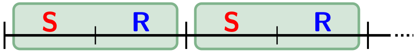
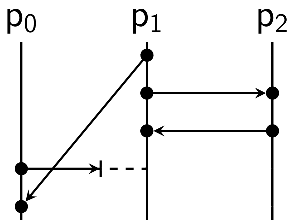
An Automata Based Approach for Synchronizable Mailbox Communication

Romain Delpy, Anca Muscholl, Grégoire Sutre

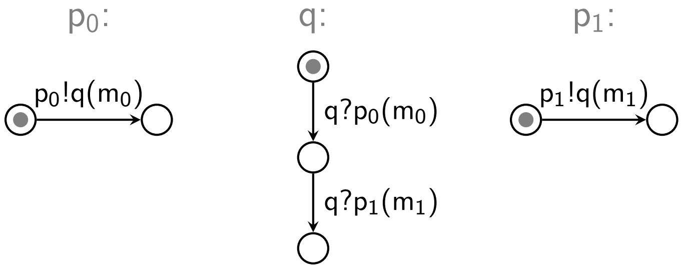
Univ. of Bordeaux, France

CONCUR 2024, Calgary

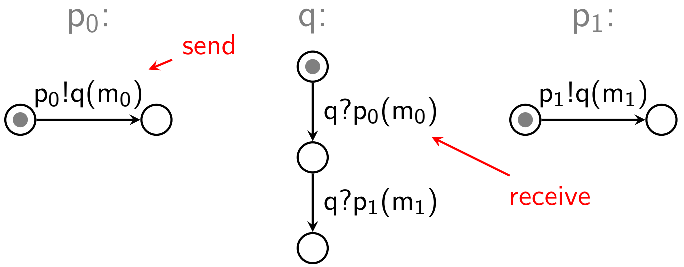




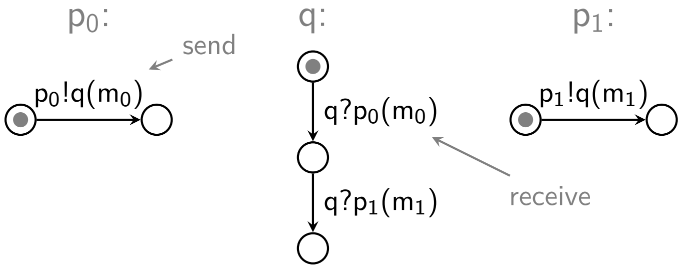
Mailbox semantics



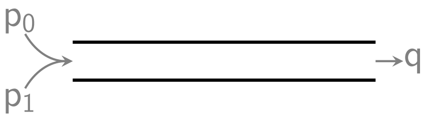
Mailbox semantics



Mailbox semantics

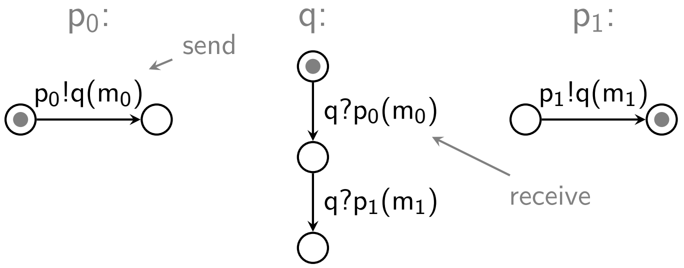


Mailbox:

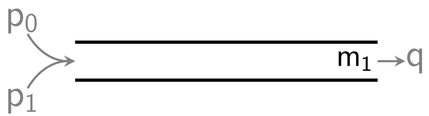


One *FIFO* channel per process.

Mailbox semantics

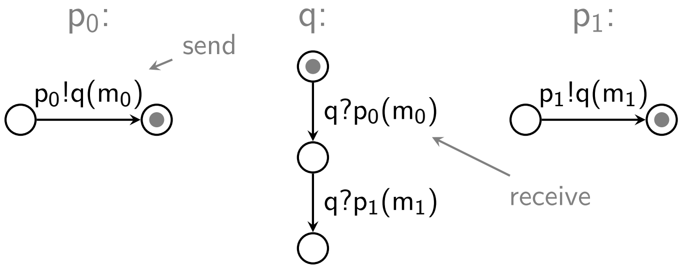


Mailbox:

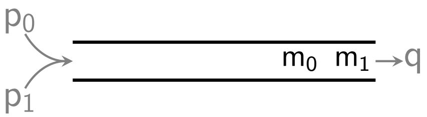


One *FIFO* channel per process.

Mailbox semantics

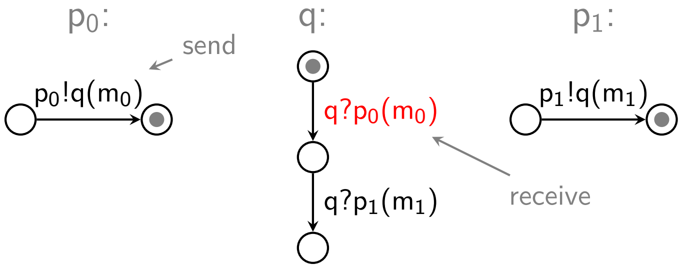


Mailbox:

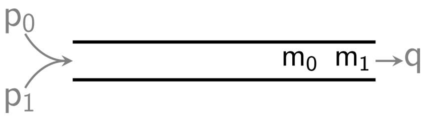


One *FIFO* channel per process.

Mailbox semantics



Mailbox:



One *FIFO* channel per process.

Mailbox executions

Executions that are possible for peer-to-peer communication may not be possible for mailbox.

Restriction

State reachability is **undecidable** [Brand-Zafiropulo 1983]:

Requires restrictions.

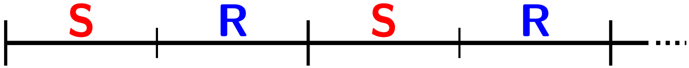
Restriction

State reachability is **undecidable** [Brand-Zafiropulo 1983]:

Requires restrictions.

Exchange: Sequence of actions with all sends before receives.

Usual for round-based systems (distributed algorithms)



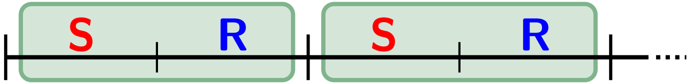
Restriction

State reachability is **undecidable** [Brand-Zafiropulo 1983]:

Requires restrictions.

Exchange: Sequence of actions with all sends before receives.

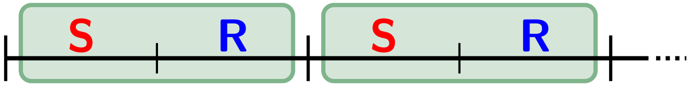
Usual for round-based systems (distributed algorithms)



Synchronizability

A CFM is synchronizable if every execution can be reordered into a sequence of exchanges (**no message split**).

Synchronizability



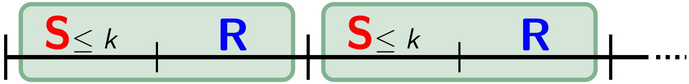
Synchronizability $\implies$ State reachability decidable?

Synchronizability decidable?

k-synchronizability

k-**exchanges** [Bouajjani et al. 2018]: Exchanges with at most *k* sends.

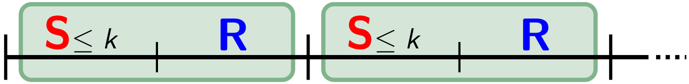
A CFM is *k*-**synchronizable** if every execution can be reordered into a sequence of *k*-exchanges (no message split).



k-synchronizability

k-**exchanges** [Bouajjani et al. 2018]: Exchanges with at most k sends.

A CFM is *k*-**synchronizable** if every execution can be reordered into a sequence of k-exchanges (no message split).



Prev. results [Bouajjani et al. 2018,Di Giusto et al. 2020/2021]

- Reachability is decidable under k-synchronizability (PSPACE).
- For fixed k, "is the CFM k-synchronizable?" is PSPACE.
- "Is there some k such that the CFM is k-synchronizable?" is decidable (no complexity).

Warning: Slightly different definition for synchronizability.

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

- $\mathsf{p!q_0(m_0)}\,\mathsf{p!q_1(m_1)}$
- $\mathsf{p_0!q(m_0)}\,\mathsf{p_1!q(m_1)}$

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

- $p!q_0(m_0)\, p!q_1(m_1)\, q_0?p(m_0)\, q_1?p(m_1)$
- $p_0!q(m_0)\, p_1!q(m_1)\, q?p_0(m_0)\, q?p_1(m_1)$

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

Marked send sequence: *unmatched* sends events are marked.

$$\mathsf{p}_0!\mathsf{q}(\mathsf{m}_0)\ \overline{\mathsf{p}_1!\mathsf{q}(\mathsf{m}_1)}$$

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

Marked send sequence: *unmatched* sends events are marked.

$$\mathsf{p_0!q(m_0)}\ \overline{\mathsf{p_1!q(m_1)}}$$

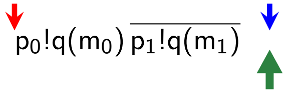
Reachability algorithm: Guess the **middle global state** [Di Giusto et al. 2020] $\longrightarrow$ exp-size automaton, PSPACE

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

Marked send sequence: *unmatched* sends events are marked.



Reachability algorithm: Guess the **middle global state** [Di Giusto et al. 2020] $\longrightarrow$ exp-size automaton, PSPACE

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

Marked send sequence: *unmatched* sends events are marked.

$$p_0!q(m_0)\ \overline{p_1!q(m_1)}\qquad q?p_0(m_0)$$

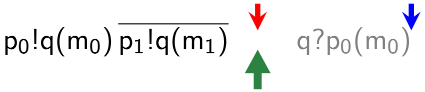
Reachability algorithm: Guess the **middle global state** [Di Giusto et al. 2020] $\longrightarrow$ exp-size automaton, PSPACE

Reachability

Exchange normal form

In mailbox semantics, every exchange is equivalent to the sequence where receives are **in the same order** as the sends.

Marked send sequence: *unmatched* sends events are marked.


$p_0!q(m_0)\ \overline{p_1!q(m_1)}\quad q?p_0(m_0)$

Reachability algorithm: Guess the **middle global state** [Di Giusto et al. 2020] $\longrightarrow$ exp-size automaton, PSPACE

Checking synchronizability

What does not synchronizable mean?

Execution that cannot be reordered into a sequence of exchanges:

Checking synchronizability

What does not synchronizable mean?

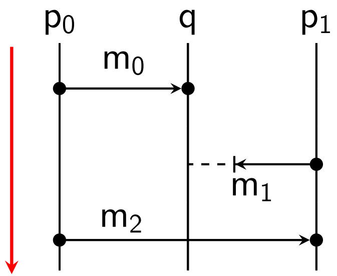
Execution that cannot be reordered into a sequence of exchanges:

$\mathsf{p}_1!\mathsf{p}_0(\mathsf{m}_0)\ \mathsf{p}_1!\mathsf{p}_2(\mathsf{m}_1)\ \mathsf{p}_2?\mathsf{p}_1(\mathsf{m}_1)\ \mathsf{p}_2!\mathsf{p}_1(\mathsf{m}_2)\ \mathsf{p}_1?\mathsf{p}_2(\mathsf{m}_2)\ \mathsf{p}_0!\mathsf{p}_1(\mathsf{m}_3)\ \mathsf{p}_0?\mathsf{p}_1(\mathsf{m}_0)$

Mailbox MSC

Message Sequence Charts (MSC)

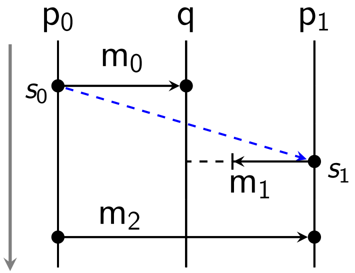
Partial-order representation of executions:
process order + message arcs .
Two executions are **equivalent** if they have the same MSC.



Mailbox MSC

Mailbox Message Sequence Charts (MSC)

Partial-order representation of executions:
process order + message arcs + **mailbox order**.
Two executions are **equivalent** if they have the same MSC.



Two sends to the same process, s_0 and s_1, are *mailbox-ordered* if:

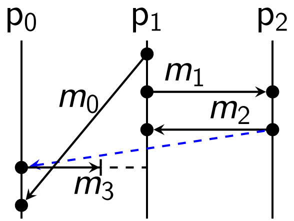
- s_0 is matched (with r_0)
- s_1 is unmatched, or is matched with r_1 and $r_0 < r_1$

Checking synchronizability

What does not synchronizable mean?

Execution that cannot be reordered into a sequence of exchanges:

$p_1!p_0(m_0)\ p_1!p_2(m_1)\ p_2?p_1(m_1)\ p_2!p_1(m_2)\ p_1?p_2(m_2)\ p_0!p_1(m_3)\ p_0?p_1(m_0)$

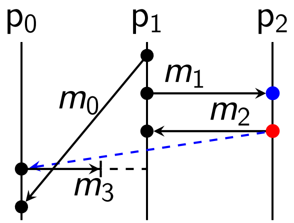


Checking synchronizability

What does not synchronizable mean?

Execution that cannot be reordered into a sequence of exchanges:

$p_1!p_0(m_0)\ p_1!p_2(m_1)\ p_2?p_1(m_1)\ p_2!p_1(m_2)\ p_1?p_2(m_2)\ p_0!p_1(m_3)\ p_0?p_1(m_0)$



- atomic sequence
- a **receive** before a **send** on some process

Atomicity

A sequence is **atomic** if no equivalent sequence can be divided into smaller sequences.

Atomicity

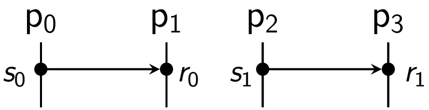
A sequence is **atomic** if no equivalent sequence can be divided into smaller sequences.

- message $p_0!p_1(m)\, p_1?p_0(m)$ **atomic**

Atomicity

A sequence is **atomic** if no equivalent sequence can be divided into smaller sequences.

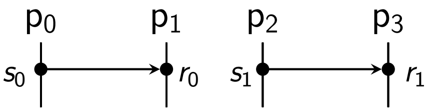
- message $\mathrm{p_0!p_1(m)\,p_1?p_0(m)}$ **atomic**
- $s_0\ s_1\ r_1\ r_0$



Atomicity

A sequence is **atomic** if no equivalent sequence can be divided into smaller sequences.

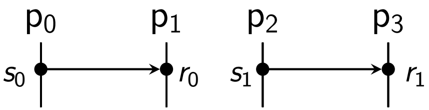
- message $\mathrm{p_0!p_1(m)\,p_1?p_0(m)}$ **atomic**
- $s_0\ s_1\ r_1\ r_0\ \equiv\ s_0\ r_0\ s_1\ r_1$ **non-atomic**



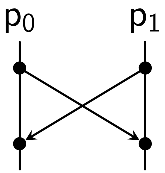
Atomicity

A sequence is **atomic** if no equivalent sequence can be divided into smaller sequences.

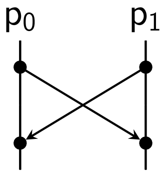
- message $\mathrm{p}_0!\mathrm{p}_1(\mathrm{m})\,\mathrm{p}_1?\mathrm{p}_0(\mathrm{m})$ **atomic**
- $s_0\ s_1\ r_1\ r_0 \ \equiv\ s_0\ r_0\ s_1\ r_1$ **non-atomic**



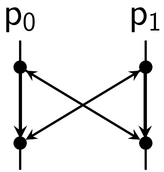
- *concurrent* messages **atomic**



Atomicity: A graphical approach

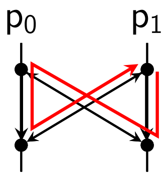


Atomicity: A graphical approach



Graph: added backward message arcs

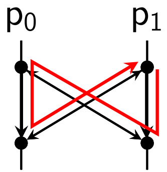
Atomicity: A graphical approach



Graph: added backward message arcs

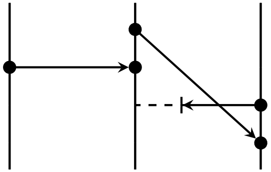
A sequence is **atomic** iff its graph is **strongly connected**.

Atomicity: A graphical approach

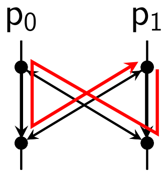


Graph: added backward message arcs

A sequence is **atomic** iff its graph is **strongly connected**.

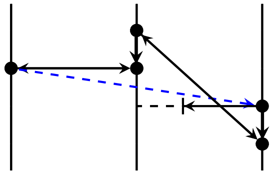


Atomicity: A graphical approach

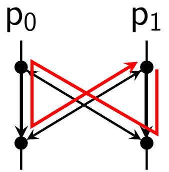


Graph: added backward message arcs

A sequence is **atomic** iff its graph is **strongly connected**.

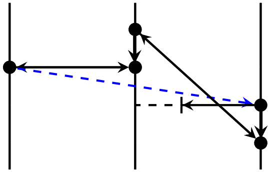


Atomicity: A graphical approach



Graph: added backward message arcs

A sequence is **atomic** iff its graph is **strongly connected**.



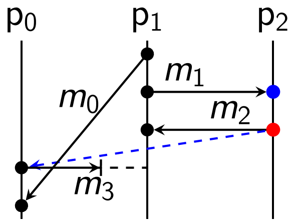
Atomicity of exchanges

The language of *marked send sequences* of atomic exchanges is **regular** (exp-size automaton built *on-the-fly*, PSPACE).

Checking synchronizability

What does not synchronizable mean?

Execution that cannot be reordered into a sequence of exchanges:

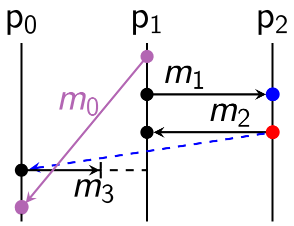


- atomic sequence
- a **receive** before a **send** on some process

Checking synchronizability

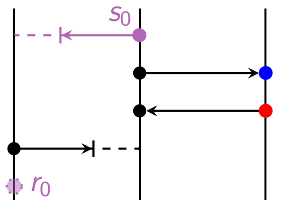
What does not synchronizable mean?

Execution that cannot be reordered into a sequence of exchanges:



- atomic sequence
- a **receive** before a **send** on some process

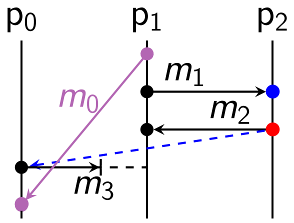
Removing last action makes it synchronizable!



Checking synchronizability

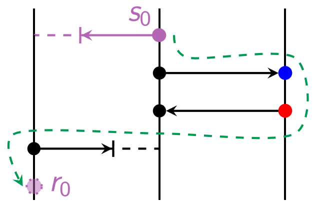
What does not synchronizable mean?

Execution that cannot be reordered into a sequence of exchanges:



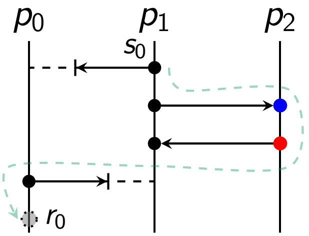
- atomic sequence
- a **receive** before a **send** on some process

Removing last action makes it synchronizable!



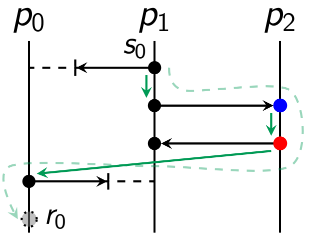
Need to check a path going from *first send action* s_0 through a receive then a send and ending before the future receive r_0

Path between exchanges



Need a way to connect path between *atomic* exchanges

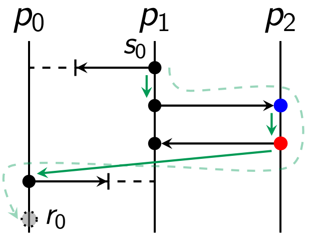
Path between exchanges



Need a way to connect path between *atomic* exchanges

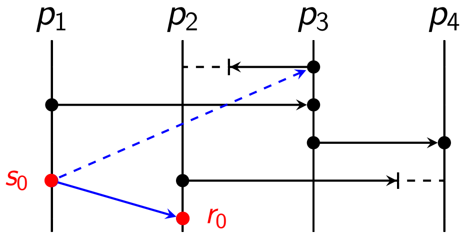
in/out-points: *out-point* at end of exchange, *in-point* at start of next exchange, there must be an order between out-point and in-point.

Path between exchanges



Need a way to connect path between *atomic* exchanges

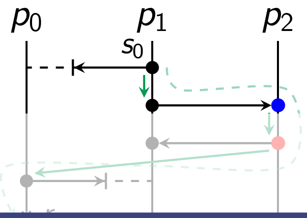
in/out-points: *out-point* at end of exchange, *in-point* at start of next exchange, there must be an order between out-point and in-point.



Not always first send that gets "unmatched" from last action.

Need to check for validity while guessing the sequence of exchanges. . .

Path between exchanges

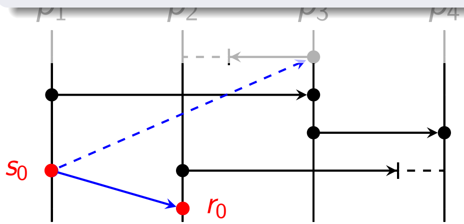


Need a way to connect path between *atomic* exchanges

in/out-points: *out-point* at end of exchange, *in-point* at start of next exchange, there must be an order

Synchronizability

Checking if a given CFM is synchronizable for mailbox semantics is PSPACE-complete



Not always first send that gets "unmatched" from last action.

Need to check for validity while guessing the sequence of exchanges. . .

Monitoring

Centralized Monitoring

For a CFM $\mathcal{A}$ and a property P, can we construct an automaton $\mathcal{B}$ accepting executions of $\mathcal{A}$ respecting P?

Monitoring

Centralized Monitoring

For a CFM $\mathcal{A}$ and a property P, can we construct an automaton $\mathcal{B}$ accepting executions of $\mathcal{A}$ respecting P?

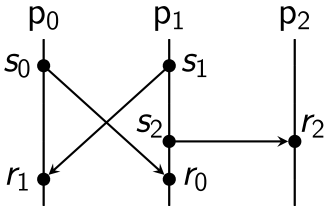
SR-monitoring: Constructing an automaton that accept executions that are synchronizable.

Monitoring

Centralized Monitoring

For a CFM $\mathcal{A}$ and a property P, can we construct an automaton $\mathcal{B}$ accepting executions of $\mathcal{A}$ respecting P?

SR-monitoring: Constructing an automaton that accept executions that are synchronizable.



Harder than it seems.

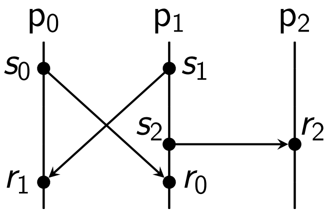
$s_0\ s_1\ r_1\ s_2\ r_2\ r_0$

Monitoring

Centralized Monitoring

For a CFM $\mathcal{A}$ and a property P, can we construct an automaton $\mathcal{B}$ accepting executions of $\mathcal{A}$ respecting P?

SR-monitoring: Constructing an automaton that accept executions that are synchronizable.



Harder than it seems.

$$s_0\ s_1\ r_1\ s_2\ r_2\ r_0$$
$$|||$$
$$s_0\ s_1\ s_2\ r_1\ r_2\ r_0$$

SR-monitoring

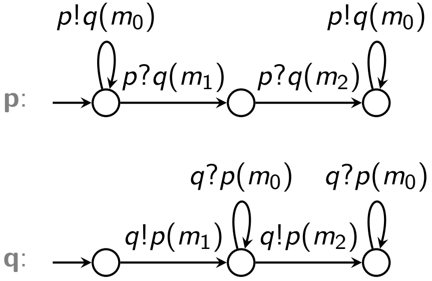
SR-monitoring

For a CFM $\mathcal{A}$, it is not possible to monitor the property "*is the execution of $\mathcal{A}$ synchronizable*".

SR-monitoring

SR-monitoring

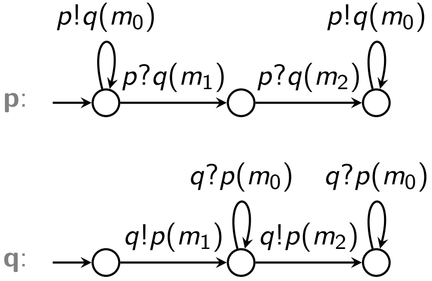
For a CFM $\mathcal{A}$, it is not possible to monitor the property "*is the execution of $\mathcal{A}$ synchronizable*".

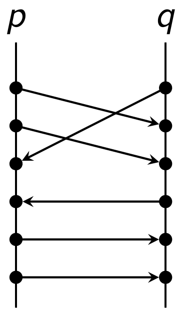


SR-monitoring

SR-monitoring

For a CFM $\mathcal{A}$, it is not possible to monitor the property "*is the execution of $\mathcal{A}$ synchronizable*".

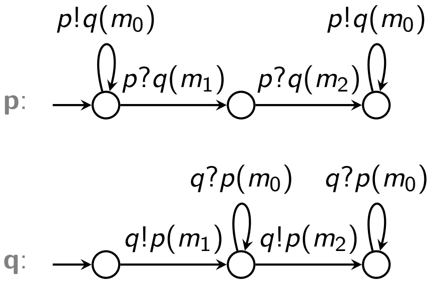


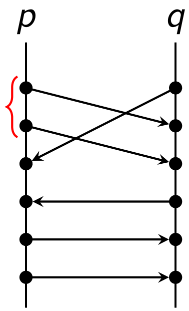


SR-monitoring

SR-monitoring

For a CFM $\mathcal{A}$, it is not possible to monitor the property "*is the execution of $\mathcal{A}$ synchronizable*".

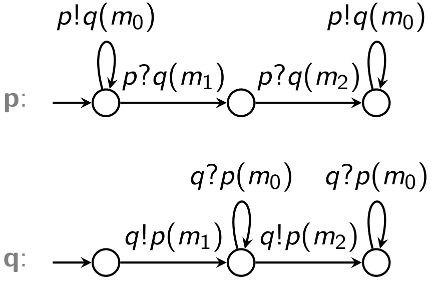


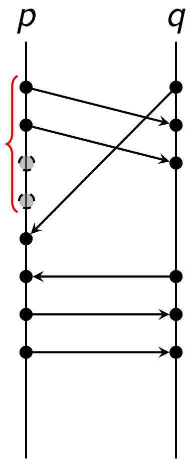


SR-monitoring

SR-monitoring

For a CFM $\mathcal{A}$, it is not possible to monitor the property "*is the execution of $\mathcal{A}$ synchronizable*".

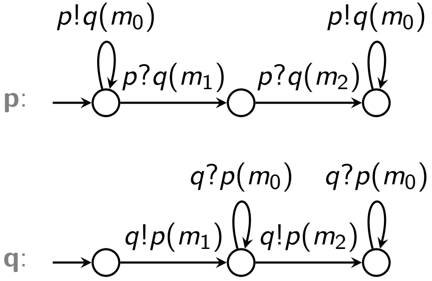


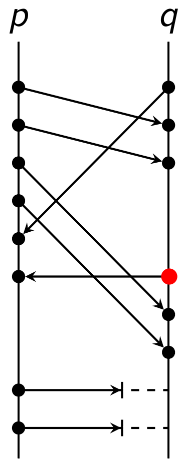


SR-monitoring

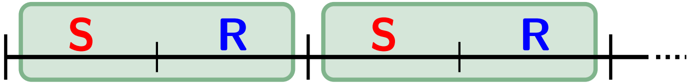
SR-monitoring

For a CFM $\mathcal{A}$, it is not possible to monitor the property "*is the execution of $\mathcal{A}$ synchronizable*".



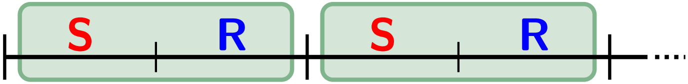


Conclusion



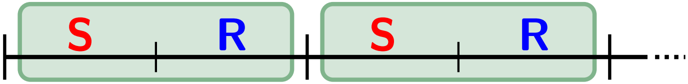
- Checking synchronizability PSPACE-complete.

Conclusion



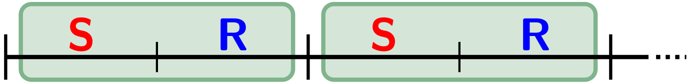
- Checking synchronizability PSPACE-complete.
- Model checking for a subclass of regular properties under synchronizability PSPACE-complete.

Conclusion



- Checking synchronizability PSPACE-complete.
- Model checking for a subclass of regular properties under synchronizability PSPACE-complete.
- Existence of k such that CFM is k-synchronizable is PSPACE-complete.

Conclusion

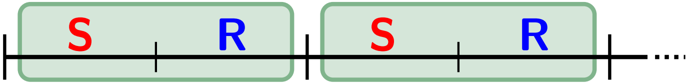


- Checking synchronizability PSPACE-complete.
- Model checking for a subclass of regular properties under synchronizability PSPACE-complete.
- Existence of k such that CFM is k-synchronizable is PSPACE-complete.
- SR-Monitoring is Not possible.

Future work:

- Analysing real programs in `Rust` (internship), finding restrictions for better complexity.

Conclusion

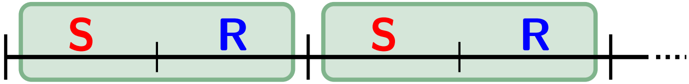


- Checking synchronizability PSPACE-complete.
- Model checking for a subclass of regular properties under synchronizability PSPACE-complete.
- Existence of k such that CFM is k-synchronizable is PSPACE-complete.
- SR-Monitoring is Not possible.

Future work:

- Analysing real programs in `Rust` (internship), finding restrictions for better complexity.
- Synthesis of CFM from specification given by sets of MSCs.

Conclusion

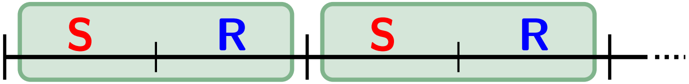


- Checking synchronizability PSPACE-complete.
- Model checking for a subclass of regular properties under synchronizability PSPACE-complete.
- Existence of k such that CFM is k-synchronizable is PSPACE-complete.
- SR-Monitoring is Not possible.

Future work:

- Analysing real programs in `Rust` (internship), finding restrictions for better complexity.
- Synthesis of CFM from specification given by sets of MSCs.
- Finding subclass of system where SR-monitoring possible.
- Distributed monitoring of synchronizability.

Conclusion



- Checking synchronizability PSPACE-complete.
- Model checking for a subclass of regular properties under synchronizability PSPACE-complete.
- Existence of k such that CFM is k-synchronizable is PSPACE-complete.
- SR-Monitoring is Not possible.

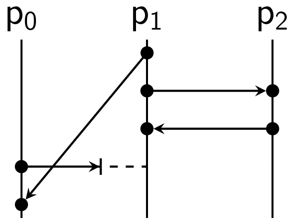
Future work:

- Analysing real programs in `Rust` (internship), finding restrictions for better complexity.
- Synthesis of CFM from specification given by sets of MSCs.
- Finding subclass of system where SR-monitoring possible.
- Distributed monitoring of synchronizability.

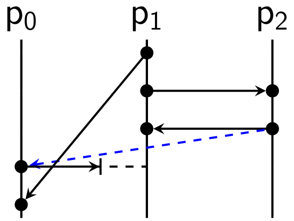
THANK YOU

Difference of synchronizability

Weak-Synchronizability [Di Giusto et. al. 2020]

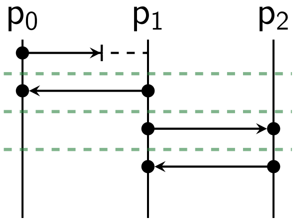


Synchronizability [Delpy et. al. 2024]



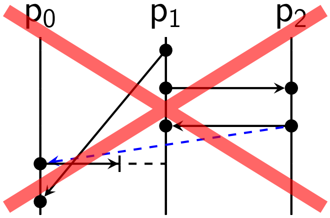
Difference of synchronizability

Weak-Synchronizability [Di Giusto et. al. 2020]



Weak-synchronizable

Synchronizability [Delpy et. al. 2024]



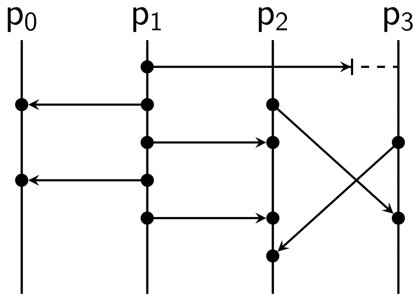
Not synchronizable

Atomicity: Automata check

Need a bounded technique: **Path labelling:** Ordered list of elements in the *marked sends* sequence displaying a path.

Atomicity: Automata check

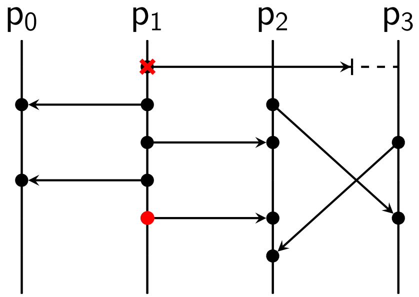
Need a bounded technique: **Path labelling:** Ordered list of elements in the *marked sends* sequence displaying a path.



$p_2!p_3$ $\overline{p_1!p_3}$ $p_1!p_0$ $p_1!p_2$ $p_1!p_0$ $p_1!p_2$ $p_3!p_2$

Atomicity: Automata check

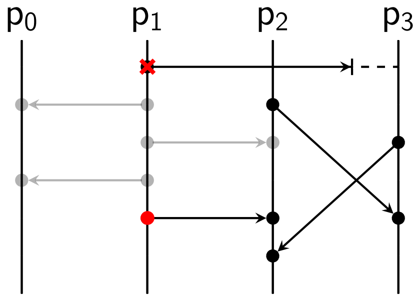
Need a bounded technique: **Path labelling:** Ordered list of elements in the *marked sends* sequence displaying a path.



$p_2!p_3$ $\overline{p_1!p_3}$ $p_1!p_0$ $p_1!p_2$ $p_1!p_0$ $p_1!p_2$ $p_3!p_2$

Atomicity: Automata check

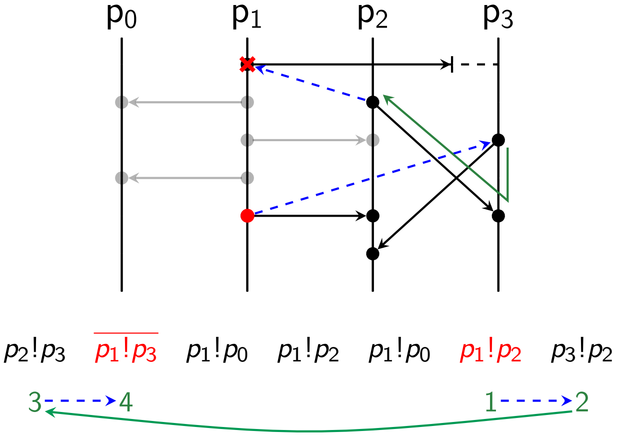
Need a bounded technique: **Path labelling:** Ordered list of elements in the *marked sends* sequence displaying a path.



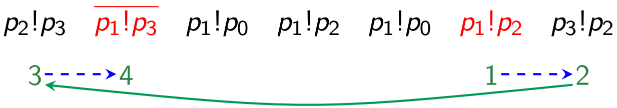
$p_2!p_3$ $\overline{p_1!p_3}$ $p_1!p_0$ $p_1!p_2$ $p_1!p_0$ $p_1!p_2$ $p_3!p_2$

Atomicity: Automata check

Need a bounded technique: **Path labelling:** Ordered list of elements in the *marked sends* sequence displaying a path.



Atomicity: Automata check (cont.)



Exists path labelling of **linear** size in the number of processes if path in augmented msc.

Atomicity: Automata check (cont.)

$$\begin{tikzcd}
p_2!p_3 & \overline{p_1!p_3} & p_1!p_0 & p_1!p_2 & p_1!p_0 & p_1!p_2 & p_3!p_2 \\
3 \arrow[r, dashed] & 4 & & & & 1 \arrow[r, dashed] & 2 \arrow[llllll, bend left]
\end{tikzcd}$$

Exists path labelling of **linear** size in the number of processes if path in augmented msc.

A sequence is atomic *iff*

- For each active process, a path from its last action to its first.
- A cycle contains all active processes.

Atomicity of exchanges

The language of marked sends sequences equivalent to atomic exchanges is **regular** (EXPSPACE automaton that can be build *on the fly*).