Checking Presence Reachability Properties on Parameterized Shared-Memory Systems

Nicolas Waldburger
PhD supervisors: Nathalie Bertrand, Nicolas Markey, Ocan Sankur
SynCoP23, 23/04/2023
Parameterized verification

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>$d_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Registers are *initialized* to value $d_0$

No atomic read/write combinations

A configuration:

<table>
<thead>
<tr>
<th>q</th>
<th>p</th>
<th>a</th>
<th>b</th>
<th>d₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>×2</td>
<td>×1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many processes are on each state. Content of the registers.
\( q \times 2 \quad p \times 1 \quad a \quad b \quad d_0 \) 

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

\( q \times 1 \quad p \times 1 \quad r \times 1 \quad a \quad b \quad a \)

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

\( q \times 1 \quad r \times 2 \quad a \quad b \quad a \)
Semantics

Initial configurations:

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

$(q, \text{write}_1(a), r)$

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

$q \times 1$ $r \times 2$ $a$ $b$ $a$

$Can be arbitrarily large$

$Registers are initialized to $d_0$

$q_0 \times n$ $d_0$ $d_0$ $d_0$

with $n \geq 0$ and $q_0$ the initial state
A small example

A single register

\[ q_0 \rightarrow B \rightarrow C \rightarrow A \rightarrow q_f \]

- **write(c)** from **A** to **q_f**
- **read(a)** from **q_f** to **A**
- **write(b)** from **A** to **C**
- **read(b)** from **C** to **q_f**
- **write(a)** from **q_f** to **A**
- **read(c)** from **C** to **A**
- **read(d_0)** from **B** to **q_f**
- **read(d_0)** from **B** to **C**
- **read(d_0)** from **B** to **q_f**
A small example

Two processes

Initial value

$q_0$ → write(c) → $A$

$read(d_0)$ → $B$

$read(c)$ → $C$

$write(b)$ → $q_f$

$read(b)$ → $C$

$write(a)$ → $C$

$read(a)$ → $q_f$
A small example

\[
dl_0 \quad q_0 \quad B \quad C \quad A \quad q_f
\]

- **Transition**:
  - \( \text{read}(d_0) \) from \( B \) to \( q_0 \)
  - \( \text{write}(c) \) from \( q_0 \) to \( A \)
  - \( \text{read}(a) \) from \( A \) to \( q_f \)
  - \( \text{write}(b) \) from \( A \) to \( C \)
  - \( \text{read}(b) \) from \( C \) to \( A \)
  - \( \text{write}(a) \) from \( A \) to \( C \) (loop)
  - \( \text{read}(d_0) \) from \( C \) to \( B \) (loop)

- **States**:
  - \( q_0 \)
  - \( B \)
  - \( C \)
  - \( A \)
  - \( q_f \)
A small example
A small example

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

\[ q_0 \quad \text{write}(c) \quad A \quad \text{read}(a) \quad q_f \]

\[ B \quad \text{read}(d_0) \quad \text{read}(c) \quad \text{write}(b) \quad C \quad \text{read}(b) \quad \text{write}(a) \]

\[ \text{read}(a) \quad \text{write}(b) \quad \text{read}(b) \]

\[ \text{read}(d_0) \quad \text{write}(c) \quad \text{read}(c) \]

\[ a \]
A small example

\[ q_0 \rightarrow \text{write}(c) \rightarrow A \]

\[ B \rightarrow \text{read}(d_0) \rightarrow C \]

\[ C \rightarrow \text{read}(d_0) \rightarrow \text{write}(a) \]

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

\[ q_f \text{ is covered } \checkmark \]
Reachability problems

Cover:
\[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^{\ast} \gamma, \gamma(q_f) > 0? \]

- Parameterized: arbitrarily many processes
- An initial configuration
- Execution
- A least one process on \( q_f \): “error state”
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\(^2\):**  \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \gamma \vDash \phi ? \)

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

\#q = number of processes on q

---

Reachability problems

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

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

PRP\textsuperscript{2}:
\[ \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^{*} \gamma, \gamma \models \phi \]

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

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

2. Inspired from CRP in: Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: 
Monotonicity

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

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

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

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

\[ \text{read}(a) \]
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: \( \exists n, \exists \gamma_0, \exists \rho: \gamma_0 \rightarrow^* \gamma, \; \gamma(q_f) > 0 \) ?
NP-completeness of COVER

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

Reduction from 3-SAT:

| \( x \) | \( d_0 \) |
| \( \neg x \) | \( d_0 \) |

Check \( x \):

- \( \text{read}_x(T) \) → \( \text{read}_{\neg x}(d_0) \)

Check \( \neg x \):

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

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

Reduction from 3-SAT:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \neg x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_0 )</td>
<td>( d_0 )</td>
</tr>
</tbody>
</table>

Check \( x \): \( \rightarrow read_x(T) \rightarrow read_{\neg x}(d_0) \)

Check \( \neg x \): \( \rightarrow read_{\neg x}(T) \rightarrow read_x(d_0) \)

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:
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 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* \((q_f \text{ may be reached from some configuration containing the state})\).
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 *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).

\[= \text{coverable}\]
\[= \text{backwards coverable}\]
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. For simplicity: 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.
For simplicity: the register is not initialized. Algorithm inspired from broadcast protocols

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.
For simplicity: 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 \(\checkmark\).
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.
For simplicity: 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: \( \phi = \bigvee_i t_{i,1} \land t_{i,2} \land \cdots \land t_{i,m_i}, \)

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

### 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
Round-based shared-memory systems

Model inspired by round-based algorithms from the literature\(^6\)\(^7\)\(^8\).

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

\[
\begin{array}{c|c|c|c|c}
\text{rounds} & \text{initial value} & \text{reg}_1[k] & \text{reg}_2[k] & \text{reg}_3[k] & \text{reg}_4[k] \\
\hline
0 & \ldots & \ldots & \ldots & \ldots & \ldots \\
1 & \ldots & \ldots & \ldots & \ldots & \ldots \\
2 & \ldots & \ldots & \ldots & \ldots & \ldots \\
3 & a & d_0 & d_0 & d_0 & d_0 \\
\end{array}
\]

Initial value

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:
## Semantics

\[
\begin{align*}
\vdash & \\
3 & \phantom{\vdash} p \times 1 \\
2 & \phantom{\vdash} b \\
1 & \phantom{\vdash} a \\
0 & \phantom{\vdash} d_0 \\
\end{align*}
\]

\[
\begin{align*}
\vdash & \\
& \phantom{\vdash} \vdash \\
3 & \phantom{\vdash} r \times 1 \\
2 & \phantom{\vdash} b \\
1 & \phantom{\vdash} a \\
0 & \phantom{\vdash} d_0 \\
\end{align*}
\]

Here with one register per round

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

\( p \times 1 \)

\( d_0 \)

\( b \)

\( a \)

\( d_0 \)

\( r \times 1 \)

\( q \times 2 \)

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

here with one register per round
Semantics

Initial configurations:

\[
\begin{array}{c}
3 \\
2 \\
1 \\
0
\end{array}
\]

\( p \times 1 \)

\[
\begin{array}{c}
3 \\
2 \\
1 \\
0
\end{array}
\]

\( q \times 3 \)

\( d_0 \)

\( a \)

\( d_0 \)

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

\[
\begin{array}{c}
3 \\
2 \\
1 \\
0
\end{array}
\]

\( p \times 1 \)

\[
\begin{array}{c}
3 \\
2 \\
1 \\
0
\end{array}
\]

\( q \times 2 \)

\( r \times 1 \)

\( d_0 \)

\( b \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( d_0 \)

\( q_0 \times n \)
## Abstraction

Initial configurations:

<table>
<thead>
<tr>
<th>:</th>
<th>( q \times )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( d_0 )</td>
</tr>
<tr>
<td>2</td>
<td>( b )</td>
</tr>
<tr>
<td>1</td>
<td>( a )</td>
</tr>
<tr>
<td>0</td>
<td>( d_0 )</td>
</tr>
</tbody>
</table>

\begin{align*}
(q, \text{write}(b), r), 1
\end{align*}

<table>
<thead>
<tr>
<th>:</th>
<th>( q \times )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>( d_0 )</td>
</tr>
<tr>
<td>2</td>
<td>( b )</td>
</tr>
<tr>
<td>1</td>
<td>( d_0 )</td>
</tr>
<tr>
<td>0</td>
<td>( d_0 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>:</th>
<th>( q_0 \times )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>( d_0 )</td>
</tr>
<tr>
<td>1</td>
<td>( d_0 )</td>
</tr>
<tr>
<td>0</td>
<td>( d_0 )</td>
</tr>
</tbody>
</table>

\( \ldots \)
An example of round-based register protocol

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

Increment round

- \( q_0 \)
- \( d_0 \)
- \( 0 \)
- \( 1 \)
- \( 2 \)

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

$q_0$ is the initial state.

- Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$q_0$</td>
<td>$d_0$</td>
</tr>
<tr>
<td>1</td>
<td>$q_0$</td>
<td>$d_0$</td>
</tr>
<tr>
<td>2</td>
<td>$q_0$</td>
<td>$d_0$</td>
</tr>
</tbody>
</table>

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

\[ q_0 \]

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

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

\[ \text{read}^{-1}(a) \]

\[ \text{read}^{-1}(d_0) \]

\[ \text{read}^{-1}(b) \]

\[ \text{read}^0(d_0) \]

\[ \text{read}^0(b) \]

\[ q_f \]

Increment round

\[ \vdots \]

\[ 2 \]

\[ 1 \]

\[ 0 \]

\[ q_0 \]

\[ A \]

\[ d_0 \]

\[ a \]

\[ d_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]

\[ q_0 \]
An example of round-based register protocol

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

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

Increment round

<table>
<thead>
<tr>
<th>round</th>
<th>symbol</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( q_0 )</td>
<td>( d_0 )</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>( d_0 )</td>
</tr>
</tbody>
</table>

\( a = \text{round 1} \)  
\( d_0 = \text{round 0} \)  
\( \text{round 2} \)
An example of round-based register protocol
An example of round-based register protocol

\[ q_0 \]

Increment round

\[ \begin{array}{c|c}
\vdots \\
2 & \vdots \\
1 & \begin{array}{c} q_0 \\ A \\ B \\ C \end{array} \\
0 & \begin{array}{c} q_0 \\ A \end{array} \\
\end{array} \]

- Red = round 0
- Blue = round 1
- Orange = round 2

\[ \begin{array}{c}
\text{write}(b) \\
\text{read}^{-1}(d_0) \\
\text{write}(a) \\
\text{read}^{-1}(b) \\
\text{read}^0(d_0) \\
\text{read}^0(b) \\
q_f
\end{array} \]
An example of round-based register protocol

increment round

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

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

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

\[ A \quad B \quad C \]

\[ q_f \]

= 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>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

= round 0

= round 1

= round 2
An example of round-based register protocol

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

Increment round

<table>
<thead>
<tr>
<th>Round</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>A, B, C</td>
</tr>
<tr>
<td>2</td>
<td>A, B, C, D</td>
</tr>
</tbody>
</table>

Symbols:
- $A$: Round 0
- $B$: Round 1
- $C$: Round 2

53
An example of round-based register protocol
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: \( \psi = \exists k (\#(q_1, k + 1) > 0 \land \text{reg}_i[k] = d) \lor \forall k \#(q_0, k) = 0'' \)

At some round, there is a process on state \( q_1 \), while register \( i \) of previous round has value \( d \)

no process is on \( q_0 \)
Complexity results

*Theorem* $^9$: Round-based COVER is PSPACE-hard.

---

Complexity results

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

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

10. W: *Checking Presence Reachability Properties on Parameterized Shared-Memory Systems*, submitted
Theorem\textsuperscript{9}: Round-based COVER is PSPACE-hard.

\textit{Theorem}\textsuperscript{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: $\theta_0 \ \theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6$

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

Witness execution: $\sigma_0^{\theta_0} \rightarrow \sigma_1^{\theta_1} \rightarrow \sigma_2^{\theta_2} \rightarrow \sigma_3^{\theta_3} \rightarrow \sigma_4^{\theta_4} \rightarrow \sigma_5^{\theta_5} \rightarrow \sigma_6^{\theta_6} \rightarrow \sigma_7^{\theta_7} \models \psi$

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

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

LaTeX code for the diagram:
```
\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node (theta0) at (0,0) {$\theta_0$};
  \node (theta1) at (1,1) {$\theta_1$};
  \node (theta2) at (1,2) {$\theta_2$};
  \node (theta3) at (0,3) {$\theta_3$};
  \node (theta4) at (1,4) {$\theta_4$};
  \node (theta5) at (2,3) {$\theta_5$};
  \node (theta6) at (3,0) {$\theta_6$};

  \draw[->] (theta0) .. controls (0.5,0.5) .. (theta1);
  \draw[->] (theta1) .. controls (1.5,2.5) .. (theta2);
  \draw[->] (theta2) .. controls (0.5,3.5) .. (theta3);
  \draw[->] (theta3) .. controls (1.5,4.5) .. (theta4);
  \draw[->] (theta4) .. controls (2.5,3.5) .. (theta5);
  \draw[->] (theta5) .. controls (3.5,2.5) .. (theta6);

  \node at (-1,4) {rounds};
  \node at (4,-1) {steps};
\end{tikzpicture}
\end{figure}
```
A non-deterministic polynomial-space algorithm
A non-deterministic polynomial-space algorithm

number of relevant rounds may be large…

storable in polynomial space?
sliding window on $v + 1$ rounds where $v$ is the highest $i$ such that some $\text{read}^{-i}(x)$ appears in the protocol

$v$ is assumed to be given in unary (here $v = 1$)
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
From this algorithm, we obtain exponential upper bounds on the number of processes and rounds needed.

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

Summary

- Two models in this talk: *roundless* register protocols and *round-based* register protocols.
- Properties studied are *reachability properties* which do not “count” processes. Two classical such problems are COVER and TARGET; PRP is a general class which encompasses these two problems.
- In the first model, despite its simplicity, PRP is NP-complete, but some restrictions make it PTIME.
- In the second model, PRP is PSPACE-complete, and similar restrictions do not decreases the complexity.

Future work

- Introducting stochastic schedulers and study almost-sure reachability (work in progress, some weird behaviors occur that make it very different from the roundless case)
- Weak memory
A challenge: exponential lower bounds

Exponential lower bounds on the number of rounds:
Exponential lower bounds on the number *active* rounds:
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:\

\[
\text{int } k := 0, \text{ bool } p \in \{0, 1\}, (\text{rg}_b[r])_{b \in \{0, 1\}, r \in \mathbb{N}} \text{ all initialized to no;}
\]

\[
\text{while true do}
\]

\[
\text{read from } \text{rg}_0[k] \text{ and } \text{rg}_1[k];
\]

\[
\text{if } \text{rg}_0[k] = \text{yes} \text{ and } \text{rg}_1[k] = \text{no} \text{ then } p := 0;
\]

\[
\text{else if } \text{rg}_0[k] = \text{no} \text{ and } \text{rg}_1[k] = \text{yes} \text{ then } p := 1;
\]

\[
\text{write yes to } \text{rg}_p[k];
\]

\[
\text{if } k > 0 \text{ then}
\]

\[
\text{read from } \text{rg}_{1-p}[k-1];
\]

\[
\text{if } \text{rg}_{1-p}[k-1] = \text{no} \text{ then return } p;
\]

\[
k := k+1;
\]
## 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>1</td>
<td>1</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>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 diagram illustrates the execution of the algorithm with 4 rounds. The states of the registers `reg₀[k]` and `reg₁[k]` are shown for each round, along with the values of `A`, `B`, and `C`. The values `0` and `1` indicate the state of each register 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>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>no</td>
<td>no</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>yes</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>writes</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>no</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
<th>reg₀[k]</th>
<th>reg₁[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no</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>
<tr>
<td>4</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

<table>
<thead>
<tr>
<th>rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

A B C

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

... yes yes
... no no
... no no
... no no
... no no
... no no
Example of execution of the algorithm
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></td>
<td>1</td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</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>
Example of execution of the algorithm

- **rounds**: 0, 1, 2, 3, 4
- **reads**: 0, 1, 1
- **A, B, C**: no, no, yes
- **reg0[k]**: no
- **reg1[k]**: yes
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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>yes</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

reads 0 → 1 ⇒ 1

…”
Example of execution of the algorithm

- **Rounds:** 0, 1, 2, 3, 4
- **writes:** yes, no, yes, yes, no
- **A, B, C:** 0, 1, 1
- **reg\(_0\)[k], reg\(_1\)[k]:** no, yes

The diagram illustrates the execution of an algorithm over multiple rounds, with updates indicated by 'writes' and values recorded in registers "reg\(_0\)[k]" and "reg\(_1\)[k]."
Example of execution of the algorithm

A  B  C

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

No winner on this round
Example of execution of the algorithm

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

**Explanation:**
- The diagram illustrates the execution of an algorithm with three variables, A, B, and C, and two registers, reg$_0^k$ and reg$_1^k$.
- The algorithm progresses through rounds, with each round indicating whether the conditions for A, B, and C are met (yes) or not (no).
- The diagram visually represents the progression and decision points for each round.
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>1</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</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>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>

\[ \text{reads} \]
Example of execution of the algorithm

rounds

... 4 3 2 1 0 ...

writes

no no no yes yes yes

A B C reg_0[k] reg_1[k]
Example of execution of the algorithm

\[ \text{reads} \]

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

<table>
<thead>
<tr>
<th>rounds</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... no no yes yes yes no no yes
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>1</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>0</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>yes</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>0</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>
Example of execution of the algorithm

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

reads

0

1

... reads...

no

yes

yes

yes

no

yes

yes

no

no

no

no

no

no

...
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></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></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></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

writes ... ...
Example of execution of the algorithm

\[ \text{reads} \]

<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></td>
<td></td>
<td></td>
<td>no</td>
<td>yes</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>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></td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Example of execution of the algorithm

A process getting to this round will convert to preference 1