Parameterized 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
- Asynchrony: some processes might be faster than others
- Non-atomic reads/writes, no fault
- Round-based: each process has its own round number $k$, which only increases; each round has its own set of registers.
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 reads/writes, no fault
- Round-based: each process has its own round number $k$, which only increases; each round has its own set of registers.

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;

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 $p$;

$k := k + 1$;

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

Aspnes’ consensus algorithm illustrated

\[
\begin{array}{c|c|c}
\text{rg}_0[k] & \text{rg}_1[k] \\
\hline
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\end{array}
\]

A \quad B \quad C
Aspnes’ consensus algorithm illustrated

\[ \begin{array}{c|c|c}
\text{process} & \text{writes} & \text{reads} \\
0 & 0 & \text{no} \\
1 & 1 & \text{no} \\
2 & \text{no} & \text{no} \\
3 & \text{no} & \text{no} \\
\end{array} \]

\[ \begin{array}{c|c|c}
\text{process} & \text{writes} & \text{reads} \\
\text{A} & 0 & \text{no} \\
\text{B} & 1 & \text{no} \\
\text{C} & 0 & \text{yes} \\
\end{array} \]
Aspnes’ consensus algorithm illustrated

\[ \begin{array}{cccc}
0 & 1 & 0 \\
A & B & C \\
\end{array} \]

\[ \begin{array}{cc}
rg_0[k] & \text{writes} \\
\text{reads} & \text{no} \rightarrow \text{no} \\
\text{reads} & \text{no} \rightarrow \text{no} \\
\text{reads} & \text{no} \rightarrow \text{no} \\
\text{reads} & \text{yes} \rightarrow \text{yes} \\
\end{array} \]
Aspnes’ consensus algorithm illustrated

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

\begin{align*}
0 & \quad \text{no} & \quad \text{no} \\
1 & \quad \text{no} & \quad \text{no} \\
2 & \quad \text{no} & \quad \text{no} \\
3 & \quad \text{yes} & \quad \text{yes}
\end{align*}

\begin{tabular}{ccc}
A & B & C \\
0 & 0 & 1 \\
1 & & \\
& 1 & 0 \\
\end{tabular}
Aspnes’ consensus algorithm illustrated

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

- **B** wants to write its preference on **0**
- **Non-atomic:** **A** may move before **B** writes
- All processes getting to round 3 will take preference 1

<table>
<thead>
<tr>
<th></th>
<th>(rg_0[k])</th>
<th>(rg_1[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>
Aspnes’ consensus algorithm illustrated

\[ \begin{array}{c|c|c}
\text{rg}_0[k] & \text{rg}_1[k] \\
\hline
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{no} & \text{no} \\
\text{yes} & \text{yes} \\
\end{array} \]
Aspnes’ consensus algorithm illustrated

\[\begin{array}{ccc}
\text{reads} & \text{writes} & \\
\text{rg}_0[k] & \text{rg}_1[k] & \\
\text{no} & \text{no} & \\
\text{no} & \text{no} & \\
\text{no} & \text{no} & \\
\text{yes} & \text{yes} & \\
\end{array}\]

\(B\) wants to write its preference on \(\text{rg}_1[k]\)

Non-atomic: \(A\) may move before \(B\) writes
Aspnes’ consensus algorithm illustrated

\[
\begin{align*}
\text{reads} & \quad \text{writes} \\
\text{rg}_0[k] & \quad \text{rg}_1[k] \\
\text{no} & \quad \text{no} \\
\text{no} & \quad \text{no} \\
\text{no} & \quad \text{no} \\
\text{yes} & \quad \text{yes} \\
\end{align*}
\]
Aspnes’ consensus algorithm illustrated

<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>no</td>
</tr>
<tr>
<td>C</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Aspnes’ consensus algorithm illustrated

\[ \text{writes} \]

\begin{align*}
\text{A} & \quad \text{B} & \quad \text{C} \\
0 & \quad 1 & \quad 0 \\
1 & \quad 0 & \quad 1 \\
2 & \quad \text{no} & \quad \text{no} \\
3 & \quad \text{no} & \quad \text{no} \\
\end{align*}

\[ \begin{array}{cc}
rg_0^k & \quad \text{no} \\
rg_1^k & \quad \text{yes} \\
\end{array} \]
Aspnes’ consensus algorithm illustrated

<table>
<thead>
<tr>
<th></th>
<th>rg₀[k]</th>
<th>rg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

A wants to write its preference on rg₀
Non-atomic: A may move before B writes

All processes getting to round 3 will take preference 1

No preference wins on this round
Aspnes’ consensus algorithm illustrated

<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>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Aspnes' consensus algorithm illustrated

<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>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>

- A wants to write its preference on rg
- Non-atomic: A may move before B writes reads writes
- B wins the race and decides, returns value 1
- All processes getting to round 3 will take preference 1
Aspnes’ consensus algorithm illustrated
Aspnes’ consensus algorithm illustrated

The considered algorithms

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


The safety problem


Results


About the algorithm


Conclusion


Aspnes’ consensus algorithm illustrated

<table>
<thead>
<tr>
<th>writes</th>
<th>0_0[k]</th>
<th>0_1[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Non-atomic: A may move before B writes.

All processes getting to round 3 will take preference 1.

A want to write its preference on 0. B wants to write its preference on 1. No preference wins on this round.
Aspnes’ consensus algorithm illustrated

- **Algorithm Overview**: Non-atomic: A may move before B writes. B wins the race and decides, returns value 1.

- **Process Timeline**:
  - Process **B** wins the race and decides, returns value 1.
  - All processes getting to round 3 will take preference 1.

- **Table**:

<table>
<thead>
<tr>
<th>Round</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no</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>no</td>
<td>no</td>
</tr>
</tbody>
</table>

- **Variables**:
  - $rg_0[k]$ and $rg_1[k]$
Aspnes’ consensus algorithm illustrated

\[
\begin{array}{c|c|c|c|c}
& \text{rg}_0[k] & \text{rg}_1[k] \\
\hline
0 & \text{yes} & \text{yes} \\
1 & \text{yes} & \text{yes} \\
2 & \text{no} & \text{yes} \\
3 & \text{no} & \text{no} \\
\hline
\end{array}
\]
Aspnes’ consensus algorithm illustrated

A  B  C

reads

rg₀[k]  rg₁[k]

no  no

no  yes

yes  yes

yes  yes
Aspnes' consensus algorithm illustrated
Aspnes’ consensus algorithm illustrated

A
B
C

writes

rg₀[k]  rg₁[k]

0
1
2
3

A
B
C

B wants to write its preference on rg₁

All process getting to round 3 will take preference 1
Aspnes’ consensus algorithm illustrated

The considered algorithms
Our model
The safety problem
Results
About the algorithm
Conclusion

Aspnes’ consensus algorithm illustrated

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A      B      C

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

reads

non      yes

non      yes

yes      yes

yes      yes

A may move before B writes reads

B wants to write its preference on rg

All process getting to round 3 will take preference 1
Aspnes’ consensus algorithm illustrated

process $B$ wins the race and decides, returns value 1

\[
\begin{array}{ccc}
3 & 1 & \checkmark \\
2 \\
1 & 0 \\
0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{A} & \text{B} & \text{C} \\
\end{array}
\]

\[
\begin{array}{cc}
\text{rg}_0[k] & \text{rg}_1[k] \\
\text{no} & \text{yes} \\
\text{no} & \text{yes} \\
\text{yes} & \text{yes} \\
\text{yes} & \text{yes} \\
\end{array}
\]

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}.

- One model for all processes: a finite automaton
- Each process has its own round number, which only increases
- A fixed number $d$ of registers per round (the total number of registers is hence unbounded)

\begin{itemize}
  \item \texttt{write(a)}
  \item \texttt{read} $^{−1}(a)$
  \item \texttt{write(b)}
  \item \texttt{read} $^{0}(b)$
\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 limited visibility range

\[
\begin{align*}
  k + 1 & \quad \cdots \quad rg_1[k + 1] & \quad \cdots \quad rg_d[k + 1] \\
  k & \quad \text{Process} \\
  k - 1 & \quad \cdots \quad rg_1[k - 1] & \quad \cdots \quad rg_d[k - 1] \\
  \vdots & & \vdots \\
  k - v & \quad \cdots \quad rg_1[k - v] & \quad \cdots \quad rg_d[k - v] \\
  k - v - 1 & \quad \cdots \quad rg_1[k - v - 1] & \quad \cdots \quad rg_d[k - v - 1]
\end{align*}
\]

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

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

\((q, \text{write}(b), r, 3)\)  
"a process on round 3 moves from \( q \) to \( r \) and writes \( b \) to the register of round 3"
Semantics of the model

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

Initial configuration of size $n$:
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. Both Agreement and Validity of Aspnes’ consensus algorithm can be encoded as safety properties.
A small example

\begin{itemize}
  \item \textbf{q}_1 \xrightarrow{\text{write}(a)} \textbf{q}_0 \xrightarrow{\text{Inc}} \textbf{q}_2 \xrightarrow{\text{Inc}} \textbf{q}_3 \xrightarrow{\text{write}(a)} \textbf{q}_4 \xrightarrow{\text{write}(b)} \textbf{q}_5 \xrightarrow{\text{read}^{-1}(a)} \textbf{q}_6 \xrightarrow{\text{read}^0(b)} \textbf{q}_{\text{err}}
\end{itemize}

1. \textbf{q}_1: initial state
2. \textbf{q}_0: increment round
3. \textbf{q}_2: read initial symbol \(d_0\)
4. \textbf{q}_3: read a from register of round \(k - 1\)
5. \textbf{q}_4: error state
6. \textbf{q}_5: read initial symbol \(d_0\)
7. \textbf{q}_6: read initial symbol \(d_0\)

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

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

\[
\begin{array}{c|c}
1 & d_0 \\
0 & q_0 & d_0
\end{array}
\]
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:

write(a)

write(a)

read$^{-1}(d_0)$

read$^0(d_0)$

read$^0(b)$

writes a

a

$q_3$

$q_4$

$q_6$

$q_{err}$
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:

$$\begin{align*}
1 & \quad d_0 \\
0 \quad q_0 \times 2 & \quad d_0
\end{align*}$$
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:
State \( q_6 \) can be covered from the initial configuration with two processes:

1. \( \negrightarrow q_5 \) reads \( a \) on \( d_0 \)
2. \( \negrightarrow q_1 \) reads \( a \)
A small example

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

- From $q_1$ to $q_6$:
  - Write $a$ in $q_1$
  - Inc in $q_1$
  - $q_0$ to $q_2$
  - Inc in $q_2$
  - Write $a$ in $q_2$
  - $q_3$ to $q_4$
  - Read $-1(a)$ in $q_3$
  - $q_4$ to $q_6$
  - Write $b$ in $q_4$
  - $q_5$ to $q_6$
  - Read $-1(a)$ in $q_5$
  - $q_6$ to $q_{err}$
  - Read $0(b)$ in $q_6$

- From $q_0$ to $q_6$:
  - Write $a$ in $q_0$
  - Inc in $q_0$
  - $q_2$ to $q_5$
  - Inc in $q_2$
  - Read $-1(a)$ in $q_2$
  - $q_5$ to $q_4$
  - Read $-1(d_0)$ in $q_5$
  - $q_4$ to $q_6$
  - Write $b$ in $q_4$
  - $q_6$ to $q_{err}$
  - Read $0(b)$ in $q_6$
A small example

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

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

Claim: the system is safe.
A small example

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. This is not possible:
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. This is not possible:

$(q_4, k)$ and $(q_6, k)$ are incompatible
**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. This is not possible:

- $(q_4, k)$ and $(q_6, k)$ are incompatible

  - To cover $(q_4, k)$, one must write to $rg[k]$ while $rg[k-1]$ still has value $d_0$;
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. This is not possible:

$(q_4, k)$ and $(q_6, k)$ are incompatible

- 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!
Theorem

Parameterized safety in round-based register protocols is PSPACE-complete.⁴

⁴Nathalie Bertrand, Nicolas Markey, Ocan Sankur, Nicolas Waldburger. Parameterized safety verification of round-based shared-memory systems. ICALP’22
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.
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!
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)
- Exploit locality of reads and writes 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)
- Exploit locality of reads and writes with respect to the round
- Rely on a sliding window along the rounds
An abstraction for safety

Two observations

- Whenever an execution puts one process on \((q, k)\), one can put arbitrarily many processes on \((q, k)\) (increasing the total number of processes).
- Whenever a symbol \(x \neq d_0\) has been written to some register \(rg_k[\alpha]\), it can be written to this register again.
An abstraction for safety

Two observations

- Whenever an execution puts one process on \((q, k)\), one can put arbitrarily many processes on \((q, k)\) (increasing the total number of processes).
- Whenever a symbol \(x \neq d_0\) has been written to some register \(rg_k[\alpha]\), it can be written to this register again.

An abstraction for safety

In an abstract configuration, one remembers only:

- which pairs (state, round) are populated by at least one process,
- which registers still have symbol \(d_0\) and which symbols can be written to which registers.
An abstraction for safety

Two observations

- Whenever an execution puts one process on \((q, k)\), one can put arbitrarily many processes on \((q, k)\) (increasing the total number of processes).
- Whenever a symbol \(x \neq d_0\) has be written to some register \(r_{g_k}[\alpha]\), it can be written to this register again.

An abstraction for safety

In an abstract configuration, one remembers only:

- which pairs (state, round) are populated by at least one process,
- which registers still have symbol \(d_0\) and which symbols can be written to which registers.

Soundness and completeness of the abstraction

For all \(q\) and \(k\), \((q, k)\) is coverable in the abstract semantics if and only if it is coverable in the original semantics.
Locations covered: $\{(q_0, 0)\}$
Symbols: $\emptyset$
Locations covered: \(((q_0, 0), (q_2, 1))\)
Symbols: \(\emptyset\)
Back to the example

- **Locations covered**: \[\{(q_0, 0), (q_2, 1), (q_1, 0)\}\]
- **Symbols**: \[\text{rg}[0] : \{a\}\]
Back to the example

Possible to read a from $rg[0]$

Locations covered: $\{(q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1)\}$

Symbols:
- $rg[0]: \{a\}$
Back to the example

rg[1] does not appear below: it still has value $d_0$

**Locations covered**: $\{(q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1), (q_6, 1)\}$

**Symbols**: $\text{rg}[0]: \{a\}$
Locations covered: \{ (q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1), (q_6, 1), (q_3, 1) \}

Symbols:
- \( \text{rg[0]} : \{a\} \)
- \( \text{rg[1]} : \{a\} \)
Back to the example

Locations covered: \{ (q_0, 0), (q_2, 1), (q_1, 0), (q_5, 1), (q_6, 1), (q_3, 1) \}

Symbols: \( \text{rg}[0] : \{ a \} , \text{rg}[1] : \{ a \} \)

\( \text{rg}[0] \) no longer has value \( d_0 \)
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: \[ 1, 4, 3, 2, 0, 1, 4 \]

rounds

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>\theta_1</td>
<td>\theta_2</td>
<td>\theta_3</td>
<td>\theta_4</td>
<td>\theta_5</td>
</tr>
</tbody>
</table>

“time”
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...
Instead: sliding window along the rounds non-deterministically guessing the execution

The sliding window

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: on moves of rounds 1 and 2 in the case of $\theta_3$. 

$\theta_4$ is forgotten 

$\theta_3$ is inserted between $\theta_0$ and $\theta_5$ 

Number of relevant rounds at a given time may be exponential... storable in polynomial space?

not too wide in the abstract semantics $\rightarrow$ storable in polynomial space
The sliding window

And so on...

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

θ_0 is forgotten, θ_3 is inserted between θ_0 and θ_5.
The sliding window

And so on...

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

sliding window $\theta_4$ is forgotten $\theta_3$ is inserted between $\theta_0$ and $\theta_5$
The polynomial-space algorithm

Outline of the algorithm

At round $k$:

- non-deterministically insert moves of round $k+1$,
- check that inserted moves are valid (possible with local information),
- forget all about round $k-\nu$ and move to next round.
The polynomial-space algorithm

Outline of the algorithm

At round $k$:

- non-deterministically insert moves of round $k+1$,
- check that inserted moves are valid (possible with local information),
- forget all about round $k-v$ and move to next round.

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

After an exponential number of steps, the information has looped and the algorithm stops.
Exponential upper bounds

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

**Exponential upper bound on “cutoff”**
There exists an exponential upper bound on the number of processes needed to reach the error state.

**Exponential upper bound on the number of rounds**
There exists an exponential upper bound on the number of rounds needed to reach the error state.
Ongoing and future work

Summary

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

Future work

- Generalisation to parameterized TARGET (“Is it possible, with enough processes, to make all processes converge towards a given state?”) and other similar problems.
- Parameterized problems for infinite executions, for example INEVITABILITY: do all infinite executions encounter a given state?
- Study almost-sure reachability in round-based register protocols with stochastic schedulers (termination of Aspnes’ algorithm).
- Explore its links with classical notions of fairness.
Classical notions of fairness are not satisfying here

Classical notions of fairness are not satisfying

- If fairness = “any move that is available infinitely often is taken infinitely often”: a fair execution with two processss can stay on $q_0$ by taking each $((q_0, \text{Inc}, q_0), k)$ twice, never reaching $q_{err}$.
- If fairness = “any transition that is available infinitely often is taken infinitely often (regardless of round)”: a fair execution with two processes can have one of them go to $q_1$ and back to $q_0$ on every round, while the other process stays on $q_0$ forever: transition from $q_1$ to $q_{err}$ is never enabled.
Termination of Aspnes’ consensus algorithm requires that “a process wins the race”, i.e., that processes are not too close in terms of speed. The two notions of fairness above cannot guarantee this. Neither does: “for all $m$, any process eventually acts $m$ times in a row while all other processes are idle”.

A stochastic scheduler allows to guarantee Aspnes’ termination. It could also be proven with some “ad hoc” fairness property.