is not working:

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

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

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

Number of relevant rounds at a given time may be exponential... storable in polynomial space? not too wide in the abstract semantics → storable in polynomial space sliding window θ₄ is forgotten θ₃ is inserted between θ₀ and θ₅
The sliding window

And so on...

Number of relevant rounds at a given time may be exponential... storable in polynomial space → not too wide in the abstract semantics → storable in polynomial space.
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
The considered algorithms

Our model

The safety problem

Results

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.