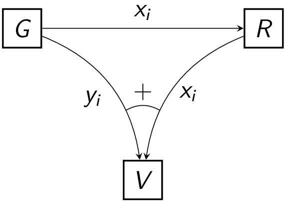
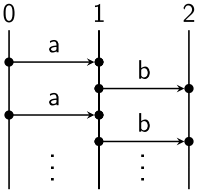
On the Send-synchronizability problem for Mailbox Communication

Romain Delpy, Anca Muscholl, Grégoire Sutre

Univ. of Bordeaux, France

Réunion PaVeDyS, 2025, Paris





1 Introduction

2 Send-synchronizability

3 1-schedulability

4 Fully-matched & Good traces

5 1-sched & Bad traces

6 Conclusion

1 Introduction

2 Send-synchronizability

3 1-schedulability

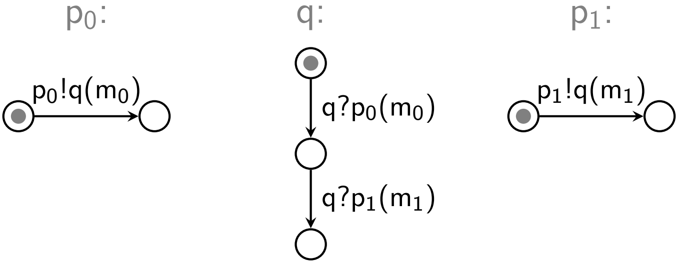
4 Fully-matched & Good traces

5 1-sched & Bad traces

6 Conclusion

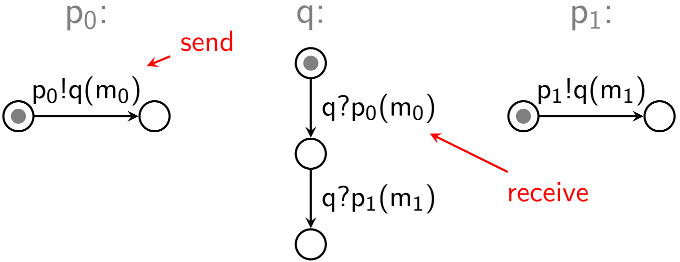
Mailbox semantics

CFM: Communicating Finite-state Machines (set of processes sending/receiving messages).



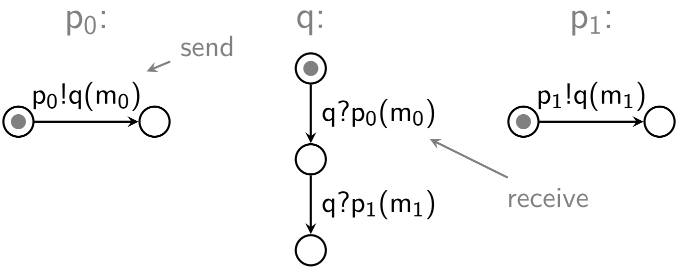
Mailbox semantics

CFM: Communicating Finite-state Machines (set of processes sending/receiving messages).

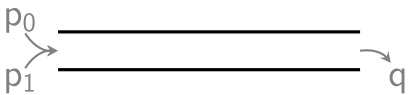


Mailbox semantics

CFM: Communicating Finite-state Machines (set of processes sending/receiving messages).



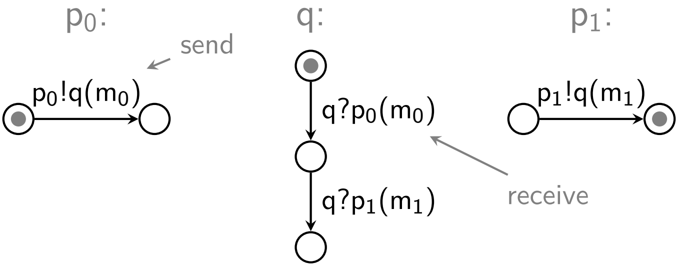
Mailbox:



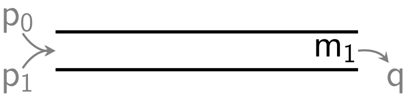
One *FIFO* channel per process.

Mailbox semantics

CFM: Communicating Finite-state Machines (set of processes sending/receiving messages).



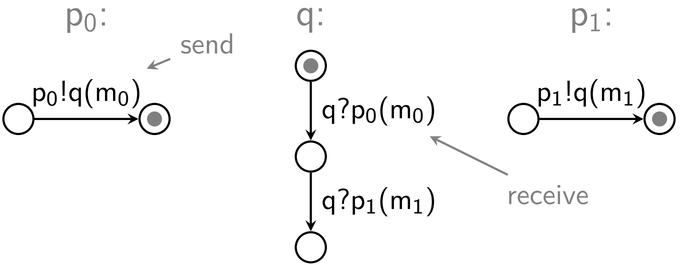
Mailbox:



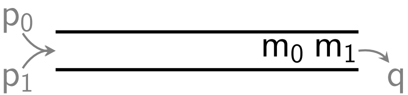
One *FIFO* channel per process.

Mailbox semantics

CFM: Communicating Finite-state Machines (set of processes sending/receiving messages).



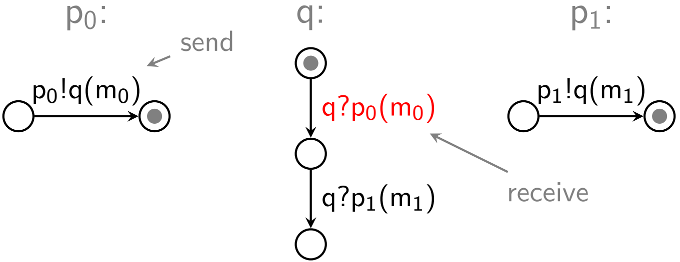
Mailbox:



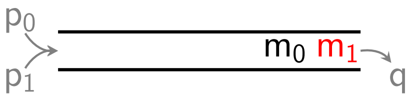
One *FIFO* channel per process.

Mailbox semantics

CFM: Communicating Finite-state Machines (set of processes sending/receiving messages).



Mailbox:



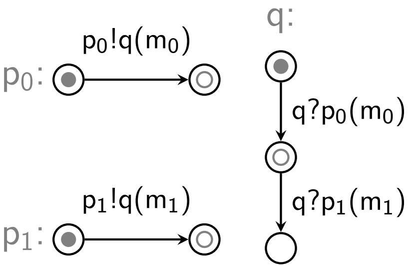
One *FIFO* channel per process.

Mailbox executions

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

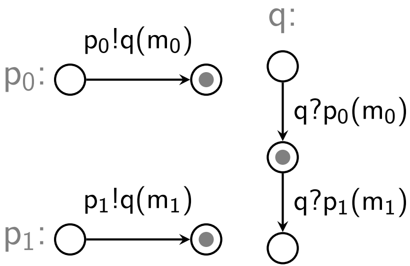
Message Sequence Charts

Multiple executions with same effect on system.



Message Sequence Charts

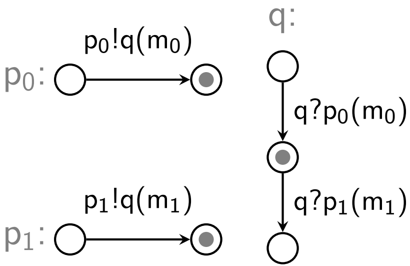
Multiple executions with same effect on system.

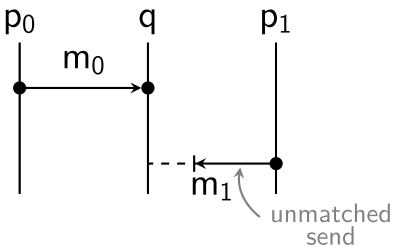


exec 0:	exec 1:
$p_0!q(m_0)$	$p_0!q(m_0)$
$q?p_0(m_0)$	$p_1!q(m_1)$
$p_1!q(m_1)$	$q?p_0(m_0)$

Message Sequence Charts

Multiple executions with same effect on system.





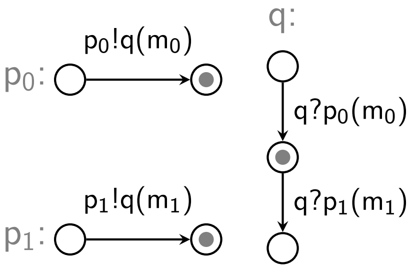
Message Sequence Charts (MSC)

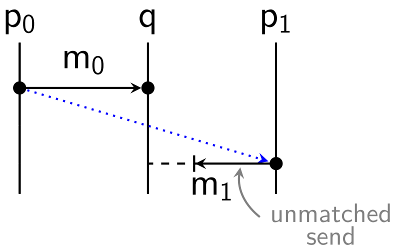
Partial-order representation of behaviors of a CFM (order between events on a same process, between paired send/receive & *mailbox order*).

Two equivalent executions have the same MSC.

Message Sequence Charts

Multiple executions with same effect on system.





Message Sequence Charts (MSC)

Partial-order representation of behaviors of a CFM (order between events on a same process, between paired send/receive & *mailbox order*).

Two equivalent executions have the same MSC.

Mailbox order: two sends to the same process are ordered if the first one is matched.

1 Introduction

2 Send-synchronizability

3 1-schedulability

4 Fully-matched & Good traces

5 1-sched & Bad traces

6 Conclusion

Send-synchronizability

$$w = p_0!q(m_0)\, q?p_0(m_0)\, p_1!q(m_1)$$

projection on sends of w

$$w|_S = p_0!q(m_0) \qquad\qquad p_1!q(m_1)$$

Send-synchronizability

$$w = p_0!q(m_0)\, q?p_0(m_0)\, p_1!q(m_1)$$

projection on sends of w

$$w|_S = p_0!q(m_0) \qquad p_1!q(m_1)$$

Executions

For a CFM $\mathcal{A}$.

- $Tr(\mathcal{A})$ set of all executions of $\mathcal{A}$.
- $Tr_{\mathtt{rdv}}(\mathcal{A})$ set of all executions where sends are directly followed by their matching receive.

Send-synchronizability

$$w = p_0!q(m_0)\; q?p_0(m_0)\; p_1!q(m_1)$$

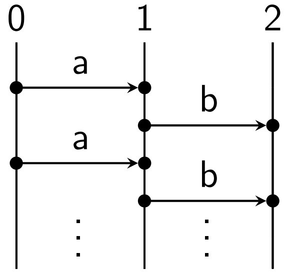
projection on sends of w

$$w|_S = p_0!q(m_0) \qquad\qquad p_1!q(m_1)$$

Executions

For a CFM $\mathcal{A}$.

- $Tr(\mathcal{A})$ set of all executions of $\mathcal{A}$.
- $Tr_{\mathtt{rdv}}(\mathcal{A})$ set of all executions where sends are directly followed by their matching receive.



Send-synchronizability

$$w = p_0!q(m_0)\ q?p_0(m_0)\ p_1!q(m_1)$$

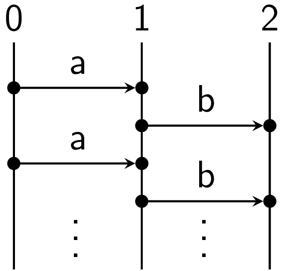
projection on sends of w

$$w|_S = p_0!q(m_0) \qquad\qquad p_1!q(m_1)$$

Executions

For a CFM $\mathcal{A}$.

- $Tr(\mathcal{A})$ set of all executions of $\mathcal{A}$.
- $Tr_{\mathtt{rdv}}(\mathcal{A})$ set of all executions where sends are directly followed by their matching receive.



$$Tr_{\mathtt{rdv}}(\mathcal{A})|_S = (ab)^*(a + \varepsilon)$$

Send-synchronizability

$$w = p_0!q(m_0)\; q?p_0(m_0)\; p_1!q(m_1)$$

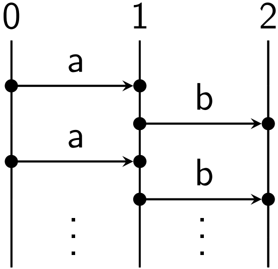
projection on sends of w

$$w|_S = p_0!q(m_0) \qquad\qquad p_1!q(m_1)$$

Executions

For a CFM $\mathcal{A}$.

- $Tr(\mathcal{A})$ set of all executions of $\mathcal{A}$.
- $Tr_{\mathtt{rdv}}(\mathcal{A})$ set of all executions where sends are directly followed by their matching receive.



$Tr_{\mathtt{rdv}}(\mathcal{A})|_S = (ab)^*(a + \varepsilon)$
$Tr(\mathcal{A})|_S \cap a^*b^* = \{a^n b^n \mid n \geq 0\}$, not regular.

Send-synchronizability

$$w = p_0!q(m_0)\; q?p_0(m_0)\; p_1!q(m_1)$$

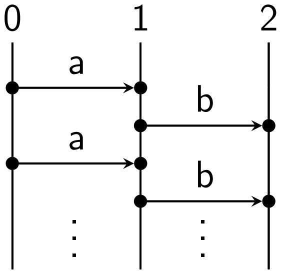
projection on sends of w

$$w|_S = p_0!q(m_0) \qquad p_1!q(m_1)$$

Executions

For a CFM $\mathcal{A}$.

- $Tr(\mathcal{A})$ set of all executions of $\mathcal{A}$.
- $Tr_{\mathtt{rdv}}(\mathcal{A})$ set of all executions where sends are directly followed by their matching receive.



$Tr_{\mathtt{rdv}}(\mathcal{A})|_S = (ab)^*(a + \varepsilon)$
$Tr(\mathcal{A})|_S \cap a^*b^* = \{a^n b^n \mid n \geq 0\}$, not regular.

Send-Synchronizability

A CFM $\mathcal{A}$ is called *sendsend-synchronizable* if
$Tr(\mathcal{A})|_S = Tr_{\mathtt{rdv}}(\mathcal{A})|_S$.

History of Send-synchronizability

- [Basu & Bultan, 2012/2016] Send-synchronizability is decidable for mailbox and peer-to-peer CFMs.

History of Send-synchronizability

- [Basu & Bultan, 2012/2016] Send-synchronizability is decidable for mailbox and peer-to-peer CFMs.
- [Finkel & Lozes, 2017] Showed that the proofs above were flawed. Send-synchronizability is undecidable for peer-to-peer CFMs. Decidable for ring topology.

History of Send-synchronizability

- [Basu & Bultan, 2012/2016] Send-synchronizability is decidable for mailbox and peer-to-peer CFMs.
- [Finkel & Lozes, 2017] Showed that the proofs above were flawed. Send-synchronizability is undecidable for peer-to-peer CFMs. Decidable for ring topology.
- [Di Giusto, Laverza & Peters, 2024] Send-synchronizability is undecidable for CFMs with final states.

History of Send-synchronizability

- [Basu & Bultan, 2012/2016] Send-synchronizability is decidable for mailbox and peer-to-peer CFMs.
- [Finkel & Lozes, 2017] Showed that the proofs above were flawed. Send-synchronizability is undecidable for peer-to-peer CFMs. Decidable for ring topology.
- [Di Giusto, Laverza & Peters, 2024] Send-synchronizability is undecidable for CFMs with final states.

Is Send-synchronizability decidable for mailbox CFMs?

If not, is there a subclass of CFMs such that Send-synchronizability is decidable ?

Undecidability (short)

PCP (variant)

A set of pairs $(x_1, y_1), \ldots, (x_K, y_K)$ of words over an alphabet Σ.
Is there a sequence of indices $i_1, \ldots, i_n \in \{1, \ldots, K\}$ such that

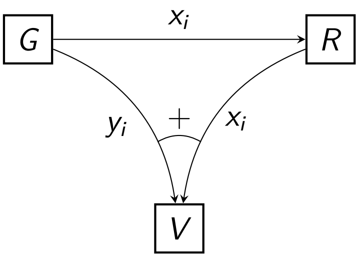
- $i_1 = 1$
- $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$
- $|x_{i_1} \ldots x_{i_n}| \geq |y_{i_1} \ldots y_{i_n}|$

Undecidability (short)

PCP (variant)

A set of pairs $(x_1, y_1), \ldots, (x_K, y_K)$ of words over an alphabet Σ.
Is there a sequence of indices $i_1, \ldots, i_n \in \{1, \ldots, K\}$ such that

- $i_1 = 1$
- $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$
- $|x_{i_1} \ldots x_{i_n}| \geq |y_{i_1} \ldots y_{i_n}|$



- **G**uess guesses the sequence of indices, sends x_i to **R**elay and y_i to **V**erif.
- **R**elay either delays letters of x_i to intertwine them with letters of y_i to **V**erif, or can receive and send anything (dummy).
- **V**erif checks that the two words are identical.

1 Introduction

2 Send-synchronizability

3 1-schedulability

4 Fully-matched & Good traces

5 1-sched & Bad traces

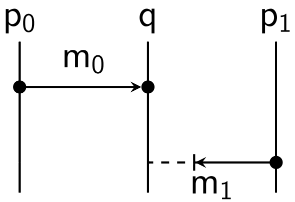
6 Conclusion

Restriction: 1-Schedulable

1-schedulability

A trace is a 1-*scheduling* if every send is either followed by its receive, or is unmatched.

$w = p_0!q(m_0)\, q?p_0(m_0)\, p_1!q(m_1)$ is a 1-scheduling



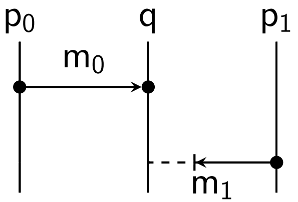
Restriction: 1-Schedulable

1-schedulability

A trace is a 1-*scheduling* if every send is either followed by its receive, or is unmatched.
A trace is 1-*schedulable* if it is equivalent to a 1-scheduling.

$w = p_0!q(m_0)\, q?p_0(m_0)\, p_1!q(m_1)$ is a 1-scheduling

$w' = p_0!q(m_0)\, p_1!q(m_1)\, q?p_0(m_0)$ is 1-schedulable, as $w' \equiv w$



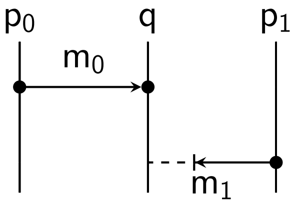
Restriction: 1-Schedulable

1-schedulability

A trace is a 1-*scheduling* if every send is either followed by its receive, or is unmatched.
A trace is 1-*schedulable* if it is equivalent to a 1-scheduling.
A CFM is 1-*schedulable* if all its traces are 1-schedulable.

$w = p_0!q(m_0)\, q?p_0(m_0)\, p_1!q(m_1)$ is a 1-scheduling

$w' = p_0!q(m_0)\, p_1!q(m_1)\, q?p_0(m_0)$ is 1-schedulable, as $w' \equiv w$

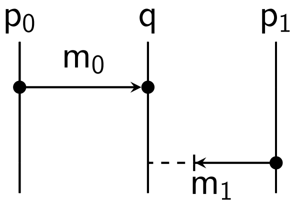


Restriction: 1-Schedulable

1-schedulability

A trace is a 1-*scheduling* if every send is either followed by its receive, or is unmatched.
A trace is 1-*schedulable* if it is equivalent to a 1-scheduling.
A CFM is 1-*schedulable* if all its traces are 1-schedulable.

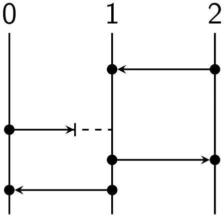
$w = p_0!q(m_0)\, q?p_0(m_0)\, p_1!q(m_1)$ is a 1-scheduling
$w' = p_0!q(m_0)\, p_1!q(m_1)\, q?p_0(m_0)$ is 1-schedulable, as $w' \equiv w$



Fully-matched 1-schedulings of a CFM = rendez-vous traces.

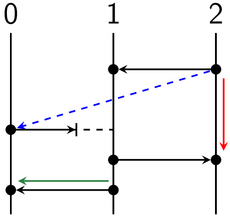
Checking 1-schedulability

- $<_{\mathbb{P}}$ order between event on same process.
- msg order between a send and its receive.
- $<_{\mathtt{mb}}$ mailbox order.



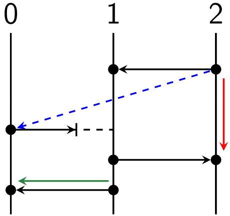
Checking 1-schedulability

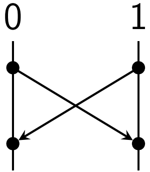
- $<_{\mathbb{P}}$ order between event on same process.
- msg order between a send and its receive.
- $<_{\mathtt{mb}}$ mailbox order.



Checking 1-schedulability

- $<_{\mathbb{P}}$ order between event on same process.
- msg order between a send and its receive.
- $<_{\texttt{mb}}$ mailbox order.

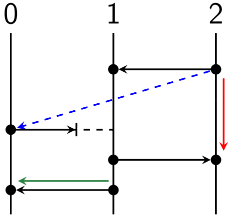


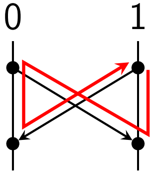


Non 1-schedulable iff
there is a non-trivial $(<_{\mathbb{P}} \cup <_{\texttt{mb}} \cup \mathrm{msg} \cup \mathrm{msg}^{-1})$-cycle.

Checking 1-schedulability

- $<_{\mathbb{P}}$ order between event on same process.
- msg order between a send and its receive.
- $<_{\mathtt{mb}}$ mailbox order.





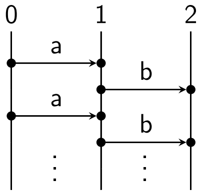
Non 1-schedulable iff
there is a non-trivial $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \mathrm{msg}^{-1})$-cycle.

1-schedulability [Delpy, Muscholl & Sutre 2024]

The question whether a CFM is 1-schedulable is PSPACE-complete.
The language of 1-scheduling of a CFM is regular.

Send-sync & 1-sched

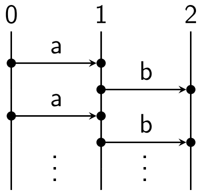
Even for 1-schedulable CFMs, send-synchronizability is still hard:



1-schedulings of $\mathcal{A}$ regular but $Tr(\mathcal{A})|_S$ is not.

Send-sync & 1-sched

Even for 1-schedulable CFMs, send-synchronizability is still hard:

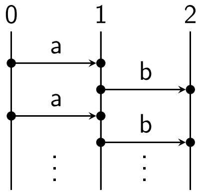


1-schedulings of $\mathcal{A}$ regular but $Tr(\mathcal{A})|_S$ is not.

Need a way to describe $Tr(\mathcal{A})|_S$ from 1-schedulings!

Send-sync & 1-sched

Even for 1-schedulable CFMs, send-synchronizability is still hard:



1-schedulings of $\mathcal{A}$ regular but $Tr(\mathcal{A})|_S$ is not.

Need a way to describe $Tr(\mathcal{A})|_S$ from 1-schedulings!

Two cases:

- Restriction of $Tr(\mathcal{A})$ to fully matched traces.
- Include not fully matched traces.

1 Introduction

2 Send-synchronizability

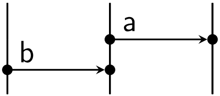
3 1-schedulability

4 Fully-matched & Good traces

5 1-sched & Bad traces

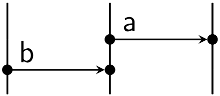
6 Conclusion

Fully-matched traces



w the 1-scheduling of this MSC has $w|_S = a\,b$, but exists $w' \equiv w$, with $w'|_S = b\,a$.

Fully-matched traces

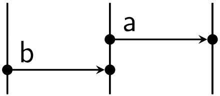


w the 1-scheduling of this MSC has $w|_S = a\,b$, but exists $w' \equiv w$, with $w'|_S = b\,a$.

Commutation: $SI \subseteq S \times S$. Let $a = p!q(m)$ and $b = p'!q'(m')$, $(a, b) \in SI$ if $p \neq p'$, $q \neq q'$ and $q \neq p'$

If $(a, b) \in SI$, $u, v \in S^*$: $uabv \Rightarrow_{SI} ubav$.

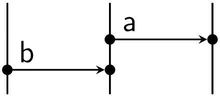
Fully-matched traces



Commutation: $SI \subseteq S \times S$. Let $a = p!q(m)$ and $b = p'!q'(m')$, $(a, b) \in SI$ if $p \neq p'$, $q \neq q'$ and $q \neq p'$

If $(a, b) \in SI$, $u, v \in S^*$: $uabv \Rightarrow_{SI} ubav$.

Fully-matched traces

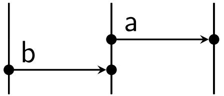


Commutation: $SI \subseteq S \times S$. Let $a = p!q(m)$ and $b = p'!q'(m')$, $(a, b) \in SI$ if $p \neq p'$, $q \neq q'$ and $q \neq p'$

If $(a, b) \in SI$, $u, v \in S^*$: $uabv \Rightarrow_{SI} ubav$.

$Cl_{SI}(v) = \{v' \in S^* \mid v \overset{*}{\Rightarrow}_{SI} v'\}$

Fully-matched traces



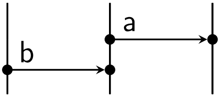
Commutation: $SI \subseteq S \times S$. Let $a = p!q(m)$ and $b = p'!q'(m')$, $(a, b) \in SI$ if $p \neq p'$, $q \neq q'$ and $q \neq p'$

If $(a, b) \in SI$, $u, v \in S^*$: $uabv \Rightarrow_{SI} ubav$.

$Cl_{SI}(v) = \{v' \in S^* \mid v \overset{*}{\Rightarrow}_{SI} v'\}$

If $\mathcal{A}$ 1-schedulable: $Tr_{\text{full-match}}(\mathcal{A})|_S = Cl_{SI}(Tr_{\texttt{rdv}}(\mathcal{A})|_S)$.

Fully-matched traces



Commutation: $SI \subseteq S \times S$. Let $a = p!q(m)$ and $b = p'!q'(m')$, $(a, b) \in SI$ if $p \neq p'$, $q \neq q'$ and $q \neq p'$

If $(a, b) \in SI$, $u, v \in S^*$: $uabv \Rightarrow_{SI} ubav$.

$Cl_{SI}(v) = \{v' \in S^* \mid v \overset{*}{\Rightarrow}_{SI} v'\}$

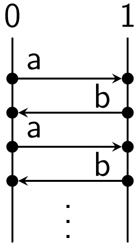
If $\mathcal{A}$ 1-schedulable: $Tr_{\text{full-match}}(\mathcal{A})|_S = Cl_{SI}(Tr_{\mathtt{rdv}}(\mathcal{A})|_S)$.

One can check if $\mathcal{L} \subseteq S^*$ regular is closed under SI.

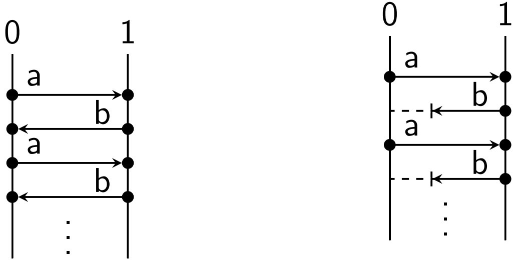
Send-sync for fully-matched 1-schedulable

The question whether a 1-schedulable CFM is send-synchronizable over fully-matched traces is decidable.

Unmatched sends makes it harder

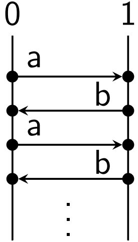


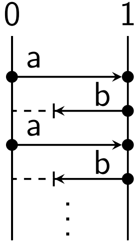
Unmatched sends makes it harder



$(a, b) \notin SI$, but commutations are oblivious to matching of sends.

Unmatched sends makes it harder





$(a, b) \notin SI$, but commutations are oblivious to matching of sends.

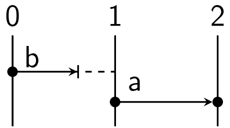
How to account for unmatched sends?

Extended order

Order caused by "possibility of receive":

$a \dashrightarrow b$

- b not matched
- $a \parallel b$ (not ordered)

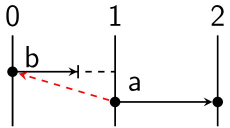


Extended order

Order caused by "possibility of receive":

$a \dashrightarrow b$

- b not matched
- $a \parallel b$ (not ordered)



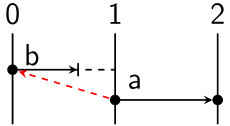
Does not depend on the order of a and b in w.

Extended order

Order caused by "possibility of receive":

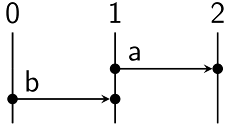
$a \dashrightarrow b$

- b not matched
- $a \parallel b$ (not ordered)



Does not depend on the order of a and b in w.

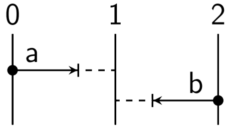
This order is convenient with SI!



Good traces

We say that $w \equiv_{\text{us}} w'$ if

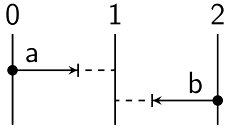
- $w \equiv w'$
- The order of unmatched sends to each p is the same in w and w'



Good traces

We say that $w \equiv_{\mathrm{us}} w'$ if

- $w \equiv w'$
- The order of unmatched sends to each p is the same in w and w'



$w = a\, b$

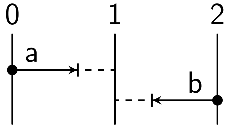
$w' = b\, a$

Here $w \equiv w'$ but $w \not\equiv_{\mathrm{us}} w'$!

Good traces

We say that $w \equiv_{\text{us}} w'$ if

- $w \equiv w'$
- The order of unmatched sends to each p is the same in w and w'



$w = a\, b$

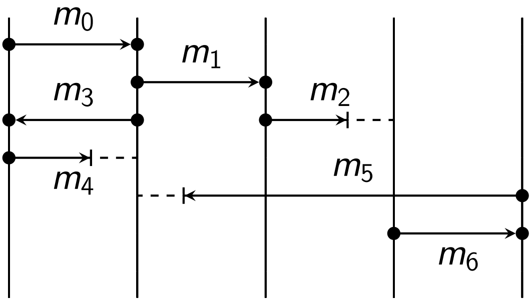
$w' = b\, a$

Here $w \equiv w'$ but $w \not\equiv_{\text{us}} w'$!

A trace w is **good** if there exists 1-scheduling $w' \equiv_{\text{us}} w$ with no $\dashrightarrow$-backward arcs.

Good trace: example

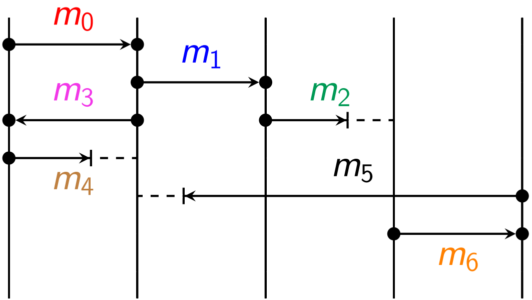
A trace w is **good** if there exists 1-scheduling $w' \equiv_{\mathrm{us}} w$ with no $\dashrightarrow$-backward arcs:



$w \; = s_0 \, r_0 \, s_1 \, s_3 \, r_1 \, s_2 \, r_3 \, s_4 \, s_5 \, s_6 \, r_6$

Good trace: example

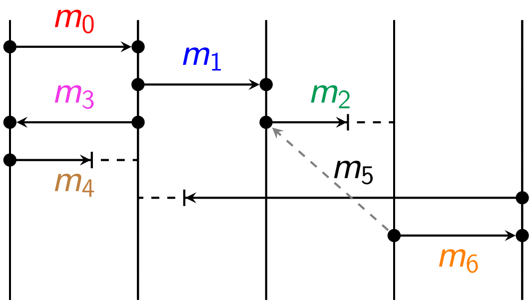
A trace w is **good** if there exists 1-scheduling $w' \equiv_{\mathrm{us}} w$ with no $\dashrightarrow$-backward arcs:



$w \;=\; s_0\; r_0\; s_1\; s_3\; r_1\; s_2\; r_3\; s_4\; s_5\; s_6\; r_6$

Good trace: example

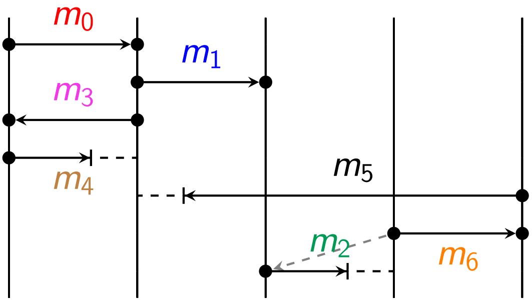
A trace w is **good** if there exists 1-scheduling $w' \equiv_{\mathrm{us}} w$ with no $\dashrightarrow$-backward arcs:



$w = s_0\ r_0\ s_1\ s_3\ r_1\ s_2\ r_3\ s_4\ s_5\ s_6\ r_6$

Good trace: example

A trace w is **good** if there exists 1-scheduling $w' \equiv_{\mathrm{us}} w$ with no $\dashrightarrow$-backward arcs:



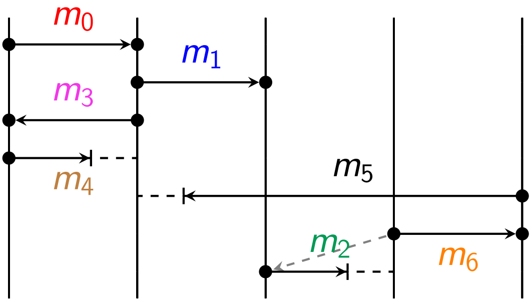
$w \;= s_0\, r_0\, s_1\, s_3\, r_1\, s_2\, r_3\, s_4\, s_5\, s_6\, r_6$

$w' = s_0\, r_0\, s_1\, r_1\, s_3\, r_3\, s_4\, s_5\, s_6\, r_6\, s_2$

$w' \equiv_{\mathrm{us}} w.$

Good trace: example

A trace w is **good** if there exists 1-scheduling $w' \equiv_{\text{us}} w$ with no $\dashrightarrow$-backward arcs:



$w \;= s_0\, r_0\, s_1\, s_3\, r_1\, s_2\, r_3\, s_4\, s_5\, s_6\, r_6$

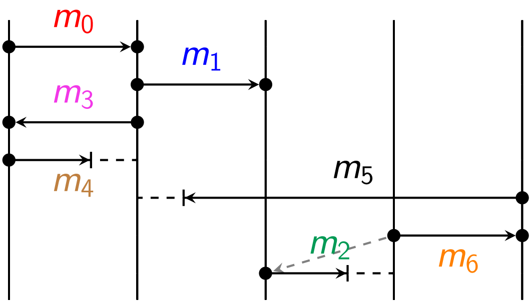
$w' = s_0\, r_0\, s_1\, r_1\, s_3\, r_3\, s_4\, s_5\, s_6\, r_6\, s_2$

$w' \equiv_{\text{us}} w.$

$w|_S \in Cl_{SI}(w'|_S)$

Good trace: example

A trace w is **good** if there exists 1-scheduling $w' \equiv_{\mathrm{us}} w$ with no $\dashrightarrow$-backward arcs:



$w \;= s_0\ r_0\ s_1\ s_3\ r_1\ s_2\ r_3\ s_4\ s_5\ s_6\ r_6$

$w' = s_0\ r_0\ s_1\ r_1\ s_3\ r_3\ s_4\ s_5\ s_6\ r_6\ s_2$

$w' \equiv_{\mathrm{us}} w$.

$w|_S \in Cl_{SI}(w'|_S)$

Good traces & Send-synchronizability

If all traces of a CFM $\mathcal{A}$ are good:
$Tr(\mathcal{A})|_S = Cl_{SI}(\{w \mid w \text{ is a 1-scheduling of } \mathcal{A}\}|_S)$.

1 Introduction

2 Send-synchronizability

3 1-schedulability

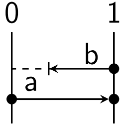
4 Fully-matched & Good traces

5 1-sched & Bad traces

6 Conclusion

Bad traces

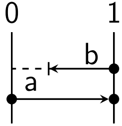
Bad traces prevent send-synchronizability of 1-schedulable CFMs:



$b\,a \in Tr(\mathcal{A})|_S$.
But b and a not ordered, so $a\,b \in Tr(\mathcal{A})|_S$.

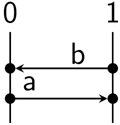
Bad traces

Bad traces prevent send-synchronizability of 1-schedulable CFMs:

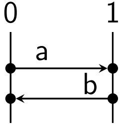


$b\,a \in Tr(\mathcal{A})|_S$.
But b and a not ordered, so $a\,b \in Tr(\mathcal{A})|_S$.

If $\mathcal{A}$ is send-synchronizable, then $Tr_{\mathtt{rdv}}(\mathcal{A})$ contains

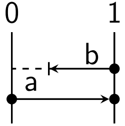


and



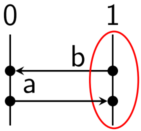
Bad traces

Bad traces prevent send-synchronizability of 1-schedulable CFMs:

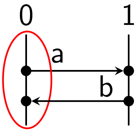


$b\,a \in Tr(\mathcal{A})|_S$.
But b and a not ordered, so $a\,b \in Tr(\mathcal{A})|_S$.

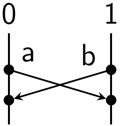
If $\mathcal{A}$ is send-synchronizable, then $Tr_{\mathtt{rdv}}(\mathcal{A})$ contains



and

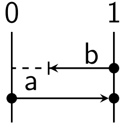


$\mathcal{A}$ will also have the following trace:



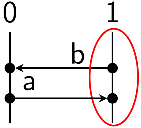
Bad traces

Bad traces prevent send-synchronizability of 1-schedulable CFMs:

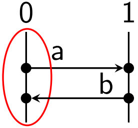


$b\,a \in Tr(\mathcal{A})|_S$.
But b and a not ordered, so $a\,b \in Tr(\mathcal{A})|_S$.

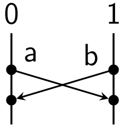
If $\mathcal{A}$ is send-synchronizable, then $Tr_{\texttt{rdv}}(\mathcal{A})$ contains



and



$\mathcal{A}$ will also have the following trace:
So $\mathcal{A}$ is not 1-schedulable!

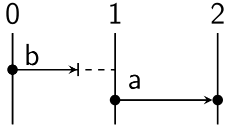


Extended order: double unmatched

Recall:

$a \dashrightarrow b$

- b unmatched
- $a \parallel b$ (not ordered)

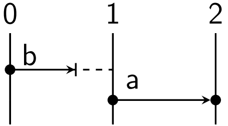


Extended order: double unmatched

Recall:

$a \dashrightarrow b$

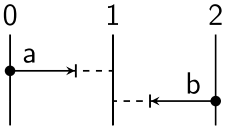
- b unmatched
- $a \parallel b$ (not ordered)



New order:

$a \ll^{w}_{\mathrm{us}} b$

- a, b unmatched
- $a \parallel b$
- a is before b in w



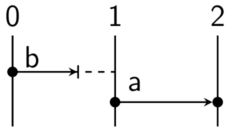
$w = a\,b$

Extended order: double unmatched

Recall:

$a \dashrightarrow b$

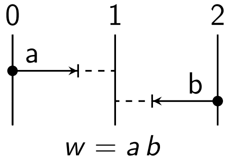
- b unmatched
- $a \parallel b$ (not ordered)



New order:

$a \ll^{w}_{\mathrm{us}} b$

- a, b unmatched
- $a \parallel b$
- a is before b in w



Rem: $u \equiv_{\mathrm{us}} v$ iff $\ll^{u}_{\mathrm{us}} = \ll^{v}_{\mathrm{us}}$.

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

Detecting bad traces

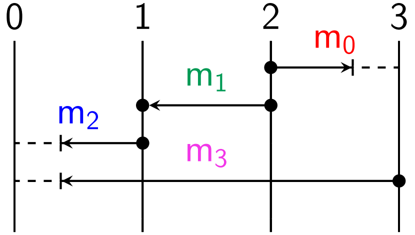
A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle

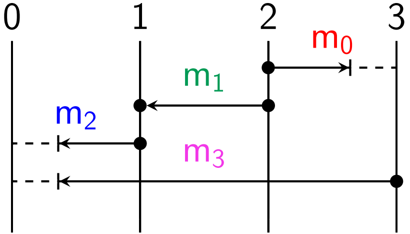


$w = s_0\, s_1\, r_1\, s_2\, s_3$

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle



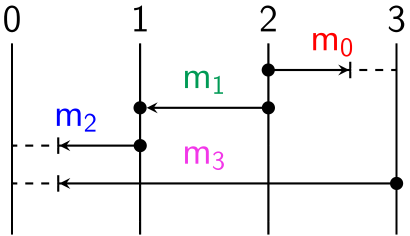
$w = s_0\ s_1\ r_1\ s_2\ s_3$

$s_0 <_{\mathbb{P}} s_1 \ \mathrm{msg}\ r_1 <_{\mathbb{P}} s_2 \ll^{w}_{\mathrm{us}} s_3 \dashrightarrow s_0$

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle



$w = s_0\, s_1\, r_1\, s_2\, s_3$

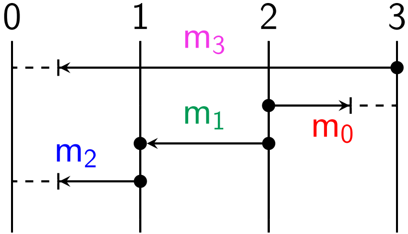
$s_0 <_{\mathbb{P}} s_1 \;\mathrm{msg}\; r_1 <_{\mathbb{P}} s_2 \ll^{w}_{\mathrm{us}} s_3 \dashrightarrow s_0$

$w' = s_3\, s_0\, s_1\, r_1\, s_2$.

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle



$w = s_0\ s_1\ r_1\ s_2\ s_3$

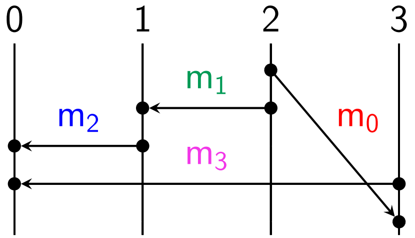
$s_0 <_{\mathbb{P}} s_1 \ \mathrm{msg}\ r_1 <_{\mathbb{P}} s_2 \ll^{w}_{\mathrm{us}} s_3 \dashrightarrow s_0$

$w' = s_3\ s_0\ s_1\ r_1\ s_2$.

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle



$w = s_0\ s_1\ r_1\ s_2\ s_3$

$s_0 <_{\mathbb{P}} s_1 \ \mathrm{msg}\ r_1 <_{\mathbb{P}} s_2 \ll^{w}_{\mathrm{us}} s_3 \dashrightarrow s_0$

$w' = s_3\ s_0\ s_1\ r_1\ s_2$.

Detecting bad traces

A trace w is good if some 1-scheduling $w \equiv_{\mathrm{us}} w'$ exists with no $\dashrightarrow$-backward arcs

A trace w is **bad** iff it has a $(<_{\mathbb{P}} \cup <_{\mathtt{mb}} \cup \mathrm{msg} \cup \dashrightarrow \cup \ll^{w}_{\mathrm{us}})$-cycle

Bad traces and send-synchronizability

If a 1-schedulable CFM has some bad trace, then it is not send-synchronizable.

Checking if a 1-schedulable CFM has some bad trace is PSPACE-complete.

$s_0 <_{\mathbb{P}} s_1 \;\mathrm{msg}\; r_1 <_{\mathbb{P}} s_2 \ll^{w}_{\mathrm{us}} s_3 \dashrightarrow s_0$

$w' = s_3\, s_0\, s_1\, r_1\, s_2$.

Conclusion

- Send-synchronizability is undecidable for mailbox CFMs.

Conclusion

- Send-synchronizability is undecidable for mailbox CFMs.
- Checking send-synchronizability is PSPACE for the subclass of 1-schedulable CFMs (property that can be checked in PSPACE).

Conclusion

- Send-synchronizability is undecidable for mailbox CFMs.
- Checking send-synchronizability is PSPACE for the subclass of 1-schedulable CFMs (property that can be checked in PSPACE).

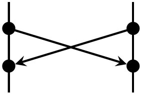
Technique used for the proof could be used for other problems (realizability?).

Conclusion

- Send-synchronizability is undecidable for mailbox CFMs.
- Checking send-synchronizability is PSPACE for the subclass of 1-schedulable CFMs (property that can be checked in PSPACE).

Technique used for the proof could be used for other problems (realizability?).

1-schedulability is very restrictive :
Can we extend to k-schedulability ?



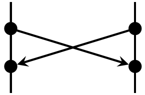
A 2-exchange.

Conclusion

- Send-synchronizability is undecidable for mailbox CFMs.
- Checking send-synchronizability is PSPACE for the subclass of 1-schedulable CFMs (property that can be checked in PSPACE).

Technique used for the proof could be used for other problems (realizability?).

1-schedulability is very restrictive :
Can we extend to k-schedulability ?



A 2-exchange.

THANK YOU

CFM for Pre-MPCP reduction