Peterson’s mutual exclusion algorithm:

For process $i \in \{0, 1\}$:

```
while true:
    do non-critical things;
    flag$_i$ = true; turn := 1 - $i$;
    wait until (flag$_{1-i}$ == false or turn == $i$)
    do critical things;
    flag$_i$ = false;
```

Correctness = the processes are not in their critical section simultaneously
Peterson’s mutual exclusion algorithm:

For process \( i \in \{0,1\} \):

\[
\text{while } \text{true:} \\
\begin{align*}
\ n_i & \quad \text{do non-critical things;} \\
\ flag_i & \quad = \text{true;} \\
\ turn & \quad := 1 - i \\
\ \text{wait until} & \quad (\text{flag}_{1-i} = \text{false or turn} = \text{i}) \\
\ c_i & \quad \text{do critical things;} \\
\ flag_i & \quad = \text{false}
\end{align*}
\]

Correctness = the processes are not in their critical section simultaneously
= \( c_0 \) and \( c_1 \) cannot be covered simultaneously

Does a distributed system satisfy a requirement? Many thanks to Nathalie Bertrand for this slide.
Does Nicolas Waldburger satisfy distributed system requirement?

\[
\begin{align*}
\text{model} & \quad \ldots \quad \text{model-checking algorithm} \
\end{align*}
\]

\[\models \mathbf{AG}(\neg c_0 \lor \neg c_1)\]

Many thanks to Nathalie Bertrand for this slide
“Traditional” model checking: describe behavior of each process separately
⇒ fix number of processes beforehand
Issues with traditional Model Checking

“Traditional” model checking: describe behavior of each process separately
⇒ fix number of processes beforehand

- Scalability issue when the size of the system is large
Issues with traditional Model Checking

“Traditional” model checking: describe behavior of each process separately
⇒ fix number of processes beforehand

- Scalability issue when the size of the system is large
- What if I don’t know the number of agents beforehand?
“Traditional” model checking: describe behavior of each process separately
⇒ fix number of processes beforehand

- Scalability issue when the size of the system is large

- What if I don’t know the number of agents beforehand?

- Often undecidable problems…
Parameterized Verification

- Parameterized system = the number of participants is not fixed in advance
- System must be correct for any number of participants

→ New techniques than can be more efficient on large systems!
Many possible models
Many possible models

- We could say that all processes are identical, or that there is one leader and all others are followers.
Many possible models

- We could say that all processes are identical, or that there is one leader and all others are followers.
- How much computing power for a given process? Finite-state machines, pushdown machines, access to private variables…
Many possible models

- We could say that all processes are identical, or that there is one leader and all others are followers.

- How much computing power for a given process? Finite-state machines, pushdown machines, access to private variables...

- Means of communication:
  
  **Rendez-vous**
  
  Two processes must synchronize.

  **Broadcast**
  
  A process sends a message to its neighbors.

  **Shared memory**
  
  A process reads from the shared memory or writes to the shared memory.
Let’s focus on shared-memory systems

From now on, all processes are identical and described by a simple finite-state machine (no stack, no private memory…) where transitions interact with the shared memory.

Shared memory

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
A basic problem: coverability

**Coverability problem:** Input: A protocol $P$ (= an automaton) with an error state $q_f$.

**Question:** Does there exists a number of processes $n$ and an execution of the system with $n$ processes where one of them gets to $q_f$?
A basic problem: coverability

Coverability problem: *Input:* A protocol $P$ (an automaton) with an error state $q_f$. *Question:* Does there exist a number of processes $n$ and an execution of the system with $n$ processes where one of them gets to $q_f$?

Parameterized problem: if answer is no then the system is safe for every value of $n$. 
Atomic combinations are a bad idea

atomic read-write combination: a process can perform a read then a write and no one else can act in between
atomic combinations are a bad idea

atomic read-write combination: a process can perform a read then a write and no one else can act in between

Atomic combinations allow for leader election: (too) powerful model

In fact, as expressive as Petri Nets: coverability is EXPSPACE-complete… → let’s forbid atomic combinations
The model we obtain

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>
</table>

Initial value in the registers

A small example

A single register

\[ \text{read}(d_0) \rightarrow q_0 \rightarrow B \]

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

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

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

\[ \text{read}(b) \rightarrow \]

\[ \text{write}(b) \rightarrow q_f \]
A small example

Two processes

Initial value

$q_0 \xrightarrow{\text{read}(d_0)} B$

$\rightarrow \text{write}(c) \rightarrow A$

$A \xrightarrow{\text{read}(c)} C$

$\xrightarrow{\text{write}(a)} \rightarrow q_f$

$B \xrightarrow{\text{read}(d_0)}$

$D_0$

Initial value

$(a)$

$(b)$

$(c)$
A small example

\[ d_0 \]

\[
\begin{align*}
q_0 & \xrightarrow{\text{read}(d_0)} B \\
B & \xrightarrow{\text{read}(d_0)} q_0
\end{align*}
\]

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

![Diagram]

- **States:**
  - $q_0$ (initial state)
  - $B$, $C$, $A$, $q_f$

- **Transitions:**
  - $q_0 \xrightarrow{\text{write}(c)} A$
  - $q_0 \xrightarrow{\text{read}(d_0)} B$
  - $A \xrightarrow{\text{read}(a)} q_f$
  - $A \xrightarrow{\text{read}(b)} C$
  - $C \xrightarrow{\text{write}(a)} q_f$

- **Labels:**
  - **Read:** $d_0$
  - **Write:** $c$, $b$

- **Variables:**
  - $a$, $b$, $c$
A small example

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

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

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

\( q_f \) is covered \( \checkmark \)
Cloning processes

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

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.
Cloning processes

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.
Cloning processes

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

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

**Copycat property**: Where we can have one process, we can have many processes.
Cloning processes

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

**Copycat property:** Where we can have one process, we can have many processes.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
Complexity of COVER

COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if \( d_0 \) cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.
COVER is decidable in polynomial time if $d_0$ cannot be read (= no initialization) using a simple saturation algorithm that computes all coverable states.

COVER is in PTIME in this case, by contrast it is:

- **NP**-complete if $\text{read}(d_0)$ transitions are allowed,
- **PSPACE**-complete if $n$ is given as input.

Parameterized problem is much easier!
A too simple model?

In this model, many parameterized questions are between PTIME and NP.
In this model, many parameterized questions are between \textsc{PTIME} and \textsc{NP}.

However, the model is limited; many shared-memory algorithms require more expressiveness!
One such example: round-based algorithms.
We want to model round-based distributed algorithms\textsuperscript{234} that look like this:

\begin{center}
\begin{itemize}
\item Asynchronous rounds
\item Each round has its own set of shared registers $\rightarrow$ unbounded memory!
\item Read and write to registers of nearby rounds only
\end{itemize}
\end{center}

\begin{itemize}
\item for $k = 0$ to $\infty$ do
\item \hspace{1em} read register 0 of round $k$ ;
\item \hspace{1em} if read value is a then ... else ... ;
\item \hspace{1em} write to register 1 of round $k$ ;
\item \hspace{1em} ... \\
\item \hspace{1em} read register 0 of round $k - 1$ ;
\item end
\end{itemize}


4. Raynal, M., Stainer, J.: \textit{A Simple Asynchronous Shared Memory Consensus Algorithm Based on Omega and Closing Sets}. CISIS, 2012
An example

Increment round

write(a)

$q_0$

read ($d_0$)

Increment round

write(b)

read (a)

$q_f$

read$^{-1}$(b)

Increment round

read ($d_0$)
Write to register of current round of the process

Write (a)

Increment round

$q_0$

Read (a)

$q_f$

Read from register of current round

Read from register one round below

Increment round

$\textit{read} (d_0)$

Send process to next round

Increment round

$\textit{write} (b)$

Increment round

$\textit{read}^{-1} (b)$

Increment round

$\textit{read} (d_0)$
An example

Two processes, both on round 0

$\begin{pmatrix}
\vdots \\
2 & d_0 \\
1 & d_0 \\
0 & d_0
\end{pmatrix}$
An example

\[
\begin{align*}
\text{write}(a) & \quad \text{write}(b) \\
q_0 & \quad q_f \\
\text{read}^{-1}(b) & \\
\text{read} (d_0) & \quad \text{read} (d_0) \\
\text{Increment round} & \quad \text{Increment round} \\
\end{align*}
\]

\[
\begin{align*}
q_0 & \\
q_f & \\
\end{align*}
\]

\[
\begin{array}{ccc}
\vdots & \vdots \\
2 & d_0 \\
1 & d_0 \\
0 & d_0 \\
\end{array}
\]
An example

\[
\begin{align*}
\text{write}(a) & \rightarrow q_0 \\
\text{read } (d_0) & \rightarrow q_0 \\
\text{Increment round} & \rightarrow q_0 \\
\text{read } (a) & \rightarrow q_f \\
\text{read}^{-1}(b) & \rightarrow q_f \\
\end{align*}
\]

\[
\begin{array}{c|c}
\text{Round} & d_n \\
\hline
2 & d_0 \\
1 & d_0 \\
0 & d_0 \\
\end{array}
\]
An example

\[ q_0 \xrightarrow{\text{write}(a)} q_1 \xrightarrow{\text{read} \ (d_0)} q_0 \xrightarrow{\text{write}(b)} q_f \xrightarrow{\text{read}^{-1}(b)} q_0 \]

\[ \vdots \]

\begin{array}{c|c}
2 & d_0 \\
1 & d_0 \\
0 & d_0 \\
\end{array}
An example

write(a)

$q_0$

Increment round

write(b)

read (d₀)

Increment round

$q_f$

read (a)

Increment round

read⁻¹(b)

Increment round

$q_f$

read (d₀)

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$

Increment round

$q_f$
\textbf{An example}

\begin{equation}
\begin{aligned}
\text{write}(a) & \rightarrow q_0 \\
\text{read } (d_0) & \rightarrow q_0 \\
\text{Increment round} & \rightarrow q_0 \\
\text{read } (d_0) & \rightarrow q_f \\
\text{write}(b) & \rightarrow q_f \\
\end{aligned}
\end{equation}

\begin{itemize}
\item \vdots
\item 2 \quad d_0
\item 1 \quad d_0
\item 0 \quad d_0
\end{itemize}
An example

\[ \begin{align*}
&0 \xrightarrow{\text{write}(a)} q_0 \\
&0 \xrightarrow{\text{write}(b)} 0 \\
&q_0 \xrightarrow{\text{read} (d_0)} q_0 \\
&2 \xrightarrow{\text{read} (a)} q_0 \\
&2 \xrightarrow{\text{read}^{-1}(b)} q_f
\end{align*} \]
An example

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

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

\[
\begin{align*}
\text{Increment round} & \quad \text{write}(a) \\
q_0 & \quad \Rightarrow \quad \text{Increment round} & \quad \text{write}(b) \\
\text{read } (d_0) & \quad \downarrow \quad \text{read } (d_0) \\
2 & \quad \text{read } (a) & \quad \text{read}^{-1}(b) \\
\text{Increment round} & \quad \downarrow \quad \downarrow \\
q_f & \quad \vdots
\end{align*}
\]

\[
\begin{align*}
& \vdots \\
& 2 \quad d_0 \\
& 1 \quad b \\
& 0 \quad a
\end{align*}
\]
An example

write(a)

$q_0$

Increment round

read ($d_0$)

read (a)

$q_f$

Increment round

read$^{-1}$(b)

Increment round

read ($d_0$)

$q_0$

write(b)

:  
:  
2  $d_0$
1  b
0  a

$q_0$

$q_f$
An example

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

\[
\begin{array}{c}
\vdots \\
2 \quad a \\
1 \quad b \\
0 \quad a \\
\end{array}
\]
An example

\[ \begin{array}{c}
\vdots \\
2 & a \\
1 & b \\
0 & a \\
\end{array} \]
An example

\[
\begin{array}{c}
q_0 \\
\text{read} (d_0) \\
\text{read} (d_0) \\
\text{write} (a) \\
\text{Increment round} \\
\end{array}
\xrightarrow{\text{Increment round}}
\begin{array}{c}
q_f \\
\text{read}^{-1}(b) \\
\text{read} (a) \\
\text{write} (b) \\
\end{array}
\]

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

Increment round

Increment round

write (a)

read (d₀)

read (a)

read⁻¹ (b)

write (b)
An example

$q_f$ is coverable

$q_0$  

- Increment round
- read (d₀)
- write (a)

$q_f$  

- Increment round
- read⁻¹ (b)

$q_f$ is coverable

Increment round
read (d₀)
write (b)
read⁻¹ (b)
read (a)
An example

- **Increment round**
  - **Write (a)**
  - **Write (b)**

- **Read (d₀)**

- **Read (a)**

- **Read⁻¹ (a)**

- **q₀**

- **q_f**

- **a is written to even rounds only**

- **q_f is not coverable**

- **a must be read two rounds in a row**
Theorem $^5$: COVER is PSPACE-complete.
Conclusion

General aim: automated methods for verification of distributed systems using model checking.

Parameterized verification:
• Systems of arbitrary number of participants
• If algorithm says yes, then the system is correct regardless of the number of participants
• Efficient techniques thanks to copycat properties

In this talk:
• Simple model for shared-memory systems with finite memory
• More complex model for round-based systems
General aim: automated methods for verification of distributed systems using model checking.

Parameterized verification:
• Systems of arbitrary number of participants
• If algorithm says yes, then the system is correct regardless of the number of participants
• Efficient techniques thanks to copycat properties

In this talk:
• Simple model for shared-memory systems with finite memory
• More complex model for round-based systems

Thanks for your attention! Any questions?