Checking Presence Reachability Properties on Parameterized Shared-Memory Systems

Nicolas Waldburger
PhD supervisors: Nathalie Bertrand, Nicolas Markey, Ocan Sankur
Séminaire MOVE, 29/07/2023
Parameterized verification

- Arbitrary number of processes
- Processes are identical agents
- No identifiers: processes are anonymous
- Modelled by a single, common finite automaton

Shared-memory systems

Two models in this talk:

- Simple model: shared-memory systems with finite memory
- More complex model: round-based shared-memory systems
A model for shared-memory systems\(^1\)

Finite number of shared registers, each register has a value from finite set of symbols \(\Sigma\)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(d_0)</td>
<td></td>
</tr>
</tbody>
</table>

Registers are initialized to value \(d_0\)

No atomic read/write combinations

---

A configuration:

| $q \times 2$ | $p \times 1$ | a | b | $d_0$ |

How many process are on each state  
Content of the registers
Semantics

\[
\begin{array}{c}
\begin{array}{c}
q \times 2 \quad p \times 1 \quad a \quad b \quad d_0
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
(q, write_3(a), r)
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
q \times 1 \quad p \times 1 \quad r \times 1 \quad a \quad b \quad a
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
(p, read_1(a), r)
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
q \times 1 \quad r \times 2 \quad a \quad b \quad a
\end{array}
\end{array}
\]
Initial configurations:

\[(q, \text{write}_3(a), r)\]

\[(p, \text{read}_1(a), r)\]

Registers are initialized to \(d_0\)

Can be arbitrarily large

with \(n \geq 1\) and \(q_0\) the initial state
A small example

A single register

A single register

\[
\begin{align*}
q_0 & \xrightarrow{\text{read}(d_0)} B \\
& \quad \xrightarrow{\text{write}(c)} A \\
& \quad \xrightarrow{\text{read}(c)} C \\
& \quad \xrightarrow{\text{write}(b)} q_f \\
& \quad \xrightarrow{\text{read}(b)} C \\
& \quad \xrightarrow{\text{write}(a)} q_f \\
& \xrightarrow{\text{read}(a)} q_f \\
\end{align*}
\]
A small example

Two processes

Initial value

d_0

\( \text{write}(c) \)

\( \text{read}(d_0) \)

\( \text{read}(d_0) \)

\( \text{read}(d_0) \)

\( \text{write}(a) \)

\( \text{write}(b) \)

\( \text{read}(a) \)

\( \text{read}(c) \)

\( \text{read}(c) \)

\( \text{read}(b) \)

\( \text{read}(b) \)

\( \text{write}(b) \)

\( \text{write}(b) \)

\( \text{write}(a) \)

\( \text{write}(a) \)
A small example

\[ \begin{array}{c}
\text{\texttt{d}_0} \\
q_0 \quad \text{write}(c) \\
B \quad \text{read}(d_0) \\
\text{\texttt{c}} \\
A \quad \text{read}(a) \\
\text{\texttt{a}} \\
C \quad \text{read}(b) \\
\text{\texttt{b}} \\
q_f \quad \text{write}(b) \\
\end{array} \]
A small example
A small example
A small example

![Diagram of a small example]

- **States:** $q_0$, $A$, $B$, $C$, $q_f$
- **Transitions:**
  - $q_0$ to $A$: write($c$)
  - $A$ to $q_f$: read($a$), write($b$)
  - $B$ to $A$: read($d_0$), read($c$)
  - $A$ to $C$: read($a$)
  - $C$ to $B$: read($b$), read($d_0$)
  - $B$ to $C$: read($d_0$)
  - $C$ to $q_f$: write($a$)
A small example

\[ q_0 \rightarrow write(c) \rightarrow A \]

\[ B \rightarrow read(d_0) \rightarrow A \]

\[ C \rightarrow read(d_0) \rightarrow A \]

\[ \text{write}(a) \rightarrow C \rightarrow \text{read}(c) \rightarrow A \]

\[ \text{write}(b) \rightarrow A \rightarrow \text{read}(a) \rightarrow q_f \]

\[ q_f \text{ is covered} \]

\[ a \text{ is covered} \]
Reachability problems

COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^{*} \gamma, \gamma(q_f) > 0 \)?
Reachability problems

COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \gamma(q_f) > 0 ? \)

TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall q \neq q_f, \gamma(q) = 0 ? \)

All processes “synchronize” on \( q_f \)
Reachability problems

COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow* \gamma, \gamma(q_f) > 0 \) ?

TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow* \gamma, \forall q \neq q_f, \gamma(q) = 0 \) ?

PRP\(^2\): \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow* \gamma, \gamma \models \phi \) ?

with \( \phi \in \mathcal{B}(\{\#q = 0, \#q > 0\}, \{\text{reg}_i = d, \text{reg}_i \neq d\}) \)

\( \#q = \text{number of processes on } q \)
Reachability problems

**COVER:** \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \, \gamma(q_f) > 0 \) ?

**TARGET:** \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \, \forall q \neq q_f, \, \gamma(q) = 0 \) ?

**PRP²:**

\[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \, \gamma \models \phi \]

with \( \phi \in \mathcal{B}(\{\#q = 0, \#q > 0\}, \{\text{reg}_i = d, \text{reg}_i \neq d\}) \)

Examples:
- \( \phi = "#q_f > 0" \) (COVER),
- \( \phi = "\land q \neq q_f \#q = 0" \) (TARGET)
- \( \phi = "(#q_1 > 0) \lor ([#q_2 = 0] \land [\text{reg}_1 = d_0])" \)

---

Monotonicity

A process may “copy” the behavior of another process on the same state.

\[\text{write}(b)\]
A process may “copy” the behavior of another process on the same state.
A process may “copy” the behavior of another process on the same state.

write(b)
A process may “copy” the behavior of another process on the same state.
A process may “copy” the behavior of another process on the same state.

Monotonicity

(read(a))
Monotonicity

A process may “copy” the behavior of another process on the same state.

\[ \text{read}(a) \]
Monotonicity

Abstraction: remember whether there is at least one process on a given state.

Sound and Complete for PRP because of monotonicity property
NP-completeness of COVER

COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \ \gamma(q_f) > 0 \)?

```
1  | \hline
   | d_0 \\
2  | d_0
```

A

```
write_1(T) \quad read_2(d_0)
```

B

```
write_2(T) \quad read_1(d_0)
```
NP-completeness of COVER

COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \gamma(q_f) > 0 \) ?

Reduction from 3-SAT:

\[
\begin{array}{c|c|c}
\hline
x & d_0 \\
\hline
\neg x & d_0 \\
\hline
\end{array}
\]

Check \( x \):

Check \( \neg x \):

\[\text{read}_x(T) \rightarrow \text{read}_{\neg x}(d_0)\]

\[\text{read}_{\neg x}(T) \rightarrow \text{read}_x(d_0)\]
NP-completeness of COVER

COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \gamma(q_f) > 0? \)

Reduction from 3-SAT:

Directly relies on initialization of registers!

COVER drops down to PTIME when the registers are not initialized (applying a simple saturation technique).
TARGET when registers are not initialized

TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall q \neq q_f, \gamma(q) = 0 \) ?

TARGET is still NP-complete when registers are not initialized. Reduction from 3-SAT:

\[
\forall x \\
\text{write}_x(\text{true})
\]

\[
\forall x \\
\text{write}_x(\text{false})
\]

Check clause 1 \( \cdots \) Check clause \( m \) \( q_f \)
TARGET with a single register

TARGET:  \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \ \forall q \neq q_f, \gamma(q) = 0 ? \)

TARGET is PTIME when only one register.
One can reduce the problem to the case when the register is not initialized.
Algorithm inspired from broadcast protocols\(^4\).
TARGET: \[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \quad \forall q \neq q_f, \gamma(q) = 0? \]

TARGET is PTIME when only one register.
One can reduce the problem to the case when the register is not initialized.
Algorithm inspired from broadcast protocols\(^4\).

Compute *coverable states* (the state can be covered from initial configurations) and *backwards coverable states* (\(q_f\) may be reached from some configuration containing the state).
TARGET with a single register

TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall q \neq q_f, \gamma(q) = 0 ? \)

TARGET is PTIME when only one register.
One can reduce the problem to the case when the register is not initialized.
Algorithm inspired from broadcast protocols\(^4\).

Compute \textit{coverable states} (the state can be covered from initial configurations)
and \textit{backwards coverable states} (\(q_f\) may be reached from some configuration containing the state).

\[
\text{= coverable}
\]

\[
\text{= backwards coverable}
\]
TARGET with a single register

TARGET: \[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall q \neq q_f, \gamma(q) = 0 \]

TARGET is PTIME when only one register.
One can reduce the problem to the case when the register is not initialized.
Algorithm inspired from broadcast protocols\(^4\).

Compute *coverable states* (the state can be covered from initial configurations) and *backwards coverable states* (*q_f* may be reached from some configuration containing the state).

Iteratively remove all states that are not coverable.

---

TARGET with a single register

TARGET: \[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \ \forall q \neq q_f, \gamma(q) = 0 ? \]

TARGET is PTIME when only one register.
One can reduce the problem to the case when the register is not initialized.
Algorithm inspired from broadcast protocols\(^4\).

Compute *coverable states* (the state can be covered from initial configurations) and *backwards coverable states* (\(q_f\) may be reached from some configuration containing the state).

Iteratively remove all states that are not...
TARGET with a single register

TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall q \neq q_f, \gamma(q) = 0 ? \)

TARGET is PTIME when only one register. One can reduce the problem to the case when the register is not initialized. Algorithm inspired from broadcast protocols\(^4\).

Compute *coverable states* (the state can be covered from initial configurations) and *backwards coverable states* (*q_f* may be reached from some configuration containing the state).

Iteratively remove all states that are not }

TARGET with a single register

TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \ \forall q \neq q_f, \gamma(q) = 0 ? \)

TARGET is PTIME when only one register.
One can reduce the problem to the case when the register is not initialized.
Algorithm inspired from broadcast protocols\(^4\).

Compute *coverable states* (the state can be covered from initial configurations)
and *backwards coverable states* (\(q_f\) may be reached from some configuration containing the state).

The algorithm is generalizable to PRP when the formula is in Disjunctive Normal Form (DNF).

**DNF-PRP:** \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \ \gamma \models \phi, \)

\( \phi \) in DNF: \( \phi = \bigvee_i \left( t_{i,1} \land t_{i,2} \land \cdots \land t_{i,m_i} \right) \),

\( t_{i,j} \in \{\#q = 0, \#q > 0\} \cup \{\text{reg}_i = d, \text{reg}_i \neq d\} \)

## Summary of complexity results

<table>
<thead>
<tr>
<th></th>
<th>COVER</th>
<th>TARGET</th>
<th>DNF-PRP</th>
<th>PRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>General case</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Not initialized</td>
<td>PTIME-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>One register</td>
<td>PTIME-complete</td>
<td>PTIME-complete</td>
<td>PTIME-complete</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>
Round-based shared-memory systems
A motivating example

Binary consensus problem:
Make all processes agree on a common value, each process starting an initial preference \( p \).

Validity: If a process decided value \( p \), some process started with value \( p \)

Agreement: Two processes that decide decide of the same value

Termination: All processes eventually decide of a value

Aspnes’ consensus algorithm:

\[
\begin{align*}
\text{int } k & := 0, \text{ bool } p \in \{0, 1\}, \text{ (rg}_b[r])_{b \in \{0, 1\}, r \in \mathbb{N}} \text{ all initialized to no}; \\
\text{while } \text{true do} & \\
\quad \text{read from } \text{rg}_0[k] \text{ and } \text{rg}_1[k]; \\
\quad \text{if } \text{rg}_0[k] = \text{yes} \text{ and } \text{rg}_1[k] = \text{no} \text{ then } p := 0; \\
\quad \text{else if } \text{rg}_0[k] = \text{no} \text{ and } \text{rg}_1[k] = \text{yes} \text{ then } p := 1; \\
\quad \text{write yes to } \text{rg}_p[k]; \\
\quad \text{if } k > 0 \text{ then} \\
\quad \quad \text{read from } \text{rg}_{1-p}[k-1]; \\
\quad \quad \text{if } \text{rg}_{1-p}[k-1] = \text{no} \text{ then return } p; \\
\quad k := k+1;
\end{align*}
\]
Example of execution of the algorithm

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

- $\text{reg}_0[k]$
- $\text{reg}_1[k]$
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>Rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reg_0[k]</th>
<th>reg_1[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></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>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reg_0[k]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>reg_1[k]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>Rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( \text{reg}_0[k] )</th>
<th>( \text{reg}_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>\text{yes}</td>
<td>\text{yes}</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>\text{no}</td>
<td>\text{no}</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>\text{no}</td>
<td>\text{no}</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>\text{no}</td>
<td>\text{no}</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>\text{no}</td>
<td>\text{no}</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>Rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></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>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

The numbers 0 and 1 are used to indicate the state of the registers at each round.
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reads</th>
<th>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Pixels read from the image do not form a table or a clear description of the algorithm. The diagram shows a sequence of rounds from 0 to 4, with states A, B, and C, and flags reg₀[k] and reg₁[k] that change from no to yes.
Example of execution of the algorithm

rounds

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( \text{reg}_0[k] )</th>
<th>( \text{reg}_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Is ready to write its preference

\( r_e \) \( g \) \( k \) \( r_e \) \( g \) \( k \)
Example of execution of the algorithm

The diagram shows the execution of a three-step algorithm. The x-axis represents rounds, with integers 0 to 4 labeled. The y-axis denotes the states of variables A, B, and C, with states 0 and 1 indicated. The legend includes two columns for `reg_0[k]` and `reg_1[k]`.

At each round:
- **Round 0:**
  - A is 0
  - B is 1
  - C is 0
  - `reg_0[k]` and `reg_1[k]` are both no

- **Round 1:**
  - A is 0
  - B is 0
  - C is 0
  - `reg_0[k]` is no, `reg_1[k]` is yes

- **Round 2:**
  - A is 0
  - B is 1
  - C is no
  - `reg_0[k]` and `reg_1[k]` are both no

- **Round 3:**
  - A is 0
  - B is 0
  - C is no
  - `reg_0[k]` and `reg_1[k]` are both no

- **Round 4:**
  - A is 0
  - B is 0
  - C is no
  - `reg_0[k]` and `reg_1[k]` are both no
Example of execution of the algorithm

Is ready to write its preference

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reg_0[k]</th>
<th>reg_1[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

reads
Example of execution of the algorithm

A
B
C

\[ \text{reg}_0[k] \]

\[ \text{reg}_1[k] \]
Example of execution of the algorithm

No winner on this round
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</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>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

The diagram illustrates the execution of the algorithm with states for rounds 0 to 4, showing transitions and conditions for registers reg₀[k] and reg₁[k].
Example of execution of the algorithm

```
rere \[k\]
reg\(_0\)[k]
reg\(_1\)[k]
reads
```
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

writes

\[ r_{\text{g}}(k) \]
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( \text{reg}_0[k] )</th>
<th>( \text{reg}_1[k] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

\( r_e g \)
Example of execution of the algorithm
Example of execution of the algorithm

```
<table>
<thead>
<tr>
<th>reads</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
```

- **A**: reg0[k]
- **B**: reg1[k]
Example of execution of the algorithm

- **Round 0**: A reads reg$_0[k]$, B reads reg$_0[k]$, C reads reg$_0[k]$
- **Round 1**: A reads reg$_1[k]$, B reads reg$_1[k]$, C reads reg$_1[k]$
- **Round 2**: A reads reg$_0[k]$, B reads reg$_0[k]$, C reads reg$_0[k]$
- **Round 3**: A reads reg$_1[k]$, B reads reg$_1[k]$, C reads reg$_1[k]$
- **Round 4**: A reads reg$_0[k]$, B reads reg$_0[k]$, C reads reg$_0[k]$

The execution follows the algorithm's rules, and no conflicts occur.
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

- **rounds**: 0, 1, 2, 3, 4
- **reads**: 0, 1
- **A, B, C**
- **reg₀[k]**, **reg₁[k]**
Example of execution of the algorithm

A process getting to this round will convert to preference 1
Round-based shared-memory systems

Model inspired by round-based algorithms from the literature\textsuperscript{678}.

Process progress in asynchronous rounds, each round having its own finite set of registers.

The round-based model

- Read transitions now mention from which round they are reading, relatively to the current round of the process
- A new type of transitions: *round increments*, which send the process to the next round

Example with one register per round:

```
write(b)

read^{-1}(a)  read^{-1}(d_0)  write(a)  read^{-1}(b)  read^0(d_0)  read^0(b)
```

- Write to register of the current round of the process
- Read from register of current round of the process
- Read from register one round below the round of the process
- Increment round
Semantics

\[ p \times 1 \quad : \quad d_0 \]
\[ \begin{array}{ccc}
3 & q \times 3 & 1 \\
2 & b & a \\
1 & a & d_0 \\
0 & d_0 & \\
\end{array} \]

\[ (p, \text{read}^{-1}(b), r), 3 \]

\[ \begin{array}{ccc}
3 & r \times 1 & 1 \\
2 & b & a \\
1 & a & d_0 \\
0 & d_0 & \\
\end{array} \]

here with one register per round
Semantics

\[ (q, \text{write}(b), r), 1 \]

here with one register per round
## Semantics

Initial configurations:

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>b</td>
<td>a</td>
<td>d₀</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

(q, write(b), r), 1

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>d₀</td>
<td>d₀</td>
<td>d₀</td>
</tr>
</tbody>
</table>

Nicolas Waldburger
Abstraction

Initial configurations:

\[
\begin{align*}
&\vdots \\
&3 \\
&2 \\
&1 \\
&0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&b \\
&a \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&\vdots \\
&3 \\
&2 \\
&1 \\
&0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&b \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&\vdots \\
&2 \\
&1 \\
&0 \\
&\vdots \\
&d_0 \\
&d_0 \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&b \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&d_0 \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&d_0 \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&d_0 \\
&d_0
\end{align*}
\]

\[
\begin{align*}
&d_0 \\
&d_0 \\
&d_0
\end{align*}
\]
An example of round-based register protocol

```
write(b)

read^{-1}(a)  read^{-1}(d_0)  write(a)  read^{-1}(b)  read^{0}(d_0)  read^{0}(b)  q_f

Increment round

0 1 2  \vdots

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>q_0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

= round 0
= round 1
= round 2
```
An example of round-based register protocol

Increment round

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_0$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$d_0$</td>
<td>$d_0$</td>
<td>$d_0$</td>
</tr>
</tbody>
</table>
An example of round-based register protocol

\[
\begin{align*}
\text{write}(b) & : \quad \text{write}(a) \\
read^{-1}(a) & : \quad \text{write}(a) \\
read^{-1}(d_0) & : \quad \text{write}(a) \\
\text{round}^{-1}(b) & : \quad \text{write}(a) \\
\end{align*}
\]
An example of round-based register protocol

\[\text{write}(b)\]

\[\begin{array}{cccccc}
\text{C} & \text{B} & \text{A} & \text{q}_0 & \text{D} & \text{E} \\
\text{read}^{-1}(a) & \text{read}^{-1}(d_0) & \text{write}(a) & \text{read}^{-1}(b) & \text{read}^0(d_0) & \text{read}^0(b)
\end{array}\]

Increment round:

\[\begin{array}{c|c|c}
0 & d_0 & a \\
1 & q_0 & A \quad B \\
2 & q_0 & d_0
\end{array}\]

- \(q_f\)

= round 0

= round 1

= round 2
An example of round-based register protocol

![Diagram of round-based register protocol]

- \( q_0 \) is the initial state.
- Increment round: 1, 2, 3
- \( q_f \) is the final state.

<table>
<thead>
<tr>
<th>Round</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( q_0 )</td>
<td>( q_0 )</td>
<td>( q_0 )</td>
</tr>
<tr>
<td>1</td>
<td>( q_0 )</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( a \) is the input symbol.
- \( d_0 \) is the data symbol.

- Green circles represent round 0.
- Blue circles represent round 1.
- Yellow circles represent round 2.

Nicolas Waldburger
An example of round-based register protocol

![Diagram](image)

<table>
<thead>
<tr>
<th>Round</th>
<th>States</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(q_0)</td>
<td>(a)</td>
</tr>
<tr>
<td>1</td>
<td>(A), (B), (C)</td>
<td>(d_0)</td>
</tr>
<tr>
<td>2</td>
<td>(q_0), (A)</td>
<td>(b)</td>
</tr>
</tbody>
</table>

- \(\text{write}(b)\)
- \(\text{write}(a)\)
- \(\text{read}^{-1}(a)\)
- \(\text{read}^{-1}(d_0)\)
- \(\text{read}^{-1}(b)\)
- \(\text{read}^0(d_0)\)
- \(\text{read}^0(b)\)

- \(\text{Increment round}\)

= round 0
= round 1
= round 2
An example of round-based register protocol

\begin{itemize}
\item $\text{write}(b)$
\item $\text{read}^{-1}(a)$
\item $\text{read}^{-1}(d_0)$
\item $\text{write}(a)$
\item $\text{read}^{-1}(b)$
\item $\text{read}^0(d_0)$
\item $\text{read}^0(b)$
\end{itemize}

Increment round

$\vdots$

2

1 $q_0$ A B C

0 $q_0$ A

$\vdots$

$d_0$

b

$\vdots$

$\text{round } 0$

$\text{round } 1$

$\text{round } 2$
An example of round-based register protocol

\[
\begin{align*}
\text{write}(b) & \quad \xrightarrow{\text{read}^{-1}(b)} A \\
\text{write}(a) & \quad \xrightarrow{\text{read}^{-1}(a)} B \\
\end{align*}
\]

Increment round

\[
\begin{array}{c|c|c}
\vdots & \vdots & \vdots \\
2 & q_0 & d_0 \\
1 & q_0 & b \\
0 & q_0 & a \\
\end{array}
\]

= round 0
= round 1
= round 2

Nicolas Waldburger
An example of round-based register protocol

Increment round

<table>
<thead>
<tr>
<th>:</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$q_0$ $D$</td>
</tr>
<tr>
<td>1</td>
<td>$q_0$ $A$ $B$ $C$</td>
</tr>
<tr>
<td>0</td>
<td>$q_0$ $A$</td>
</tr>
</tbody>
</table>

- $= \text{round 0}$
- $= \text{round 1}$
- $= \text{round 2}$

Nicolas Waldburger
An example of round-based register protocol

\[ \begin{align*}
\text{read}^{-1}(a) &\quad \text{read}^{-1}(d_0) &\quad \text{write}(a) &\quad \text{read}^{-1}(b) &\quad \text{read}^0(d_0) &\quad \text{read}^0(b) \\
C &\quad B &\quad A &\quad q_0 &\quad D &\quad E \\
\text{write}(b) &\quad &\quad &\quad &\quad &\quad \text{q}_f
\end{align*} \]

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>q_0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>q_0</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>q_0</td>
<td>yellow</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{write}(a) &\quad \text{read}^{-1}(b) &\quad \text{read}^0(d_0) \\
A &\quad B &\quad C &\quad D &\quad E
\end{align*} \]

= round 0
= round 1
= round 2

\[ \begin{align*}
\text{write}(b) &\quad \text{read}^{-1}(a) \\
A &\quad B &\quad C &\quad D &\quad E
\end{align*} \]

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>q_0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>q_0</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>q_0</td>
<td>yellow</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{write}(b) &\quad \text{read}^{-1}(a) \\
A &\quad B &\quad C &\quad D &\quad E
\end{align*} \]

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>q_0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>q_0</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>q_0</td>
<td>yellow</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{write}(a) &\quad \text{read}^{-1}(b) &\quad \text{read}^0(d_0) \\
A &\quad B &\quad C &\quad D &\quad E
\end{align*} \]

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>q_0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>q_0</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>q_0</td>
<td>yellow</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{write}(b) &\quad \text{read}^{-1}(a) \\
A &\quad B &\quad C &\quad D &\quad E
\end{align*} \]

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>q_0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>q_0</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>q_0</td>
<td>yellow</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{write}(a) &\quad \text{read}^{-1}(b) &\quad \text{read}^0(d_0) \\
A &\quad B &\quad C &\quad D &\quad E
\end{align*} \]

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>q_0</td>
<td>red</td>
</tr>
<tr>
<td>1</td>
<td>q_0</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>q_0</td>
<td>yellow</td>
</tr>
</tbody>
</table>
An example of round-based register protocol

To write $b$ to $\text{reg}[k]$, one must write to $\text{reg}[k]$ while $\text{reg}[k-1]$ still has value $d_0$

To cover $q_f$ at round $k$, one must have written $b$ to $\text{reg}[k-1]$ while $\text{reg}[k]$ still has value $d_0$

$q_f$ cannot be covered!
Reachability problems in round-based setting

Round-based COVER: \[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \exists k \gamma(q_f, k) > 0 \]

There exists a round \( k \) such that some process is at round \( k \) and on state \( q_f \)
Reachability problems in round-based setting

Round-based COVER: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \exists k \gamma(q_f, k) > 0 \) ?

Round-based TARGET: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall k, \forall q \neq q_f, \gamma(q, k) = 0 \) ?

Every process is on state \( q_f \) regardless of its round
Reachability problems in round-based setting

Round-based COVER:

\[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \exists k \gamma(q_f,k) > 0 ? \]

Round-based TARGET:

\[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \forall k, \forall q \neq q_f, \gamma(q,k) = 0 ? \]

Round-based PRP:

\[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \gamma \models \psi ? \]

with \( \psi \) a first-order formula on rounds with no nested quantifiers

Examples:

\[ \psi = \left( \exists k \left( \#(q_1, k + 1) > 0 \land \text{reg}_i[k] = d \right) \right) \lor \left( \forall k \#(q_0, k) = 0 \right) \]

- At some round, there is a process on state \( q_1 \) while register \( i \) of previous round has value \( d \)
- no process is on \( q_0 \)
A challenge: exponential lower bounds

Exponential lower bounds on the number of rounds:
A challenge: exponential lower bounds

Exponential lower bounds on the number of rounds:

Similar lower bounds for the number of processes and of active rounds
Complexity results

*Theorem*\(^9\): Round-based COVER is PSPACE-hard.
Complexity results

Theorem\textsuperscript{9}: Round-based COVER is PSPACE-hard.

Theorem\textsuperscript{9,10}: Round-based PRP is PSPACE-complete.

Complexity results

Theorem\(^9\): Round-based COVER is PSPACE-hard.

Theorem\(^{9,10}\): Round-based PRP is PSPACE-complete.

Challenge: the number of rounds relevant at the same time may need to be exponential.

A non-deterministic polynomial-space algorithm

Witness execution: \( \sigma_0 \xrightarrow{\theta_0} \sigma_1 \xrightarrow{\theta_1} \sigma_2 \xrightarrow{\theta_2} \sigma_3 \xrightarrow{\theta_3} \sigma_4 \xrightarrow{\theta_4} \sigma_5 \xrightarrow{\theta_5} \sigma_6 \xrightarrow{\theta_6} \sigma_7 \models \psi \)
A non-deterministic polynomial-space algorithm

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

Actions: $\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6$

Rounds: 1 4 3 2 0 1 4
A non-deterministic polynomial-space algorithm

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

Actions: $\theta_0 \ \theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6$

Rounds: $1 \ 4 \ 3 \ 2 \ 0 \ 1 \ 4$

![Diagram showing the execution steps and actions]
A non-deterministic polynomial-space algorithm
A non-deterministic polynomial-space algorithm

number of relevant rounds may be large…

storable in polynomial space?
A non-deterministic polynomial-space algorithm

sliding window on \( \nu + 1 \) rounds where \( \nu \) is the highest \( i \) such that some \( \text{read}^{-i}(x) \) appears in the protocol.

\( \nu \) is assumed to be given in unary (here \( \nu = 1 \))

storable in polynomial space using abstract representation
A non-deterministic polynomial-space algorithm

- Insert actions taking place at round 2
- Forget about round 0
A non-deterministic polynomial-space algorithm
As the execution is guessed, we progressively guess why the configuration reached will satisfy $\psi$. 

A non-deterministic polynomial-space algorithm
A non-deterministic polynomial-space algorithm

As the execution is guessed, we progressively guess why the configuration reached will satisfy $\psi$.

From this algorithm, we obtain exponential upper bounds on the number of processes and rounds needed.
Round-based shared-memory systems with stochastic schedulers
Many consensus algorithms rely on good luck for termination.

First idea: considering *fair* executions.
Many consensus algorithms rely on good luck for termination.

First idea: considering *fair* executions.

**Definition 1:** For a given $k$, any transition that is available infinitely often at round $k$ is taken infinitely often.
Many consensus algorithms rely on good luck for termination.

First idea: considering *fair* executions.

**Definition 1:** For a given $k$, any transition that is available infinitely often at round $k$ is taken infinitely often.

**Definition 2:** Any transition that is available infinitely often overall is taken infinitely often.
Many consensus algorithms rely on good luck for termination.

First idea: considering *fair* executions.

**Definition 1**: For a given $k$, any transition that is available infinitely often at round $k$ is taken infinitely often.

**Definition 2**: Any transition that is available infinitely often overall is taken infinitely often.

**Definition 3**: Processes tend to perform similar number of steps.

*For Aspnes’ algorithm, a process must win the race!*
Many consensus algorithms rely on good luck for termination.

First idea: considering *fair* executions.

**Definition 1**: For a given $k$, any transition that is available infinitely often at round $k$ is taken infinitely often.

**Definition 2**: Any transition that is available infinitely often overall is taken infinitely often.

**Definition 3**: Processes tend to perform similar number of steps.

For Aspnes’ algorithm, a process must win the race!

**Definition 4**: For every $m$, every process eventually performs $m$ steps in a row.

For Aspnes’ algorithm, a process that is far behind could perform many steps in a row and not decide...

→ We need stochastic schedulers!
At every step:
- the next process to move is picked uniformly at random among all processes,
- its action is picked uniformly at random among all its available actions.

**Almost-sure coverability**: Is it the case that, for $n$ large enough, $\mathbb{P}_n(\text{eventually somebody on } q_f) = 1$?

**Almost-sure target**: Is it the case that, for $n$ large enough, $\mathbb{P}_n(\text{eventually everybody on } q_f) = 1$?
At every step:
- the next process to move is picked uniformly at random among all processes,
- its action is picked uniformly at random among all its available actions.

Almost-sure coverability: Is it the case that, for $n$ large enough, $\mathbb{P}_n(\text{eventually somebody on } q_f) = 1$?

Almost-sure target: Is it the case that, for $n$ large enough, $\mathbb{P}_n(\text{eventually everybody on } q_f) = 1$?

In the roundless case, almost-sure coverability can be stated as a deterministic property:
$q_f$ is covered with probability 1 iff, from every reachable configuration, some process can cover $q_f$.

Not true for round-based systems…
An annoying example

An example where, from any reachable configuration, $q_f$ can still be covered, but $q_f$ is not covered with probability 1.

(gadget to ensure that at least one process goes on U and one goes on D)
An annoying example

An example where, from any reachable configuration, $q_f$ can still be covered, but $q_f$ is not covered with probability 1.
First idea: Forbid processes to move up at different rates.

Balanced condition: there exists $m$ s.t., on every path of length $m$ of the automaton, there is exactly one increment.
First idea: Forbid processes to move up at different rates.

Balanced condition: there exists $m$ s.t., on every path of length $m$ of the automaton, there is exactly one increment.

Not enough!
« $m$ processes synchronize on some round » ~ return to zero of a balanced $(m - 1)$ - dimensional random walk

If $m$ is large, non-zero probability of never occurring after some point (proven for $m \geq 6$, conjectured for $m \geq 4$)
We can build a protocol where:

- the balanced condition is met
- \( q_f \) can be reached from all reachable configurations (for \( n \) large enough)
- \( \mathbb{P}(q_f \text{ covered}) < 1 \) for every \( n \)

**First idea:** Forbid processes to move up at different rates.

**Balanced condition:** there exists \( m \) s.t., on every path of length \( m \) of the automaton, there is exactly one increment.

*Not enough!*

\(< m \) processes synchronize on some round \( \sim \) return to zero of a balanced \((m - 1)\) - dimensional random walk

If \( m \) is large, non-zero probability of never occurring after some point (proved for \( m \geq 6 \), conjectured for \( m \geq 4 \))
A stronger restriction

**Almost-sure obstruction freedom (ASOF):** from any reachable configuration, any process left to play in isolation (all other processes are left idle) reaches $q_f$ with probability 1.
Almost-sure obstruction freedom (ASOF): from any reachable configuration, any process left to play in isolation (all other processes are left idle) reaches $q_f$ with probability 1.

For example, it is the case in Aspnes’ algorithm: any process acting in isolation will reach blank rounds.
Almost-sure obstruction freedom (ASOF): from any reachable configuration, any process left to play in isolation (all other processes are left idle) reaches $q_f$ with probability 1.

For example, it is the case in Aspnes’ algorithm: any process acting in isolation will reach blank rounds.

Proposition: Deciding whether a given protocol is ASOF is a PSPACE-complete problem.
Almost-sure obstruction freedom (ASOF): from any reachable configuration, any process left to play in isolation (all other processes are left idle) reaches $q_f$ with probability 1.

For example, it is the case in Aspnes’ algorithm: any process acting in isolation will reach blank rounds.

Proposition: Deciding whether a given protocol is ASOF is a PSPACE-complete problem.

Proposition: If a protocol is ASOF, then for every $n$, all agents end up in $q_f$ with probability 1 (almost-sure TARGET).

$q_f$ is a deadlocked state
Thanks for your attention!
Any questions?
A challenge: exponential lower bounds

Exponential lower bounds on the number of *active* rounds:
Several negative results

Post* included in Pre*
There exists N s.t., for all \( n \geq N \), one can cover \( q_f \) from every reachable configuration of size \( n \)

ALMOST-SURE
There exists N s.t., for all \( n \geq N \), \( \mathbb{P}_n(\text{covering } q_f) = 1 \)

every other property implies COVER

COVER
There exists N s.t., for all \( n \geq N \), one can cover \( q_f \)

LIMIT-SURE
\( \mathbb{P}_n(\text{covering } q_f) \to_{n \to \infty} 1 \)

Conjecture
But the implication holds if no initialization of registers!

implication holds

implication \textbf{does not} hold
**The conjecture**

*Conjecture:* In the following example, $P_n(\text{covering } q_f) \not\rightarrow_{n \to \infty} 1.$

Asymptotic probability that a process in the D region catches up with the highest process in the U region?
Simulations

Evolution de la probabilité (limite = 1 000 000, sample size = 1 000, p= 0,55)

Probabilité d’atteindre $q_f$