Local Testing of Message Sequence Charts is Difficult

K Narayan Kumar

Chennai Mathematical Institute
http://www.cmi.ac.in/~kumar

(Joint work with P. Bhateja, P. Gastin and M. Mukund)

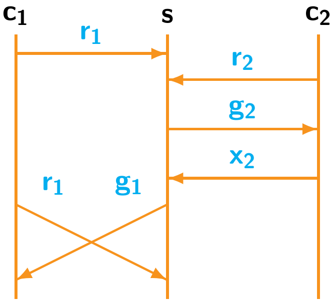
ENS de Cachan
29 May 2007

Message sequence charts (MSC)

- Telecommunications
- Describes a pattern of interaction (a Scenario)
- Attractive visual formalism
- Messages sent between communicating agents
- UML
  - Sequence diagrams
  - Interaction between objects
    e.g., method invocations etc

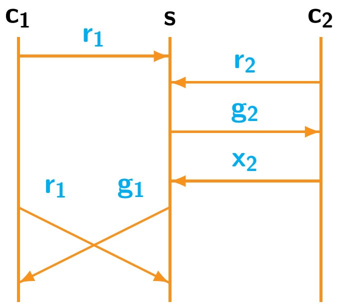
Message sequence charts: Partial Orders

Two clients and a server

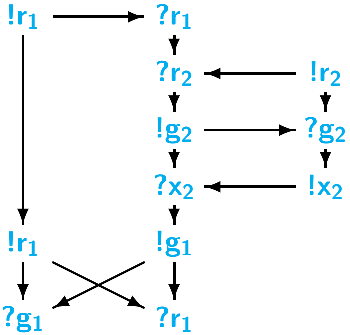


Message sequence charts: Partial Orders

Two clients and a server,

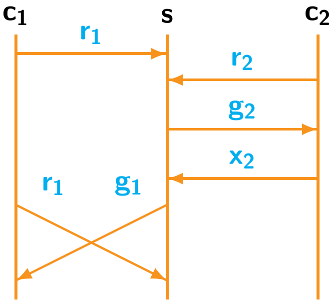


and a partial order representation

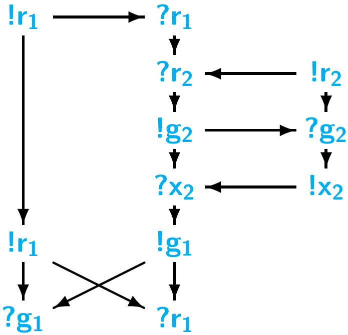


Message sequence charts: Partial Orders

Two clients and a server,



and a partial order representation



All channels are assumed to be FIFO.

MSC can be regenerated from any one sequentialization.

Collections of MSCs

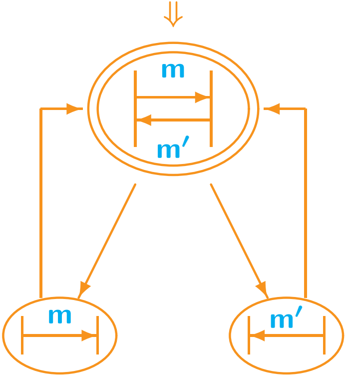
- Often need to specify a collection of scenarios
- Finite collection can be exhaustively enumerated
- Infinite collection needs a generating mechanism

High level MSCs (HMSCs)

- A finite state automaton
- Each state is labelled by a MSC
- Each (legal) path in the automaton generates a MSC

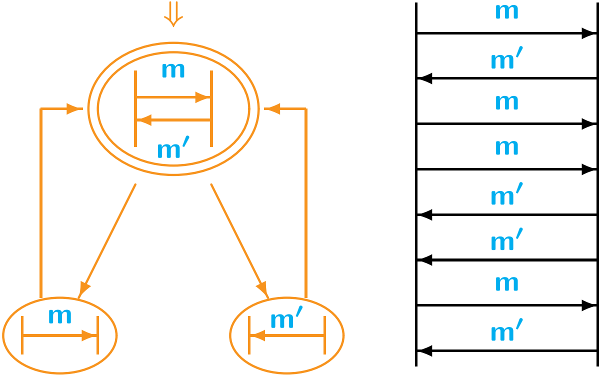
High level MSCs (HMSCs)

- A finite state automaton
- Each state is labelled by a MSC
- Each (legal) path in the automaton generates a MSC



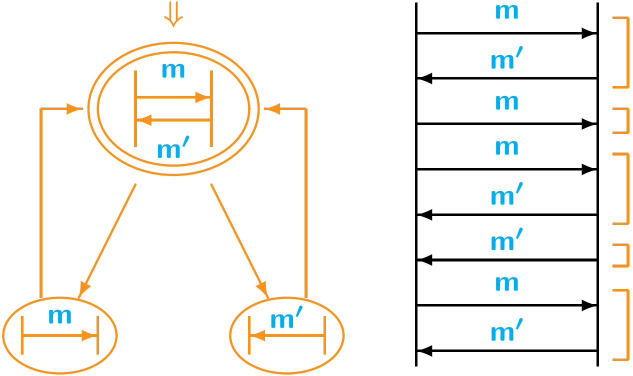
High level MSCs (HMSCs)

- A finite state automaton
- Each state is labelled by a MSC
- Each (legal) path in the automaton generates a MSC



High level MSCs (HMSCs)

- A finite state automaton
- Each state is labelled by a MSC
- Each (legal) path in the automaton generates a MSC



Concatenation of MSCs

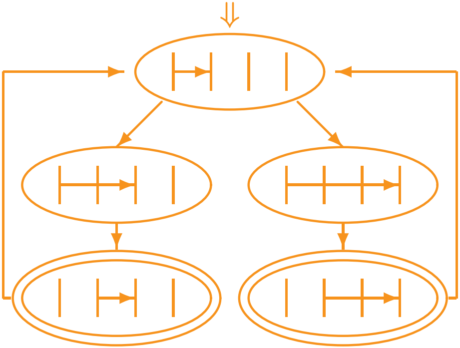
- First MSC finishes before second starts : *synchronous*

Concatenation of MSCs

- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

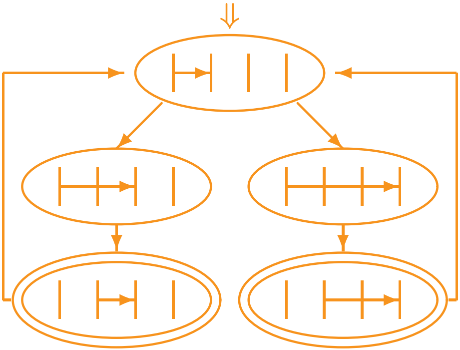
Concatenation of MSCs

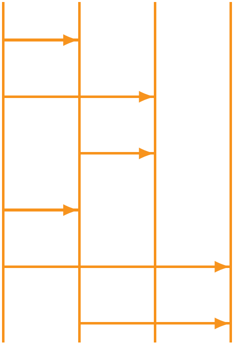
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC



Concatenation of MSCs

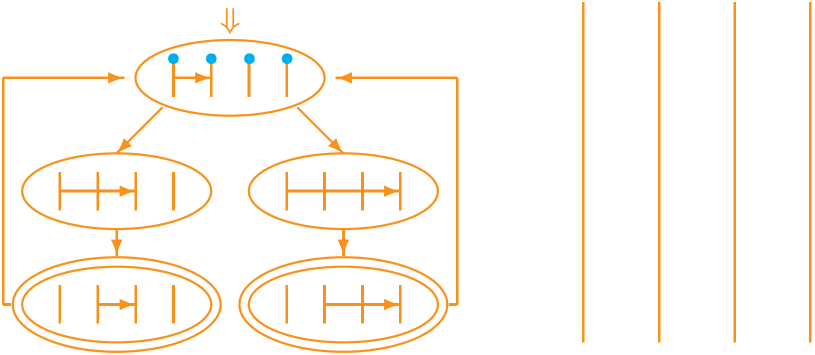
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC





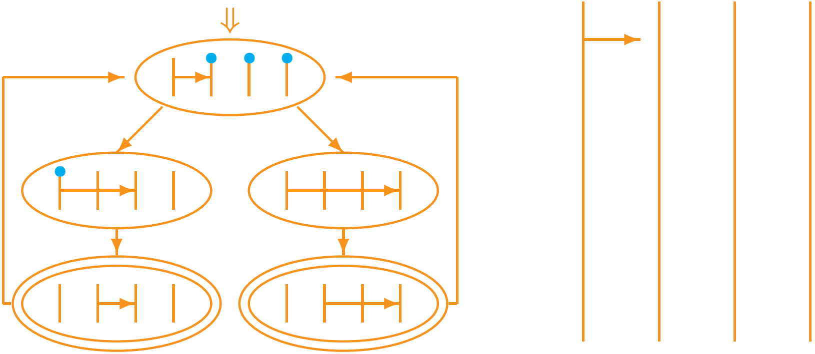
Concatenation of MSCs

- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC



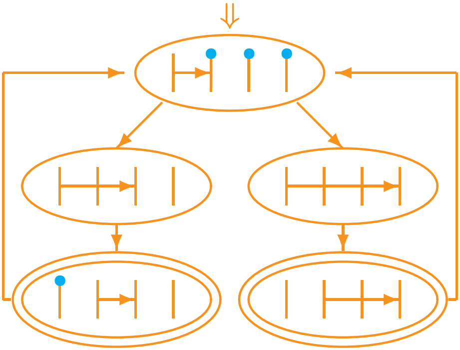
Concatenation of MSCs

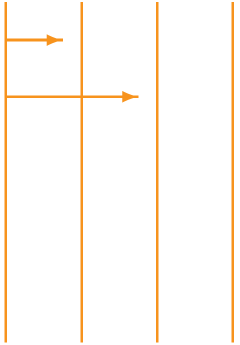
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
    - Some processes may proceed to second MSC before others complete actions of first MSC



Concatenation of MSCs

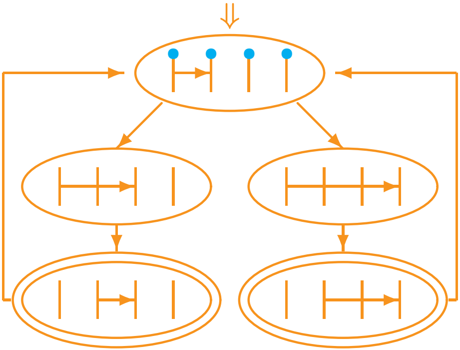
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

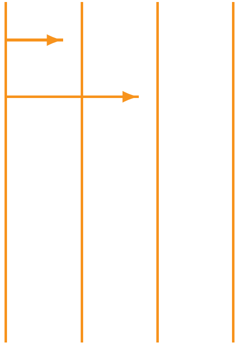




Concatenation of MSCs

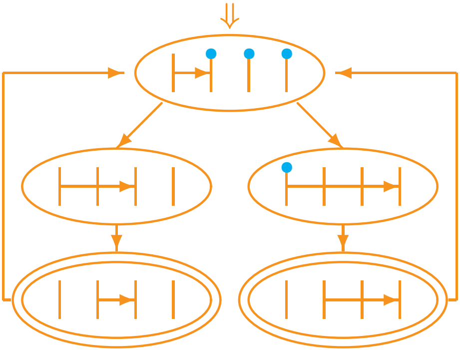
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
    - Some processes may proceed to second MSC before others complete actions of first MSC

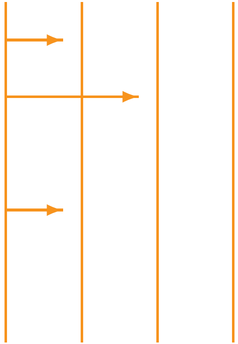




Concatenation of MSCs

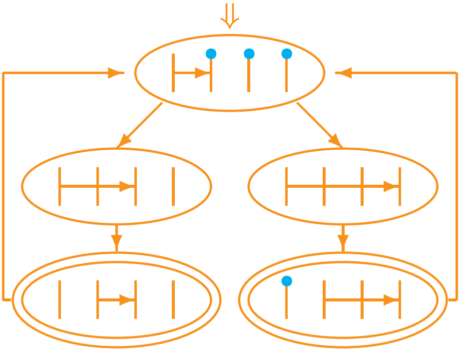
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
    - Some processes may proceed to second MSC before others complete actions of first MSC

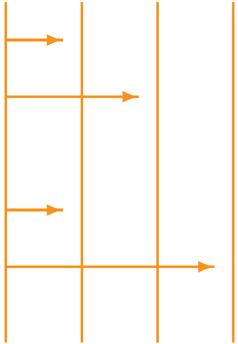




Concatenation of MSCs

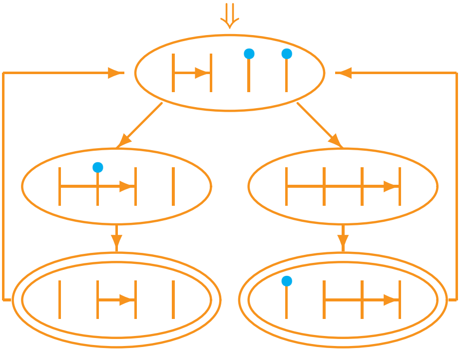
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

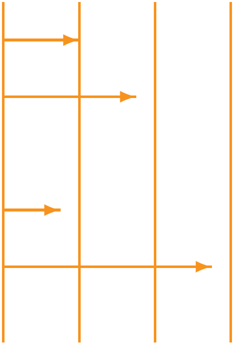




Concatenation of MSCs

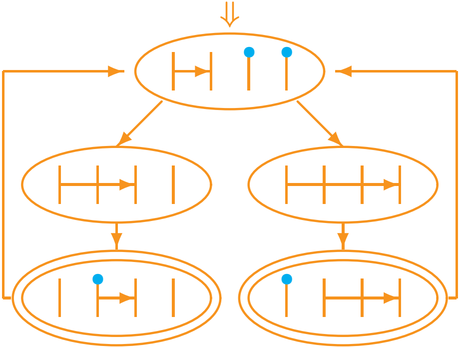
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

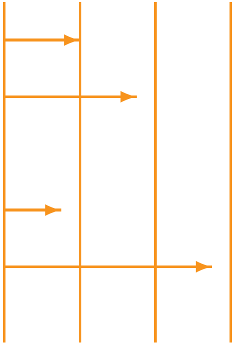




Concatenation of MSCs

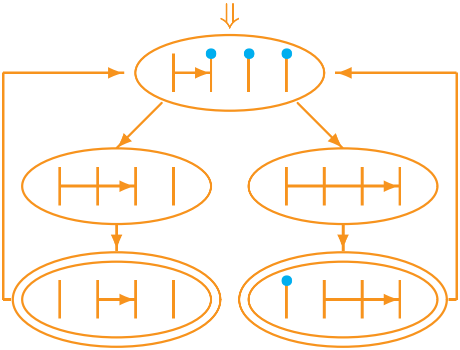
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

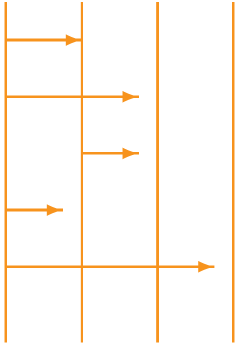




Concatenation of MSCs

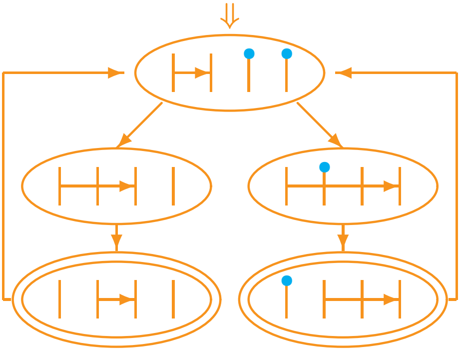
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
    - Some processes may proceed to second MSC before others complete actions of first MSC

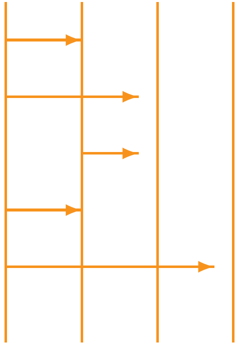




Concatenation of MSCs

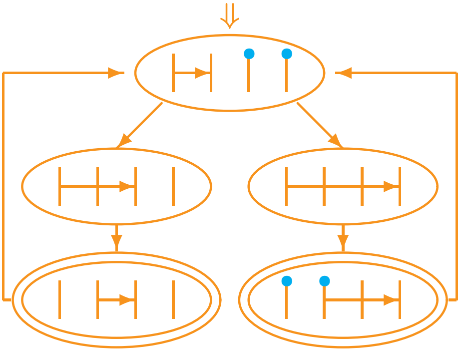
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

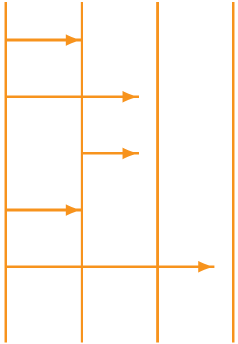




Concatenation of MSCs

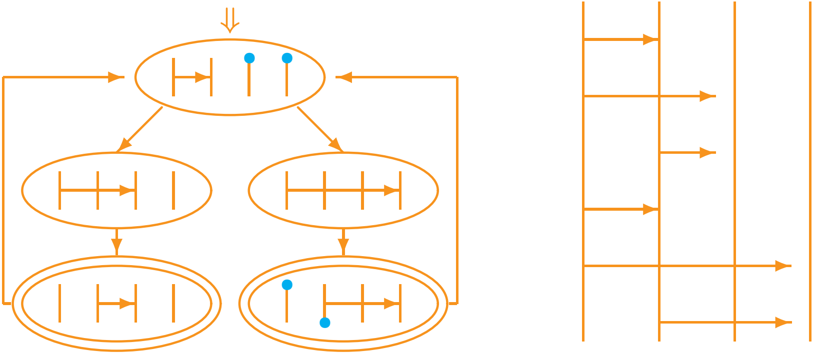
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC





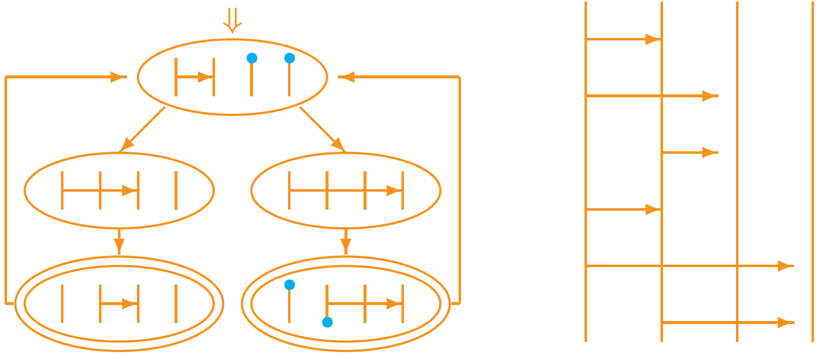
Concatenation of MSCs

- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC



Concatenation of MSCs

- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
- "Executing" HMSC may require unbounded history



Regular MSC languages

- An MSC is (uniquely) determined by its linearizations

Regular MSC languages

- An MSC is (uniquely) determined by its linearizations
  - Set of strings over send actions **p!q(m)** and receive actions **p?q(m)**

Regular MSC languages

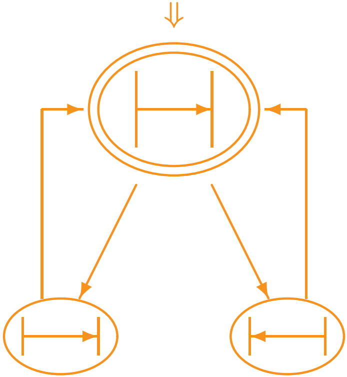
- An MSC is (uniquely) determined by its linearizations
  - Set of strings over send actions **p!q(m)** and receive actions **p?q(m)**
- Regular collection of MSCs $\triangleq$
  linearizations form a regular language

HMSCs and regularity

- HMSC specifications may not be regular

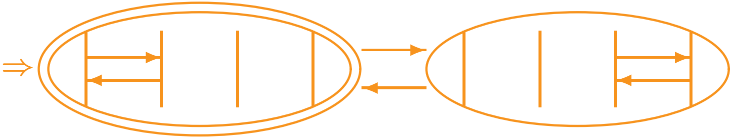
HMSCs and regularity

- HMSC specifications may not be regular
- Problem 1 Unbounded buffers



HMSCs and regularity

- HMSC specifications may not be regular
- Problem 1 Unbounded buffers
- Problem 2 Global synchronization yields context-free behaviours



- Sufficient structural conditions on HMSCs to guarantee regularity . . . [AY99,MP99]

  Locally Synchronized HMSCs

Locally synchronized HMSCs

- Construct communication graph for a MSC
  $\mathbf{p} \longrightarrow \mathbf{q}$ iff $\mathbf{p}$ sends a message to $\mathbf{q}$

Locally synchronized HMSCs

- Construct communication graph for a MSC
  $\mathbf{p} \longrightarrow \mathbf{q}$ iff $\mathbf{p}$ sends a message to $\mathbf{q}$
- For each loop, communication graph is one strongly connected component plus isolated vertices

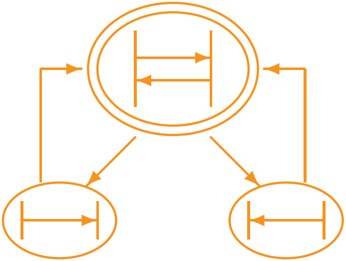
Locally synchronized HMSCs

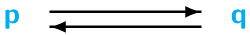
- Construct communication graph for a MSC
  $\mathbf{p} \longrightarrow \mathbf{q}$ iff $\mathbf{p}$ sends a message to $\mathbf{q}$
- For each loop, communication graph is one strongly connected component plus isolated vertices
- In each loop, every message is "acknowledged"

Locally synchronized HMSCs

- Construct communication graph for a MSC
  $\mathbf{p} \longrightarrow \mathbf{q}$ iff $\mathbf{p}$ sends a message to $\mathbf{q}$
- For each loop, communication graph is one strongly connected component plus isolated vertices
- In each loop, every message is "acknowledged"

$\Downarrow$

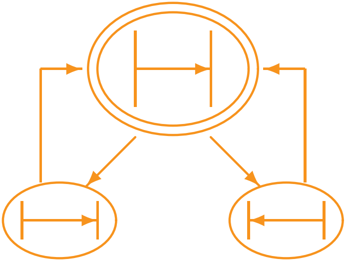




Locally synchronized HMSCs

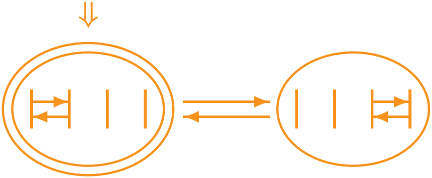
- Construct communication graph for a MSC
  $\mathbf{p} \longrightarrow \mathbf{q}$ iff $\mathbf{p}$ sends a message to $\mathbf{q}$
- For each loop, communication graph is one strongly connected component plus isolated vertices
- In each loop, every message is "acknowledged"

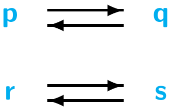
$\Downarrow$



Locally synchronized HMSCs

- Construct communication graph for a MSC
  $\mathbf{p} \longrightarrow \mathbf{q}$ iff $\mathbf{p}$ sends a message to $\mathbf{q}$
- For each loop, communication graph is one strongly connected component plus isolated vertices
- In each loop, every message is "acknowledged"





Local Testing

Given a HMSC specification and a implementation can we check whether the implementation is correct using local observations?

Local Testing

Given a HMSC specification and a implementation can we check whether the implementation is correct using local observations?

- For each process there is an observer who records the sequences of events.

Local Testing

Given a HMSC specification and a implementation can we check whether the implementation is correct using local observations?

- For each process there is an observer who records the sequences of events.
- If each observer records a possible scenario in the language of the HMSC then the implementation is deemed to be correct.

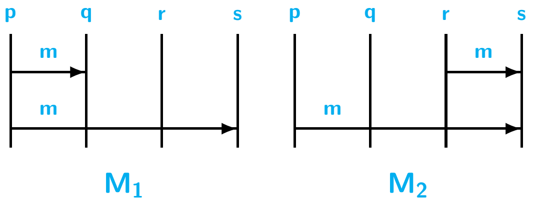
Local Testing

Given a HMSC specification and a implementation can we check whether the implementation is correct using local observations?

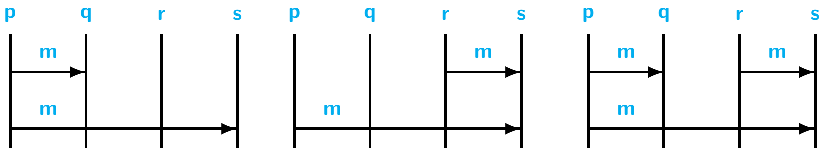
- For each process there is an observer who records the sequences of events.
- If each observer records a possible scenario in the language of the HMSC then the implementation is deemed to be correct.

Will local testing suffice to check (regular) HMSC languages?

Implied scenarios [AEY, ICSE '00]



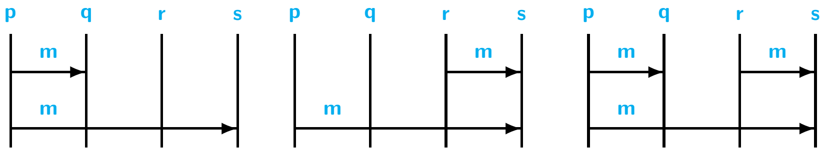
Implied scenarios [AEY, ICSE '00]



$\mathbf{M_1}$ $\mathbf{M_2}$ $\mathbf{M}$

- **p** and **q** believe $\mathbf{M}$ is $\mathbf{M_1}$
- **r** and **s** believe $\mathbf{M}$ is $\mathbf{M_2}$

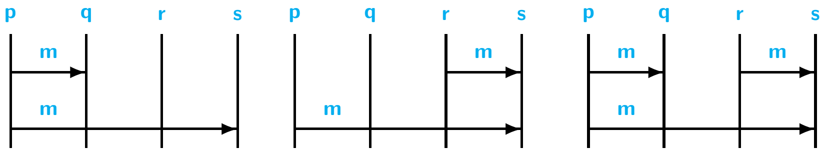
Implied scenarios [AEY, ICSE '00]



M_1 M_2 M

- **p** and **q** believe **M** is M_1
- **r** and **s** believe **M** is M_2
- MSC **M** is implied by **L** if for each process **p**, the **p**-projection of **M** matches the **p**-projection of some MSC in **L**

Implied scenarios [AEY, ICSE '00]



$\mathbf{M_1}$ $\mathbf{M_2}$ $\mathbf{M}$

- **p** and **q** believe **M** is $\mathbf{M_1}$
- **r** and **s** believe **M** is $\mathbf{M_2}$
- MSC **M** is implied by **L** if for each process **p**, the **p**-projection of **M** matches the **p**-projection of some MSC in **L**
- An MSC language is weakly realizable if it is closed with respect to implied MSCs

Implied scenarios . . .

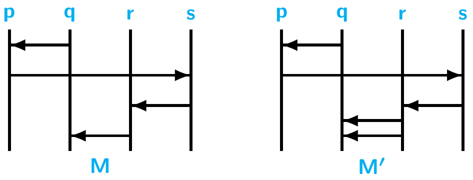
- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]

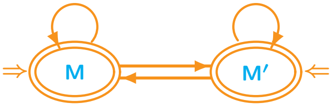
Implied scenarios . . .

- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]
- Even if the original language has bounded channels, its weak closure may not

Implied scenarios . . .

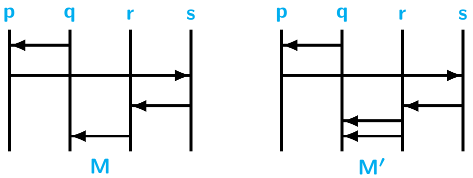
- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]
- Even if the original language has bounded channels, its weak closure may not

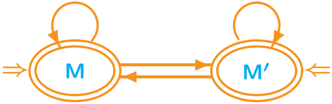


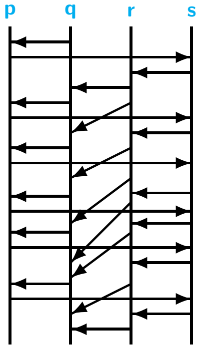


Implied scenarios . . .

- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]
- Even if the original language has bounded channels, its weak closure may not

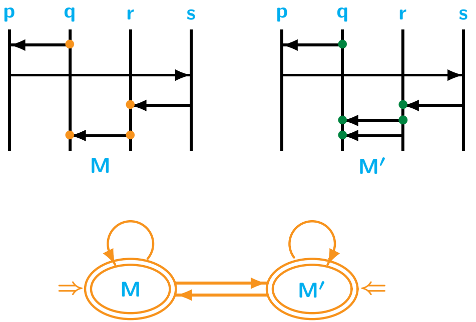


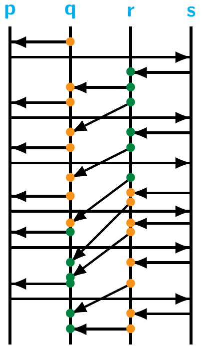




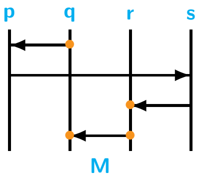
Implied scenarios . . .

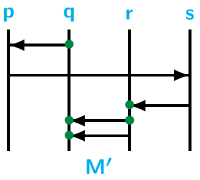
- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]
- Even if the original language has bounded channels, its weak closure may not

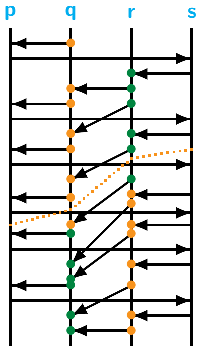




- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]
- Even if the original language has bounded channels, its weak closure may not







Confusing $M^{2k}M'^{k}$ and $M'^{k}M^{2k}$ generates upto k messages in $p \longrightarrow s$ channel

Local Testing

- Local observers record the behaviour of each process.

Local Testing

- Local observers record the behaviour of each process.
- Each observer verifies that the local behaviour is consistent with some permitted behaviour of the system.

Local Testing

- Local observers record the behaviour of each process.
- Each observer verifies that the local behaviour is consistent with some permitted behaviour of the system.
- Will this guarantee that the global behaviour is permitted?

Local Testing

- Local observers record the behaviour of each process.
- Each observer verifies that the local behaviour is consistent with some permitted behaviour of the system.
- Will this guarantee that the global behaviour is permitted?
    - Not Always. If and only if the system has no implied scenarios.

Local Testing

- Local observers record the behaviour of each process.
- Each observer verifies that the local behaviour is consistent with some permitted behaviour of the system.
- Will this guarantee that the global behaviour is permitted?
    - Not Always. If and only if the system has no implied scenarios.
- Local testability is equivalent to absence of implied scenarios.

Local Testing

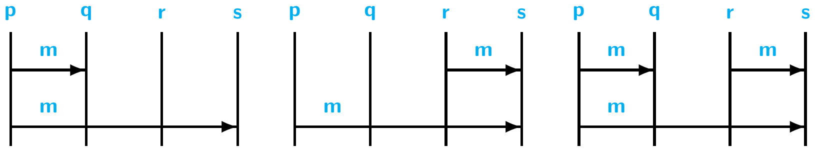
- Local observers record the behaviour of each process.
- Each observer verifies that the local behaviour is consistent with some permitted behaviour of the system.
- Will this guarantee that the global behaviour is permitted?
  - Not Always. If and only if the system has no implied scenarios.
- Local testability is equivalent to absence of implied scenarios.
  - Local Testability is not decidable.

Combined Observations

What if we have observers that record the behaviours of sets of processes?

Combined Observations

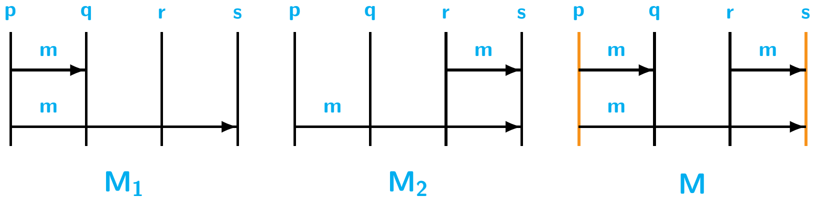
What if we have observers that record the behaviours of sets of processes?



M_1 M_2 M

Combined Observations

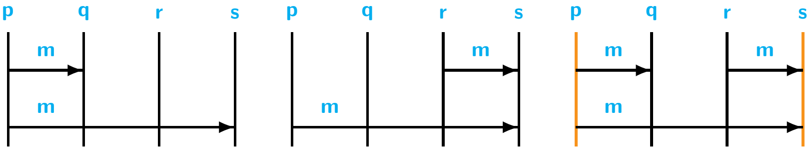
What if we have observers that record the behaviours of sets of processes?



M is detected as an illegal MSC.

Combined Observations

What if we have observers that record the behaviours of sets of processes?



$\mathbf{M_1}$ $\mathbf{M_2}$ $\mathbf{M}$

$\mathbf{M}$ is detected as an illegal MSC.

Observers recording multiple processes can detect more violations.

Combined Observations ...

- Fix a set of Observers $1, 2, \ldots r$.
- Observer i records the events on the processes in the set P_i.

Combined Observations ...

- Fix a set of Observers $\mathbf{1}, \mathbf{2}, \ldots \mathbf{r}$.
- Observer $\mathbf{i}$ records the events on the processes in the set $\mathbf{P_i}$.

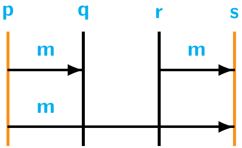
Given a HMSC $\mathbf{G}$ can we decide whether its language is testable by the observers $(\mathbf{P_i})_{1 \leq i \leq r}$?

P-Observations

Let **M** be an MSC. A **P**-observation of **M** w.r.t. a set of processes **P** is the tuple of words consisting of the projection of **M** on each process in **P**.

P-Observations

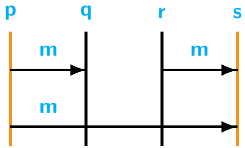
Let **M** be an MSC. A **P**-observation of **M** w.r.t. a set of processes **P** is the tuple of words consisting of the projection of **M** on each process in **P**.



M

P-Observations

Let **M** be an MSC. A **P**-observation of **M** w.r.t. a set of processes **P** is the tuple of words consisting of the projection of **M** on each process in **P**.



M

The $\{p, s\}$-observation of **M** is (**p!q(m)p!s(m)**,**s?r(m)s?p(m)**).

$\mathbf{P}$-Observations

Let $\mathbf{M}$ be an MSC. A $\mathbf{P}$-observation of $\mathbf{M}$ w.r.t. a set of processes $\mathbf{P}$ is the tuple of words consisting of the projection of $\mathbf{M}$ on each process in $\mathbf{P}$.

- We write $\mathbf{M}\restriction_{\mathbf{P}}$ for the $\mathbf{P}$-observation of $\mathbf{M}$.

$\mathbf{P}$-Observations

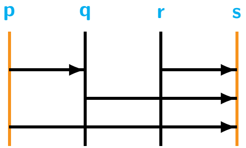
Let $\mathbf{M}$ be an MSC. A $\mathbf{P}$-observation of $\mathbf{M}$ w.r.t. a set of processes $\mathbf{P}$ is the tuple of words consisting of the projection of $\mathbf{M}$ on each process in $\mathbf{P}$.

- We write $\mathbf{M}\restriction_{\mathbf{P}}$ for the $\mathbf{P}$-observation of $\mathbf{M}$.
- We can also formulate $\mathbf{P}$-observation as a partial order, where the causality between processes is induced by messages both sent and received by processes in $\mathbf{P}$.

P-Observations

Let $\mathbf{M}$ be an MSC. A $\mathbf{P}$-observation of $\mathbf{M}$ w.r.t. a set of processes $\mathbf{P}$ is the tuple of words consisting of the projection of $\mathbf{M}$ on each process in $\mathbf{P}$.

- We write $\mathbf{M}\restriction_{\mathbf{P}}$ for the $\mathbf{P}$-observation of $\mathbf{M}$.
- We can also formulate $\mathbf{P}$-observation as a partial order, where the causality between processes is induced by messages both sent and received by processes in $\mathbf{P}$.



$\mathbf{M'}$

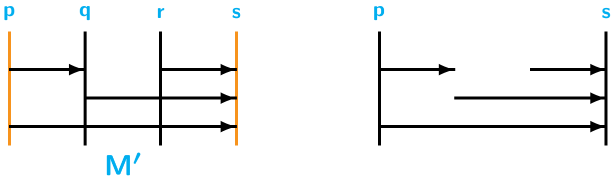
The $\{p, s\}$-observation of $\mathbf{M'}$ is

$$(p!q()p!s(),s?r()s?q()s?p())$$

P-Observations

Let **M** be an MSC. A **P**-observation of **M** w.r.t. a set of processes **P** is the tuple of words consisting of the projection of **M** on each process in **P**.

- We write $\mathbf{M}\restriction_{\mathbf{P}}$ for the **P**-observation of **M**.
- We can also formulate **P**-observation as a partial order, where the causality between processes is induced by messages both sent and received by processes in **P**.



The $\{p, s\}$-observation of $\mathbf{M'}$ is

$$(p!q()p!s(),s?r()s?q()s?p())$$

$\mathbf{P}$-Observations

Let $\mathbf{M}$ be an MSC. A $\mathbf{P}$-observation of $\mathbf{M}$ w.r.t. a set of processes $\mathbf{P}$ is the tuple of words consisting of the projection of $\mathbf{M}$ on each process in $\mathbf{P}$.

- We write $\mathbf{M}{\restriction}_{\mathbf{P}}$ for the $\mathbf{P}$-observation of $\mathbf{M}$.
- We can also formulate $\mathbf{P}$-observation as a partial order, where the causality between processes is induced by messages both sent and received by processes in $\mathbf{P}$.
- For a language $\mathbf{L}$, the $\mathbf{P}$-observation of $\mathbf{L}$ is given by $\mathbf{L}{\restriction}_{\mathbf{P}} \quad = \quad \{\mathbf{M}{\restriction}_{\mathbf{P}} \mid \mathbf{M} \in \mathbf{L}\}$

k-testing

- Record all P-observations where P is any set of processes of size k.

k-testing

- Record all P-observations where P is any set of processes of size k.
- The k-closure of a language L is the set.

$$\text{k-closure}(L) \quad = \quad \{M \mid \forall P{:}|P|{=}k.\ \exists M'{\in}L.\ M{\restriction_P}{=}M'{\restriction_P}\}$$

k-testing

- Record all $\mathbf{P}$-observations where $\mathbf{P}$ is any set of processes of size $\mathbf{k}$.
- The **k**-closure of a language $\mathbf{L}$ is the set.

$$\mathsf{k}\text{-closure}(\mathbf{L}) \quad = \quad \{\mathbf{M} \mid \forall \mathbf{P}{:}|\mathbf{P}|{=}\mathbf{k}.\ \exists \mathbf{M}'{\in}\mathbf{L}.\ \mathbf{M}{\restriction_{\mathbf{P}}}{=}\mathbf{M}'{\restriction_{\mathbf{P}}}\}$$

- A **k**-implied scenario $\mathbf{M}$ for a language $\mathbf{L}$ is a MSC that is in the $\mathbf{k}$-closure of $\mathbf{L}$ but not in $\mathbf{L}$.

k-testing

- Record all P-observations where P is any set of processes of size k.
- The k-closure of a language L is the set.

$$\text{k-closure}(L) \quad = \quad \{M \mid \forall P{:}|P|{=}k.\ \exists M'{\in}L.\ M{\restriction_P}{=}M'{\restriction_P}\}$$

- A k-implied scenario M for a language L is a MSC that is in the k-closure of L but not in L.
- A language is k-testable if it equals its k-closure.

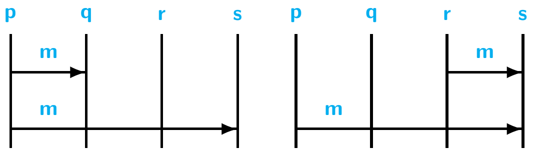
k-testing

- Record all P-observations where P is any set of processes of size k.
- The k-closure of a language L is the set.

$$k\text{-closure}(L) \quad = \quad \{M \mid \forall P{:}|P|{=}k.\ \exists M'{\in}L.\ M\restriction_P = M'\restriction_P\}$$

- A k-implied scenario M for a language L is a MSC that is in the k-closure of L but not in L.
- A language is k-testable if it equals its k-closure.
  Weak Realizability is 1-testability

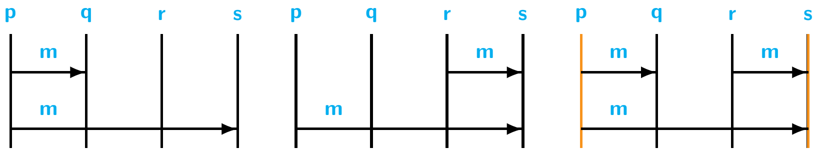
k-testability ...



M_1 M_2

The set $\{M_1, M_2\}$ is 2-testable but not 1-testable.

k-testability ...



M_1 M_2 M

The set $\{M_1, M_2\}$ is 2-testable but not 1-testable.

k-testability ...

$\mathbf{k}$-testability ...

- For all $\mathbf{n}$ and $\mathbf{k} < \mathbf{n}$ there are collections of MSCs over $\mathbf{n}$ processes that are not $\mathbf{k}$-testable.

k-testability ...

- For all n and $k < n$ there are collections of MSCs over n processes that are not k-testable.
- 1-testability is undecidable for 4 or more processes. [AY99].

$\mathbf{k}$-testability ...

- For all $\mathbf{n}$ and $\mathbf{k} < \mathbf{n}$ there are collections of MSCs over $\mathbf{n}$ processes that are not $\mathbf{k}$-testable.
- $\mathbf{1}$-testability is undecidable for $\mathbf{4}$ or more processes. [AY99].
- $\mathbf{n}$-testability is decidable.

$\mathbf{k}$-testability ...

- For all $\mathbf{n}$ and $\mathbf{k} < \mathbf{n}$ there are collections of MSCs over $\mathbf{n}$ processes that are not $\mathbf{k}$-testable.
- $\mathbf{1}$-testability is undecidable for $\mathbf{4}$ or more processes. [AY99].
- $\mathbf{n}$-testability is decidable.
  - What about $\mathbf{k}$-testability for $\mathbf{1} < \mathbf{k} < \mathbf{n}$?

k-testability ...

- For all n and $k < n$ there are collections of MSCs over n processes that are not k-testable.
- 1-testability is undecidable for 4 or more processes. [AY99].
- n-testability is decidable.
  - What about k-testability for $1 < k < n$?
  - What is the smallest $k \leq n$ such that k-testability is decidable?

$\mathbf{k}$-testability ...

- For all $\mathbf{n}$ and $\mathbf{k} < \mathbf{n}$ there are collections of MSCs over $\mathbf{n}$ processes that are not $\mathbf{k}$-testable.
- $\mathbf{1}$-testability is undecidable for $\mathbf{4}$ or more processes. [AY99].
- $\mathbf{n}$-testability is decidable.
  - What about $\mathbf{k}$-testability for $\mathbf{1} < \mathbf{k} < \mathbf{n}$?

    $\mathbf{k}$-testability is undecidable for all $\mathbf{1} \leq \mathbf{k} < \mathbf{n}$ and all $\mathbf{n} > \mathbf{1}$.
  - What is the smallest $\mathbf{k} \leq \mathbf{n}$ such that $\mathbf{k}$-testability is decidable?

    $\mathbf{k} = \mathbf{n}$

Modified PCP

We reduce the MPCP (Modified PCP) from Hopcroft and Ullman to k-testability.

Modified PCP

We reduce the MPCP (Modified PCP) from Hopcroft and Ullman to $\mathbf{k}$-testability.

MPCP: Given a sequence $(\mathbf{v_1}, \mathbf{w_1}), (\mathbf{v_2}, \mathbf{w_2}) \dots (\mathbf{v_r}, \mathbf{w_r})$ of words (over some finite alphabet $\boldsymbol{\Sigma}$) is there a sequence of integers $\mathbf{1}, \mathbf{i_2}, \mathbf{i_3} \dots \mathbf{i_m}$ such that

$$\mathbf{v_1 v_{i_2} \dots v_{i_m}} \;=\; \mathbf{w_1 w_{i_2} \dots w_{i_m}}?$$

Modified PCP

We reduce the MPCP (Modified PCP) from Hopcroft and Ullman to $\mathbf{k}$-testability.

MPCP: Given a sequence $(\mathbf{v_1}, \mathbf{w_1}), (\mathbf{v_2}, \mathbf{w_2}) \ldots (\mathbf{v_r}, \mathbf{w_r})$ of words (over some finite alphabet $\boldsymbol{\Sigma}$) is there a sequence of integers $\mathbf{1}, \mathbf{i_2}, \mathbf{i_3} \ldots \mathbf{i_m}$ such that

$$\mathbf{v_1 v_{i_2} \ldots v_{i_m}} = \mathbf{w_1 w_{i_2} \ldots w_{i_m}}?$$

We may assume that

1. the given instance has a solution if and only if, in addition to the above, for each $\mathbf{l} < \mathbf{m}$, $\mathbf{w_1 w_{i_2} \ldots w_{i_l}}$ is a proper prefix of $\mathbf{v_1 v_{i_2} \ldots v_{i_l}}$.
2. $|\mathbf{v_1}| > |\mathbf{w_1}| + \mathbf{1}$.

Modified PCP

We reduce the MPCP (Modified PCP) from Hopcroft and Ullman to $\mathbf{k}$-testability.

MPCP: Given a sequence $(\mathbf{v_1}, \mathbf{w_1}), (\mathbf{v_2}, \mathbf{w_2}) \ldots (\mathbf{v_r}, \mathbf{w_r})$ of words (over some finite alphabet $\mathbf{\Sigma}$) is there a sequence of integers $1, \mathbf{i_2}, \mathbf{i_3} \ldots \mathbf{i_m}$ such that

$$\mathbf{v_1 v_{i_2} \ldots v_{i_m}} = \mathbf{w_1 w_{i_2} \ldots w_{i_m}}?$$

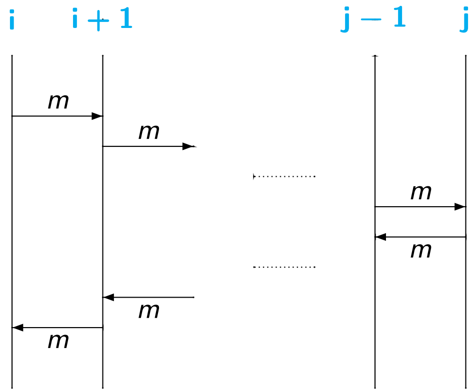
We may assume that

1. the given instance has a solution if and only if, in addition to the above, for each $\mathbf{l} < \mathbf{m}$, $\mathbf{w_1 w_{i_2} \ldots w_{i_l}}$ is a proper prefix of $\mathbf{v_1 v_{i_2} \ldots v_{i_l}}$.
2. $|\mathbf{v_1}| > |\mathbf{w_1}| + 1$.

The MPCP problem is undecidable.

Undecidability

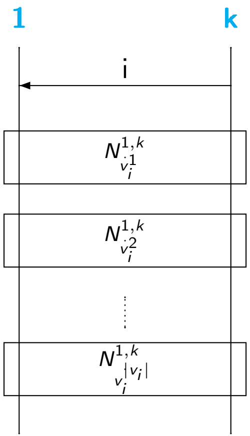
For each pair of processes $\mathbf{i, j}$ with $\mathbf{i < j}$ the MSC $\mathbf{N^m_{ij}}$ is as follows:



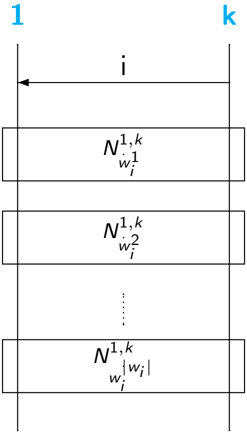
Undecidability ...

For each pair (v_i, w_i) we define three MSCs associated with the pair.

The MSC $\mathbf{M_{v_i}}$

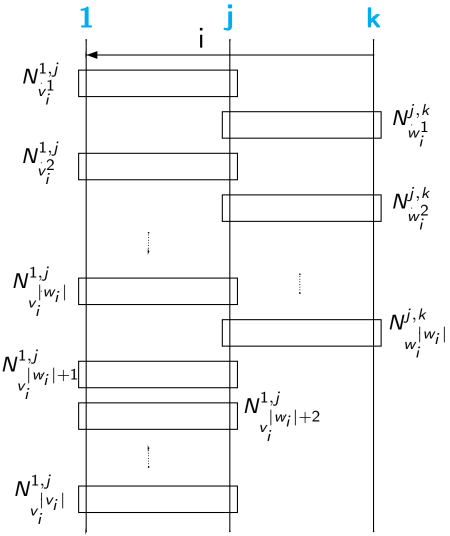


The MSC $\mathbf{M_{w_i}}$



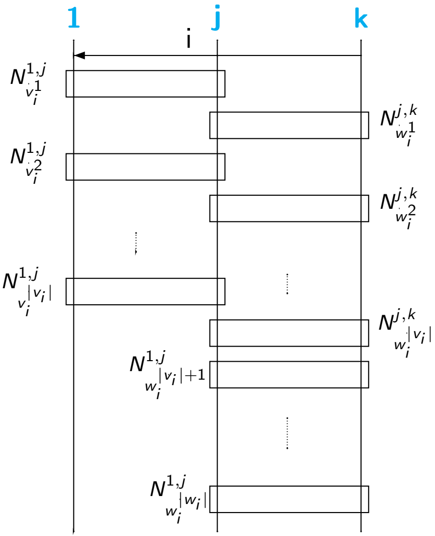
Undecidability ...

The MSC $\mathbf{M^{j}_{v_i w_i}}$



Undecidability ...

The MSC $\mathbf{M_{v_i w_i}}$



Undecidability ...

Let

$$\begin{array}{rcl} L_v & = & M_{v_1}.\{M_{v_i} \mid 1 \leq i \leq r\}^* \\ L_w & = & M_{w_1}.\{M_{w_i} \mid 1 \leq i \leq r\}^* \\ L^j_{vw} & = & M^j_{v_1 w_1}.\{M^j_{v_i w_i} \mid 1 \leq i \leq r\}^* \end{array}$$

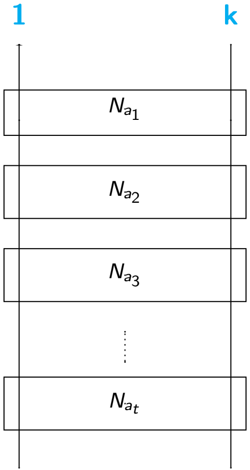
and

$$\mathbf{L_\Delta} \;=\; \mathbf{L_v} \cup \mathbf{L_w} \cup \bigcup_{\mathbf{1<j<k}} \mathbf{L^j_{vw}}$$

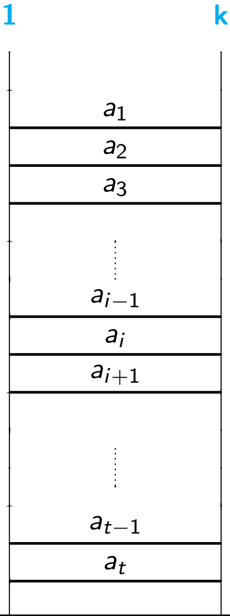
Then, $\mathbf{L_\Delta}$ has a $\mathbf{(k-1)}$-implied scenario if and only if the given instance of MPCP has a solution.

Undecidability ...

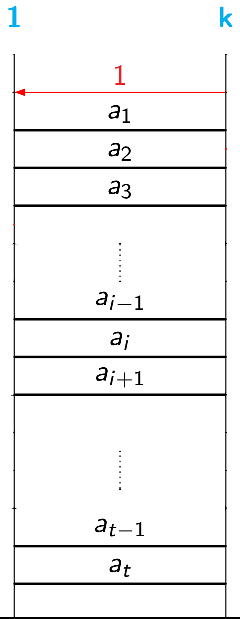
Suppose $\mathbf{a_1 a_2 \ldots a_t} \;=\; \mathbf{v_1 v_{i_2} \ldots v_{i_m}} \;=\; \mathbf{w_1 w_{i_2} \ldots w_{i_m}}$



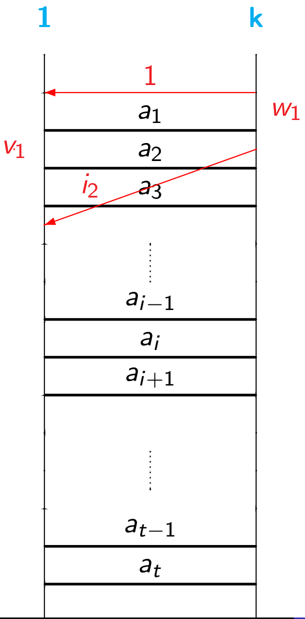
Undecidability ...



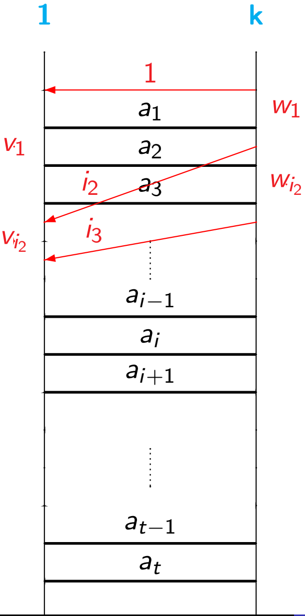
Undecidability ...



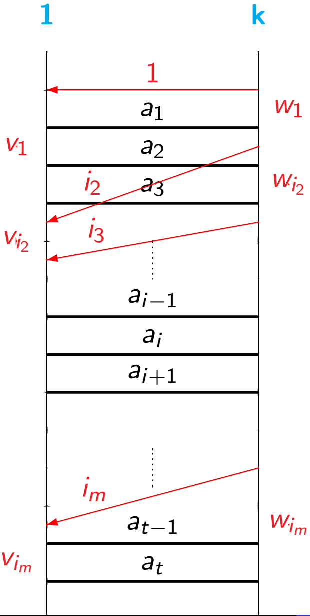
Undecidability ...



Undecidability ...

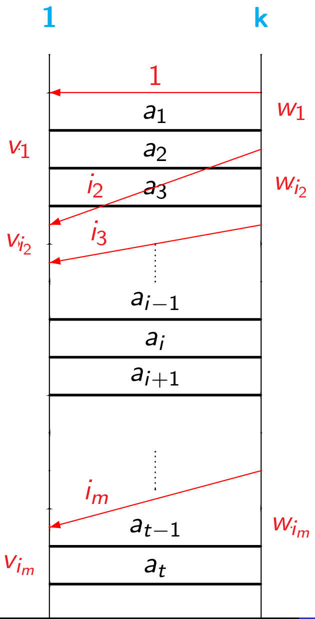


Undecidability ...



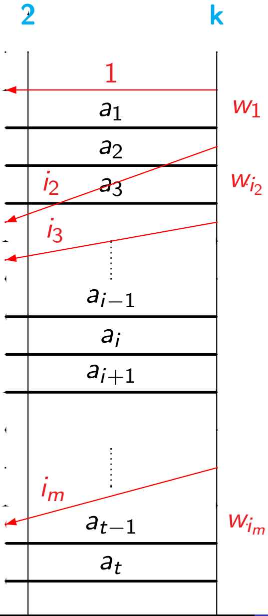
The MSC **M**

Undecidability ...



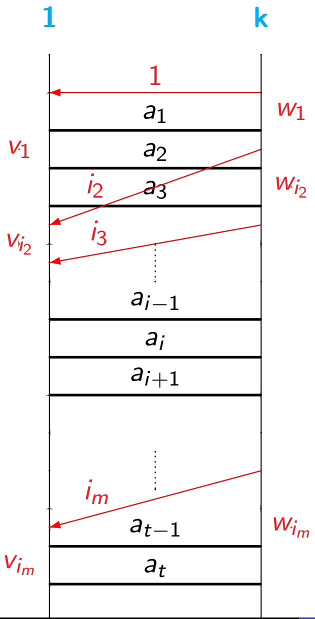
The MSC **M**

Undecidability ...



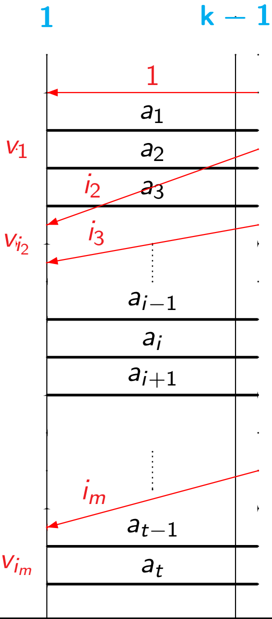
$$\mathbf{M}\restriction_{\{2,3,\ldots,k\}} \in \mathbf{L_w}\restriction_{\{2,3,\ldots,k\}}$$

Undecidability ...



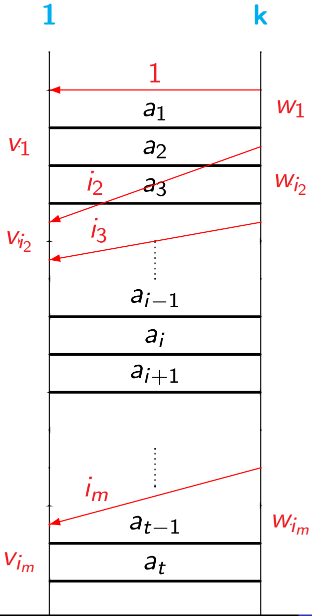
The MSC **M**

Undecidability ...



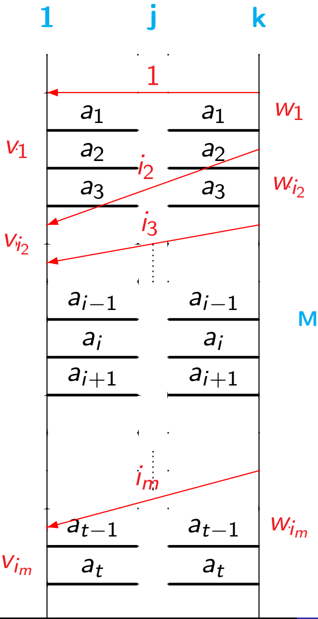
$$\mathbf{M}\restriction_{\{1,2,\ldots,k-1\}} \in \mathbf{L_v}\restriction_{\{1,2,\ldots,k-1\}}$$

Undecidability ...



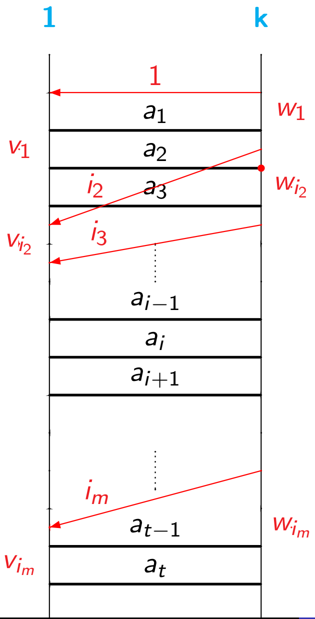
The MSC **M**

Undecidability ...



$\mathbf{M}\restriction_{\{1,\ldots,j-1,j+1,\ldots,k\}} \in \mathbf{L}^{j}_{vw}\restriction_{\{1,\ldots,j-1,j+1,\ldots,k\}}$

Undecidability ...



$\mathbf{M} \not\in \mathbf{L}_{\Delta}$

Undecidability ...

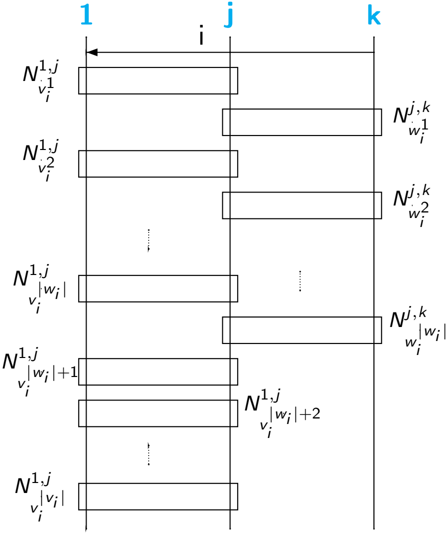
Let $\mathbf{M}$ be a $(\mathbf{k}-\mathbf{1})$-implied scenario.

Undecidability ...

Undecidability ...

Case 1: $\exists \mathbf{j} : \mathbf{1} < \mathbf{j} < \mathbf{k}.\ \mathbf{M}\restriction_{\mathbf{j}} \in \mathbf{L}^{\mathbf{j}}_{\mathbf{vw}}\restriction_{\mathbf{j}}$

Undecidability ...



Undecidability ...

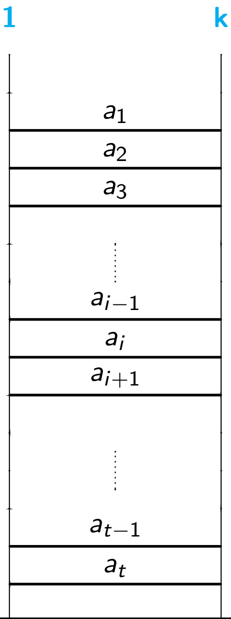
Case 1: $\exists \mathbf{j} : \mathbf{1} < \mathbf{j} < \mathbf{k}.\ \mathbf{M}\restriction_{\mathbf{j}} \in \mathbf{L}^{\mathbf{j}}_{\mathbf{vw}}\restriction_{\mathbf{j}}$

$\mathbf{M} \in \mathbf{L}_{\Delta}$

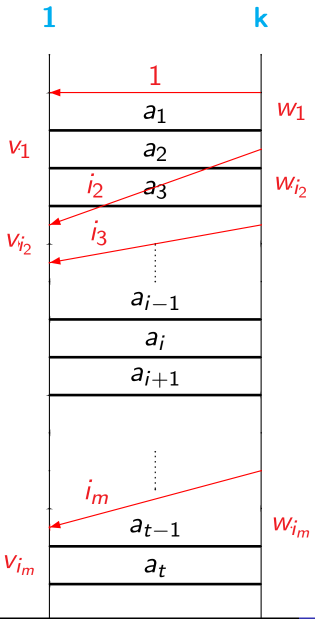
Undecidability ...

Case 2: $\forall \mathbf{j : 1 < j < k.\ M}\restriction_{\mathbf{j}} \in \mathbf{L_v}\restriction_{\mathbf{j}}$

Undecidability ...



Undecidability ...



The MSC **M**

Undecidability ...

Case 2: $\forall \mathbf{j} : \mathbf{1} < \mathbf{j} < \mathbf{k}.\ \mathbf{M}\restriction_{\mathbf{j}} \in \mathbf{L_v}\restriction_{\mathbf{j}}$

$\mathbf{M}$ codes a solution to the MPCP.

More Undecidability

For $\mathbf{n = 2}$ processes, we can once again reduce MPCP (albeit via a different reduction) to $\mathbf{1}$-testability.

More Undecidability

For $\mathbf{n = 2}$ processes, we can once again reduce MPCP (albeit via a different reduction) to $\mathbf{1}$-testability.

$\mathbf{(k-1)}$-testability is undecidable for $\mathbf{k > 1}$.

More Undecidability

For $\mathbf{n = 2}$ processes, we can once again reduce MPCP (albeit via a different reduction) to $\mathbf{1}$-testability.

$\mathbf{(k-1)}$-testability is undecidable for $\mathbf{k > 1}$.

By adding $\mathbf{n-k}$ inactive processes to this construction we obtain the undecidability of $\mathbf{k}$-testability for all $\mathbf{k < n}$ and $\mathbf{n \geq 3}$.

More Undecidability

For $\mathbf{n = 2}$ processes, we can once again reduce MPCP (albeit via a different reduction) to $\mathbf{1}$-testability.

$\mathbf{(k - 1)}$-testability is undecidable for $\mathbf{k > 1}$.

By adding $\mathbf{n - k}$ inactive processes to this construction we obtain the undecidability of $\mathbf{k}$-testability for all $\mathbf{k < n}$ and $\mathbf{n \geq 3}$.

Do these results hold even when the message alphabet is singleton?

More Undecidability

For $n = 2$ processes, we can once again reduce MPCP (albeit via a different reduction) to 1-testability.

$(k - 1)$-testability is undecidable for $k > 1$.

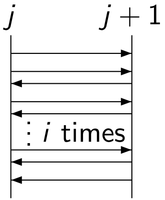
By adding $n - k$ inactive processes to this construction we obtain the undecidability of k-testability for all $k < n$ and $n \geq 3$.

Do these results hold even when the message alphabet is singleton?

k-testability is undecidable for $1 < k < n - 1$ for all $n > 3$ even when the message alphabet is singleton.

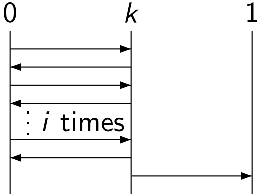
Eliminating Messages

The message **i** from a process to it neighbour can be replaced by the following MSC:



Eliminating Messages ...

The message **i** from **k** to **1** cannot be dealt with similarly. We must permit arbitrary delays in the delivery of this message.



1-testability over the singleton alphabet

- Each channel behaves as counter.

1-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.

1-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.
- Recall that the given HMSC is **B**-bounded.

$\mathbf{1}$-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.
- Recall that the given HMSC is $\mathbf{B}$-bounded.
    1. Check if the Net can reach a configuration where one of the channels has $\mathbf{B+1}$ messages and from where a final marking is reachable.

1-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.
- Recall that the given HMSC is B-bounded.
  1. Check if the Net can reach a configuration where one of the channels has $B + 1$ messages and from where a final marking is reachable.
  2. If the answer to 1 is yes, then say No.

1-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.
- Recall that the given HMSC is $\mathbf{B}$-bounded.
    1. Check if the Net can reach a configuration where one of the channels has $\mathbf{B+1}$ messages and from where a final marking is reachable.
    2. If the answer to 1 is yes, then say No.
    3. Else, check if the $\mathbf{B}$-bounded language of the net has any words not in the language of the HMSC.

$\mathbf{1}$-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.
- Recall that the given HMSC is $\mathbf{B}$-bounded.
  1. Check if the Net can reach a configuration where one of the channels has $\mathbf{B+1}$ messages and from where a final marking is reachable.
  2. If the answer to 1 is yes, then say No.
  3. Else, check if the $\mathbf{B}$-bounded language of the net has any words not in the language of the HMSC.

This is a special case of a result due to R. Morin ([M02]).

1-testability over the singleton alphabet

- Each channel behaves as counter.
- We may code the behaviours of such a HMSC as a Petri Net.
- Recall that the given HMSC is **B**-bounded.
  1. Check if the Net can reach a configuration where one of the channels has $\mathbf{B+1}$ messages and from where a final marking is reachable.
  2. If the answer to 1 is yes, then say No.
  3. Else, check if the **B**-bounded language of the net has any words not in the language of the HMSC.

This is a special case of a result due to R. Morin ([M02]).
This also gives an algorithm to check if the **1**-closure of the HMSC is regular, since the infiniteness of the number of intermediate markings of a Net is a decidable problem.

Specification to Implementation

- Convert scenarios into executable form — set of communicating finite state-machines

  Message Passing Automata (MPA)

Specification to Implementation

- Convert scenarios into executable form — set of communicating finite state-machines

  Message Passing Automata (MPA)

- Execution model

Specification to Implementation

- Convert scenarios into executable form — set of communicating finite state-machines

  Message Passing Automata (MPA)

- Execution model
  - Each component is finite state

Specification to Implementation

- Convert scenarios into executable form — set of communicating finite state-machines

  Message Passing Automata (MPA)

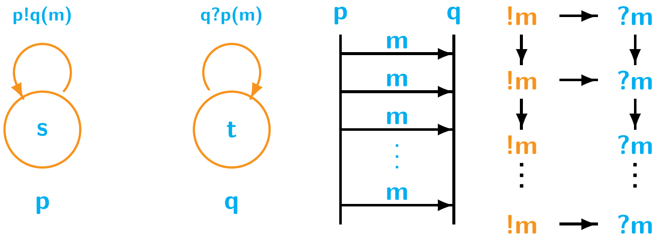
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels

Specification to Implementation

- Convert scenarios into executable form — set of communicating finite state-machines

  Message Passing Automata (MPA)

- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels



Specification to Implementation

- Convert scenarios into executable form — set of communicating finite state-machines

  Message Passing Automata (MPA)

- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels
  - Globally finite state $\Rightarrow$ channels are bounded

MPAs and Regular MSC Languages

- Message Passing Automata with bounded channels generate only regular MSC languages
- What about the converse?

Theorem: [[HMNST, I&C '05],[MNS, Concur'00]] Every regular MSC language is recognized by a deterministic message passing automaton with bounded channels.

- Messages are tagged with extra information.

MPAs and Regular MSC Languages

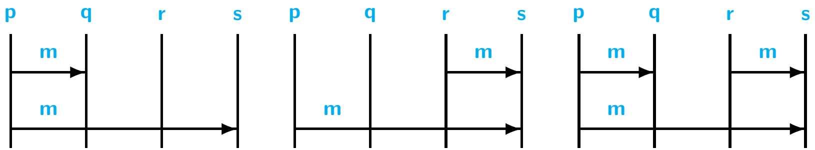
- Message Passing Automata with bounded channels generate only regular MSC languages
- What about the converse?

Theorem: [[HMNST, I&C '05],[MNS, Concur'00]] Every regular MSC language is recognized by a deterministic message passing automaton with bounded channels.

- Messages are tagged with extra information.
- Uses global accepting states.

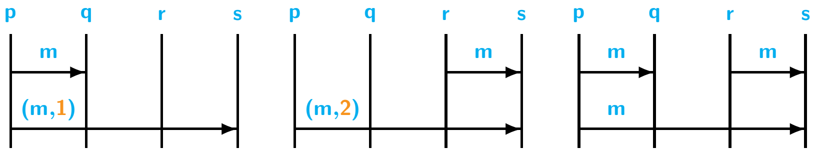
Adding Information to the Messages

Adding Information to the Messages



M_1 M_2 M

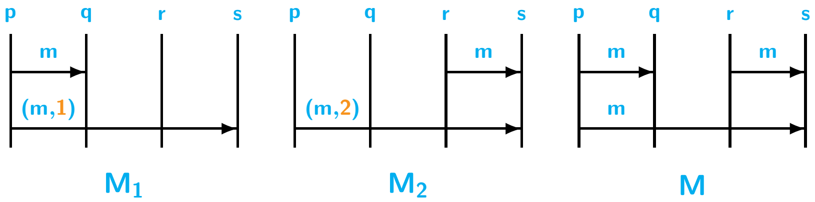
Adding Information to the Messages



M_1 M_2 M

- By tagging auxiliary information to **m**, **p** informs **s** whether it has sent a message to **q**

Adding Information to the Messages



- By tagging auxiliary information to **m**, **p** informs **s** whether it has sent a message to **q**
- This rules out the implied scenario **M**

Causal closure

- Recall that a MSC is a partially ordered set of events

Causal closure

- Recall that a MSC is a partially ordered set of events
- For a process **p** and a MSC **M**, **p**'s causal view of **M** is the set of all events in **M** that lie below some event of **p**

Causal closure

- Recall that a MSC is a partially ordered set of events
- For a process **p** and a MSC **M**, **p**'s causal view of **M** is the set of all events in **M** that lie below some event of **p**
- **M** is causally implied by **L** if each process **p**'s causal view of **M** matches its causal view of some MSC in **L**

Causal closure

- Recall that a MSC is a partially ordered set of events
- For a process **p** and a MSC **M**, **p**'s causal view of **M** is the set of all events in **M** that lie below some event of **p**
- **M** is causally implied by **L** if each process **p**'s causal view of **M** matches its causal view of some MSC in **L**
- An MSC language is causally closed if it is closed with respect to causal implication

Theorem:[[AMNN FSTTCS'05]] The causal closure of a regular MSC language is always regular and can be effectively constructed.

Causal closure . . .

- Given an automaton for L, we may tag each message with auxiliary information

Causal closure . . .

- Given an automaton for **L**, we may tag each message with auxiliary information
- Processes can use this auxiliary information to obtain information about the state of the rest of the system

Causal closure . . .

- Given an automaton for **L**, we may tag each message with auxiliary information
- Processes can use this auxiliary information to obtain information about the state of the rest of the system
- The causal closure of a regular MSC language **L** is always regular

Causal closure . . .

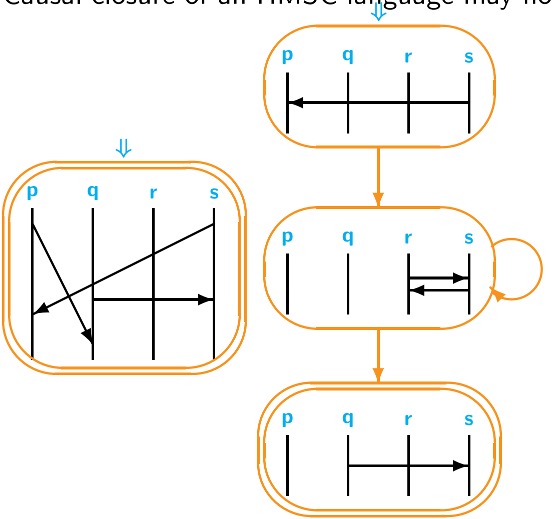
- Given an automaton for **L**, we may tag each message with auxiliary information
- Processes can use this auxiliary information to obtain information about the state of the rest of the system
- The causal closure of a regular MSC language **L** is always regular
- We can effectively construct a bounded message-passing automaton with local accepting states recognizing the causal closure.

HMSCs and causal closure

Causal closure of an HMSC language may not be finitely generated

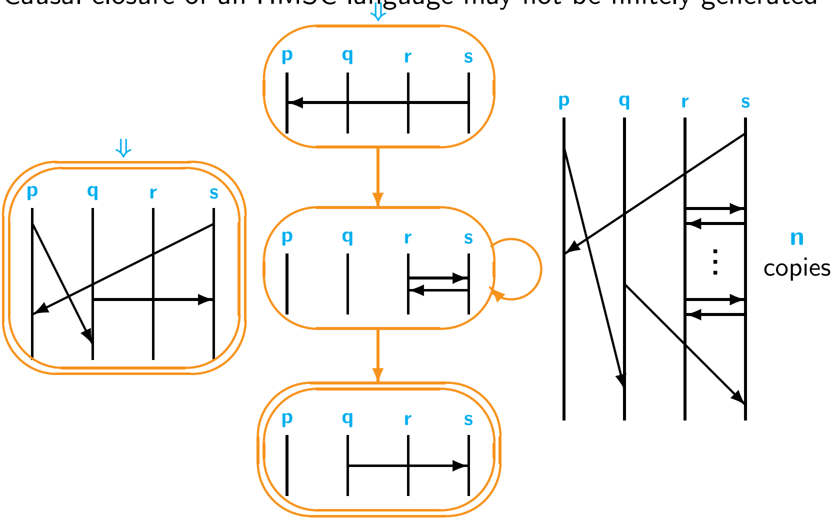
HMSCs and causal closure

Causal closure of an HMSC language may not be finitely generated



HMSCs and causal closure

Causal closure of an HMSC language may not be finitely generated



Summary and future work

- HMSCs – a formalism for specifying collections of scenarios.
  - attractive visual formalism
  - Sufficient condition for regularity.

Summary and future work

- HMSCs – a formalism for specifying collections of scenarios.
  - attractive visual formalism
  - Sufficient condition for regularity.
- Testability is undecidable in most situations.

Summary and future work

- HMSCs – a formalism for specifying collections of scenarios.
  - attractive visual formalism
  - Sufficient condition for regularity.
- Testability is undecidable in most situations.
- Look for sufficient conditions that indicate violation of testability.