

# Realizing and model-checking HMSCs

joint work with B. Genest<sup>\*</sup>, A. Muscholl<sup>\*</sup>, H. Seidl<sup>\*\*</sup>

(<sup>\*</sup>) LIAFA, CNRS, Univ. Paris 7. Supported by ARC FISC

(<sup>\*\*</sup>) Univ. Trier

# Outline of the talk

- High-level Message sequence charts (HMSCs)

# Outline of the talk

- High-level Message sequence charts (HMSCs)

Two problems...

- **Realizability** of HMSCs languages by CFMs
- **Model-checking** of HMSCs against HMSCs

# Outline of the talk

- High-level Message sequence charts (HMSCs)

Two problems...

- **Realizability** of HMSCs languages by CFMs
- **Model-checking** of HMSCs against HMSCs

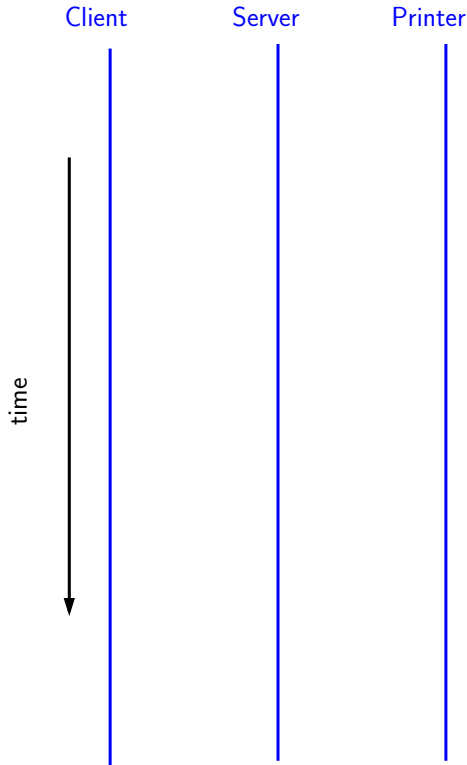
... answered for 3 infinite-state HMSC subclasses

- Globally-cooperative HMSCs
- Locally-cooperative HMSCs
- Local-choice HMSCs

# What is a Message sequence chart (MSC)?

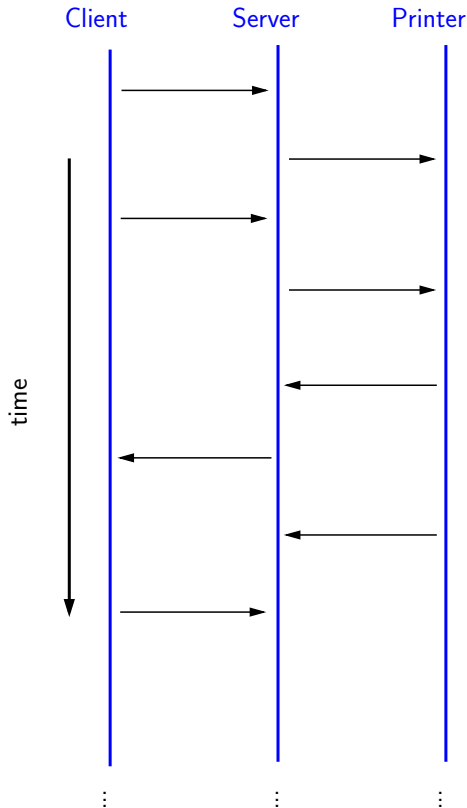
- **Graphical** description of one execution of a multi-threaded system
- Focus on (discrete) time line of each process and message exchanges
- Used at early design stage of protocols to check
  - requirements (completeness)
  - non-existence of undesired behaviors (consistency)
- **Standardized** ITU Z120 (1996, 2000), part of UML, available tools (ObjectGeode, RationalRose, Slim/SoFat)
- **Practical** applications (eg. code skeleton generation)

# Message sequence charts (MSC)



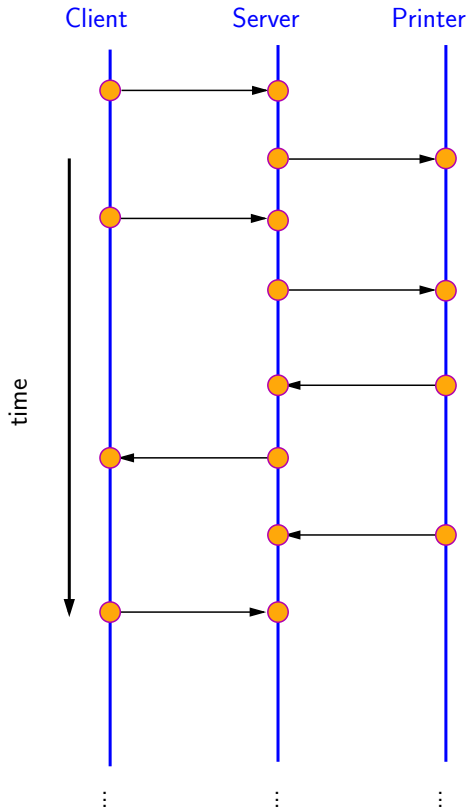
- Finite set of processes

# Message sequence charts (MSC)



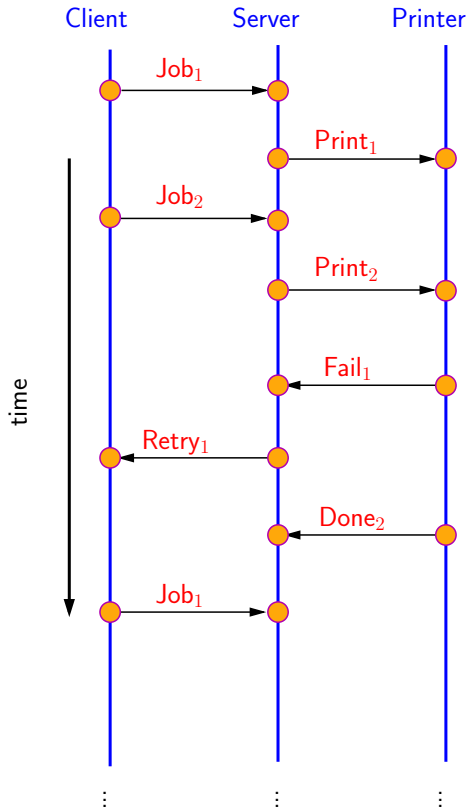
- Finite set of **processes**
- Point-to-point, reliable FIFO buffers

# Message sequence charts (MSC)



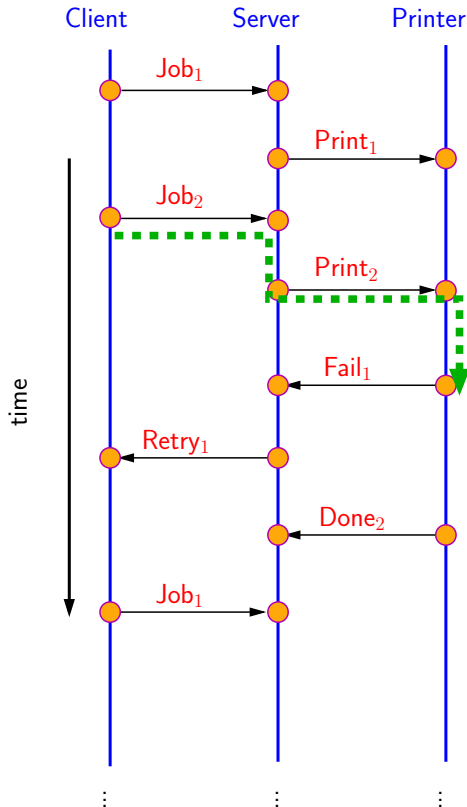
- Finite set of **processes**
- Point-to-point, reliable FIFO buffers
- **Events** of 2 types: send  $k!_{m,j}$ /receive  $k?_{m,j}$
- Each event is located on one process

# Message sequence charts (MSC)



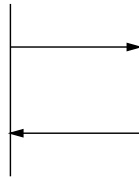
- Finite set of processes
- Point-to-point, reliable FIFO buffers
- Events of 2 types: send  $k!_{m,j}$ /receive  $k?_{m,j}$
- Each event is located on one process
- Messages contents

# Message sequence charts (MSC)

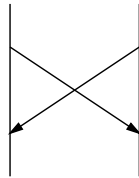


- Finite set of processes
- Point-to-point, reliable FIFO buffers
- Events of 2 types: send  $k!_{m,j}$ /receive  $k?_{m,j}$
- Each event is located on one process
- Messages contents
- Partial order on events (visual order  $\leq$ )
  - Total order  $\leq_p$  on events of process  $p$
  - Any 'send' before corresponding 'receive'  
→ acyclic relation

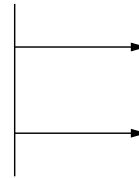
# MSC examples



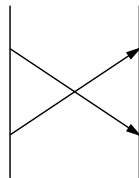
OK



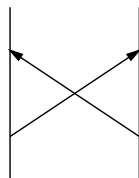
OK



OK



NOT FIFO



NO (CYCLIC)

# Asynchronous concatenation of MSCs

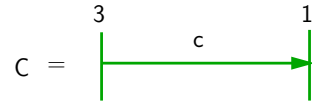
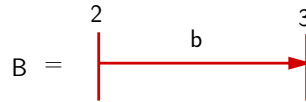
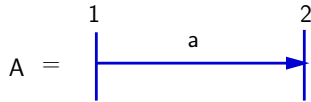
Concatenation process by process, **NO** synchronization

$A \cdot B$ : “glue B under A”

# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

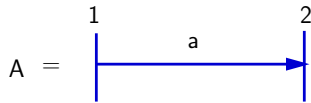
$A \cdot B$ : "glue B under A"



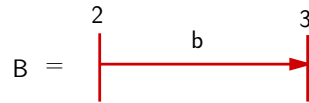
# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

$A \cdot B$ : "glue B under A"

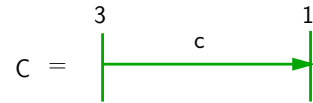


1



2

3



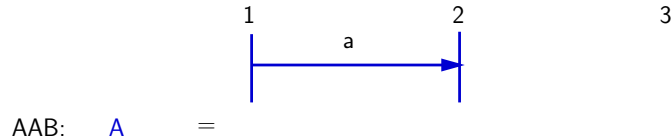
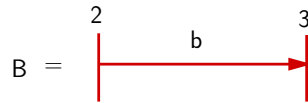
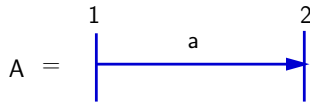
AAB:

=

# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

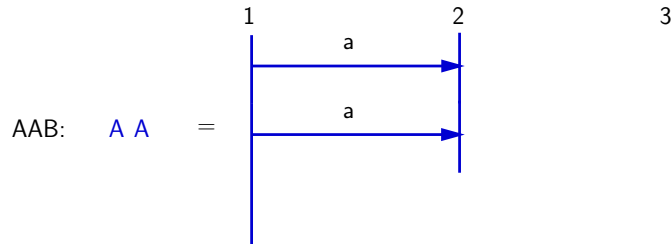
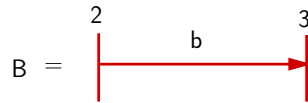
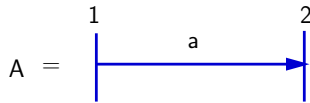
$A \cdot B$ : "glue B under A"



# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

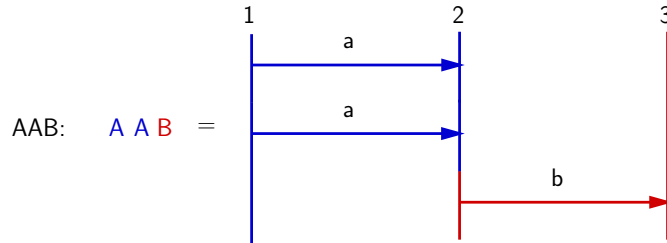
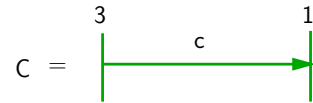
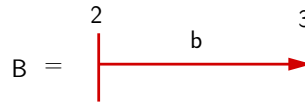
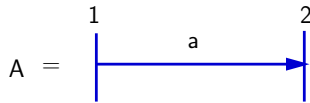
$A \cdot B$ : "glue B under A"



# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

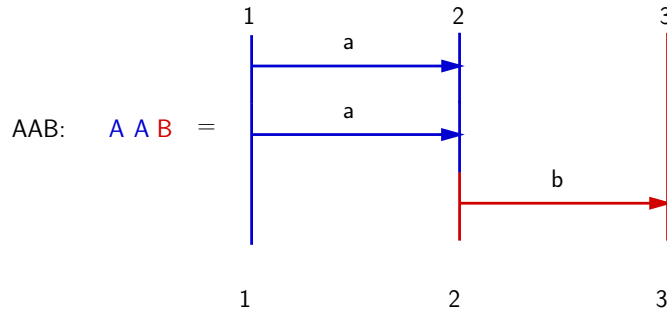
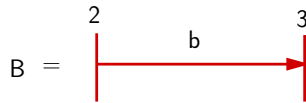
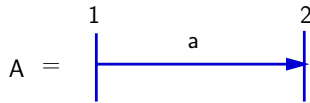
$A \cdot B$ : "glue B under A"



# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

$A \cdot B$ : "glue B under A"

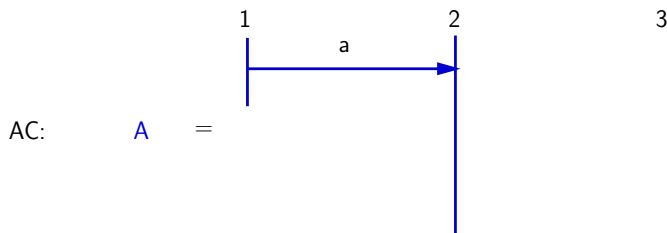
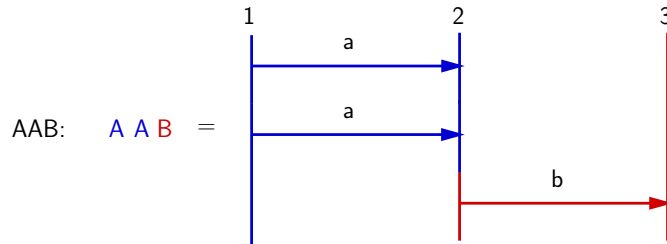
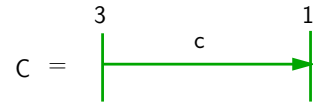
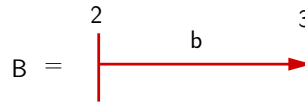
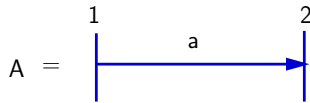


AC: =

# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

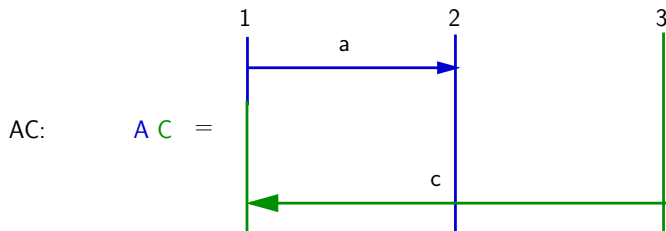
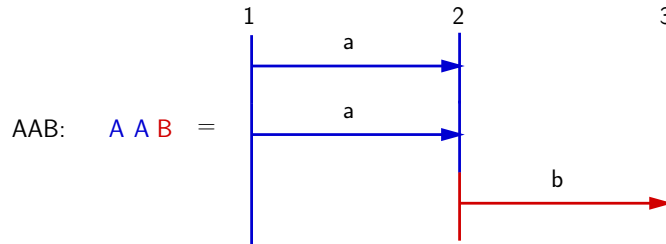
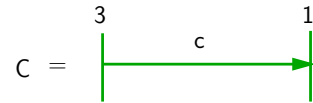
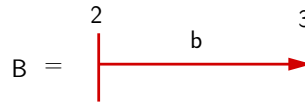
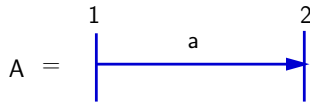
$A \cdot B$ : "glue B under A"



# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

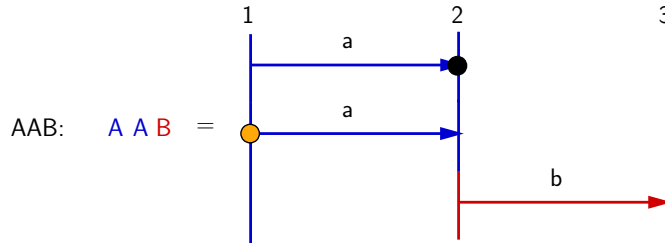
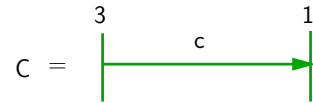
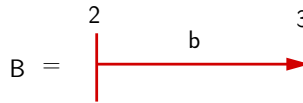
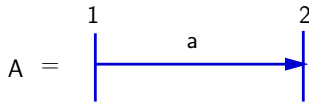
$A \cdot B$ : "glue B under A"



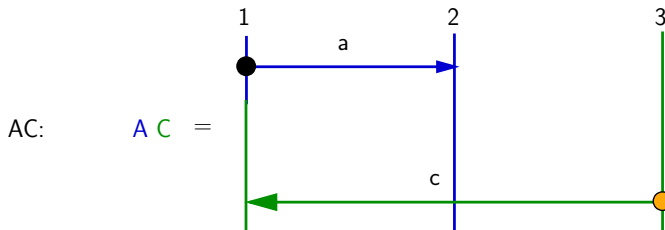
# Asynchronous concatenation of MSCs

Concatenation process by process, **NO** synchronization

$A \cdot B$ : "glue B under A"



● and ● are **concurrent** events



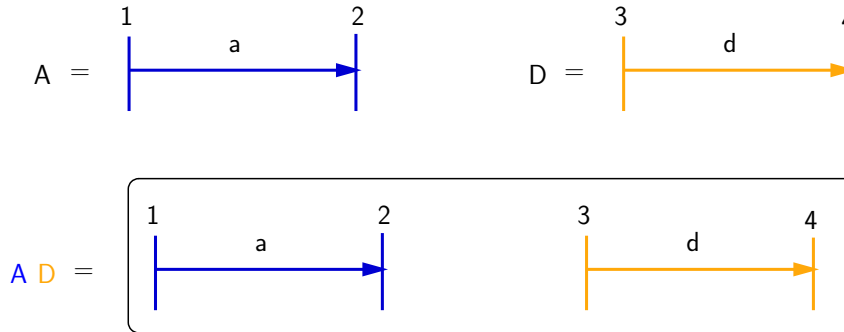
In a linearization of  $<$ ,

● can be executed **before** ●

even if it appears in a **later factor**

# Asynchronous concatenation and dependency

“Parallel” Concatenation of disjoint-process MSCs for free



$P(M)$  = set of communicating processes of  $M$

$$M_1 \parallel M_2 \text{ iff } P(M_1) \cap P(M_2) = \emptyset$$

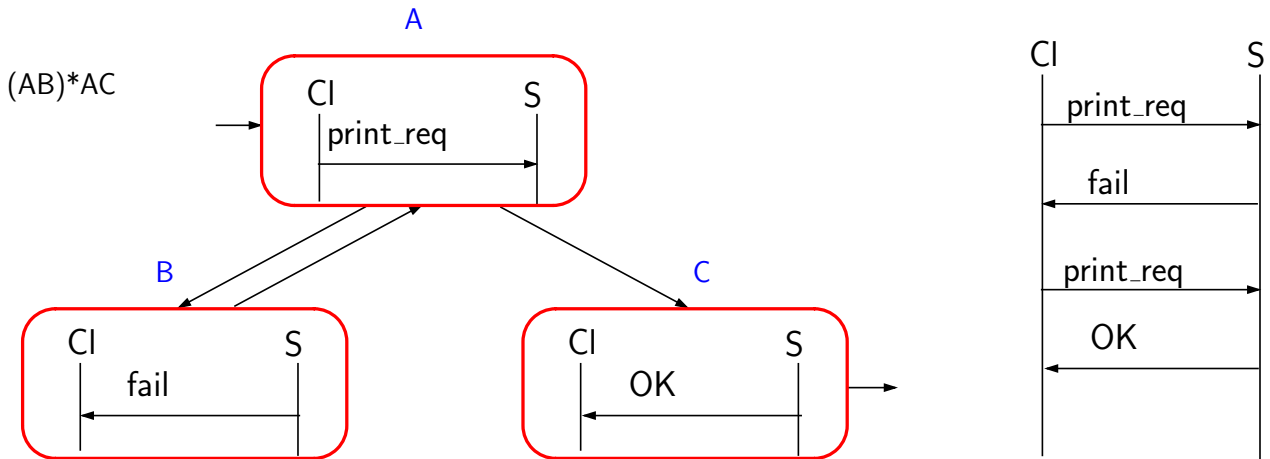
**Remark:**  $M_1 \parallel M_2 \implies M_1 M_2 = M_2 M_1$   
⟨Atomic MSCs, concatenation,  $\parallel$ ⟩ = trace monoid  $\mathcal{M}$

# High-level MSC (HMSC)

To describe protocols combining **several** scenarios: automata over  $\mathcal{M}$

**HMSC** = finite directed graph, initial, final states  
+ labeling of nodes by MSCs

**Semantics** = { asynchronous concatenations of successful paths }  $\subseteq \mathcal{M}$



# High-level MSC (HMSC)

HMSC = finite transition system  $G = \langle V, R, v^0, v^f, \lambda \rangle$  where

- $V$ : set of nodes,
- $R \subseteq V \times V$ : set of transitions,
- initial node  $v^0$ , no ingoing edge,
- terminal node  $v^f$ , no outgoing edge,
- each node  $v$  is labeled by a finite MSC  $\lambda(v)$ .

**Execution** of  $G$ : labeling  $\lambda(v_0)\lambda(v_1)\cdots\lambda(v_k)$  of some successful path  
 $v^0 = v_0, v_1, \dots, v_k = v^f$

# Regular HMSCs [HMKT00]

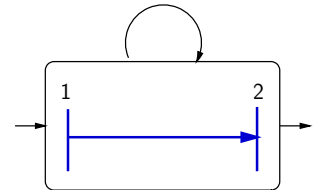
G HMSC       $\text{Lin}(G) = \{\text{linearizations of executions of } G\} \subseteq \text{Events}^*$

Definition G is a **regular** HMSC iff  $\text{Lin}(G)$  is regular

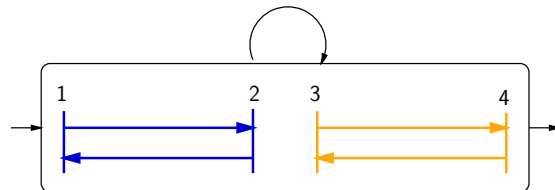
Remark G regular  $\implies$  G buffer-bounded

Characteristic causes of non-regularity [Mor01]

- Buffers may be unbounded



- Not recognizable in  $\mathcal{M}$



Aim: model-check and implement possibly **non-regular** HMSCs.

# Outline of the talk

- High-level Message sequence charts

## Two problems...

- **Realizability** of HMSCs languages by CFMs
- **Model checking** of HMSCs against HMSCs

... answered for 3 infinite-state HMSC subclasses

Globally-cooperative HMSCs

Locally-cooperative HMSCs

Local-choice HMSCs

# Realizability problem

Implementation model: Communicating Finite State Machines (CFM)  
with distributed control

A CFM  $C$  generates a sequence of events  $L(C)$

A CFM  $C$  is an implementation of and HMSC  $G$  if  $L(C) = \text{Lin}(G)$

# Realizability problem

Implementation model: Communicating Finite State Machines (CFM)  
with **distributed** control

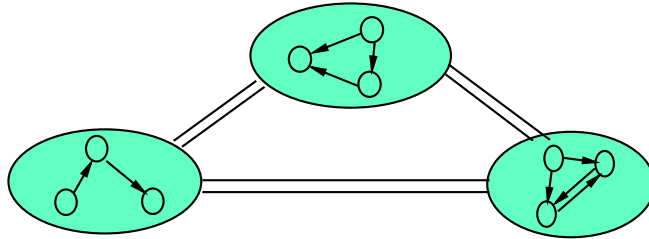
A CFM **C** generates a sequence of events  $L(\mathbf{C})$

A CFM **C** is an **implementation** of and HMSC **G** if  $L(\mathbf{C}) = \text{Lin}(\mathbf{G})$

## Realizability problem

- Input: HMSC **G**.
  - Question: Does there exist an implementation for **G**?  
If so, is the implementation computable?
- ▷ **Distribute the centralized control of an HMSC over processes**

# Communicating Finite State Machines (CFM)



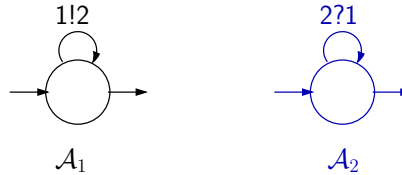
$\langle \mathcal{A}_k, \mathcal{B}_{i,j}, \mathcal{C} \rangle_{1 \leq i,j,k \leq n}$  = finite automata communicating via FIFO buffers

- $\mathcal{A}_k = (\Sigma_k, Q_k, i_k, \rightarrow_k, F_k)$  automaton representing process  $k$
- Point-to-point, **reliable FIFO** buffer  $\mathcal{B}_{i,j}$  from  $i$  to  $j$  for  $i \neq j$
- Finite alphabet  $\mathcal{C}$  of **message contents**.

- Alphabets of events  $\Sigma_k = \bigcup_{j \neq k, m \in \mathcal{C}} \{k!_{m,j}, k?_{m,j}\}$   $1 \leq k \leq n$

# Communicating Finite State Machines (CFM)

Example:  $C = \langle \mathcal{A}_1, \mathcal{A}_2 \rangle$

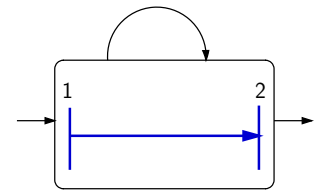


## Computation

- begins in state  $s^0 = (i_1, \dots, i_n)$ , ends in  $\prod F_k$ , with empty buffers.
- a transition by  $k!_m j$  appends message  $\mathbf{m}$  in buffer  $k \rightarrow j$
- a transition by  $k?_m j$  removes message  $\mathbf{m}$  in buffer  $j \rightarrow k$

$L(C) =$  set of sequences of transition labels of computations.

The CFM of the example is an implementation of



# Standard CFM [AEY00]

For an HMSC  $G$ , standard CFM of  $G$  obtained as follows

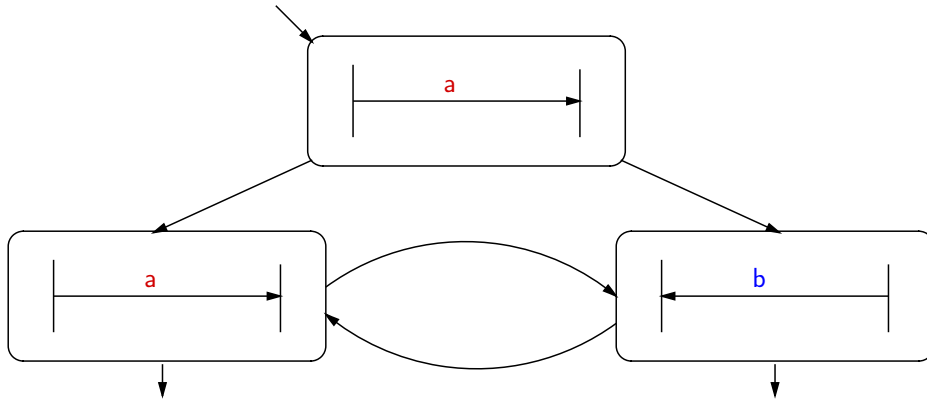
$\mathcal{A}_k$  recognizes the projection of the language  $L(G)$  on process  $k$

- The standard CFM of  $G$  recognizes **at least** the language of  $G$
- Realizability  $\iff$  realizability by the standard CFM [Mo02]
- Realizability of an HMSC is **undecidable** [AEY00]

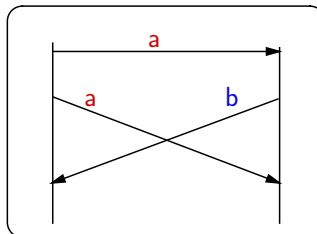
**Note** Realizability is decidable with non-FIFO, bounded CFMs [Mo02]

# Standard CFM: implied scenarios

Standard CFM of

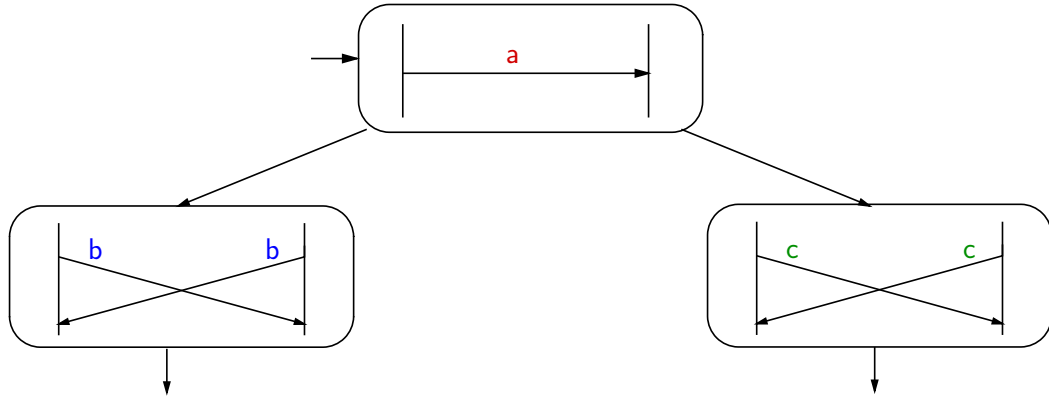


produces a new (undesired) MSC:

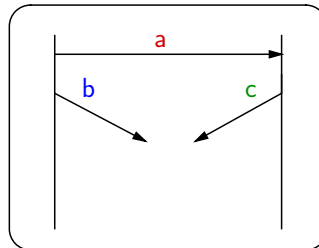


# Standard CFM: deadlocks

Standard CFM of



produces a deadlock



# Weak vs. safe realizability [AEY01]

- An HMSC is (weakly) realizable if it is realized by the standard CFM.
- An HMSC is safely realizable if it is realized by the standard CFM, and if this implementation has no deadlocks.

## Safe realizability

- ▷ is undecidable for general HMSCs [Loh02].
- ▷ is EXPSPACE-complete for bounded HMSCs [AEY01, Loh02]

# Model-checking problems

- System to model-check: HMSC **S**
- Property to verify: given by another HMSC **P**
  - **Positive** model-checking:  $\text{Executions}(\mathbf{S}) \subseteq \text{Executions}(\mathbf{P})?$
  - **Negative** model-checking:  $\text{Executions}(\mathbf{S}) \cap \text{Executions}(\mathbf{P}) = \emptyset?$

# Model-checking problems

- System to model-check: HMSC **S**
- Property to verify: given by another HMSC **P**
  - **Positive** model-checking:  $\text{Executions}(\mathbf{S}) \subseteq \text{Executions}(\mathbf{P})?$
  - **Negative** model-checking:  $\text{Executions}(\mathbf{S}) \cap \text{Executions}(\mathbf{P}) = \emptyset?$

In general, both problems are **undecidable**  
(as many problems on **executions**)

# Outline of the talk

- High-level Message sequence charts

Two problems...

- Realizability of HMSCs languages by CFMs
- Model checking of HMSCs against HMSCs

... answered for 3 infinite-state HMSC subclasses

- Globally-cooperative HMSCs
- Locally-cooperative HMSCs
- Local-choice HMSCs

# Ideas

- Find reasonable **syntactical restrictions** on HMSCs for which model-checking and realizability are decidable/efficient.
- For the model-checking, capture all linearizations of the HMSCs by a **regular set of representatives**.
- For the realizability problem, allow implementation messages to carry **extra data**. (Extra messages not allowed.) Thus, a process may transmit information on its current state, e.g:
  - a vision of its **past**
  - a anticipation of its **future**

For a given HMSC, its extended implementation should use a finite amount of memory

# Cooperative HMSCs

An MSC  $M$  is **atomic** if  $M = AB$  implies  $M = A$  or  $M = B$ .

An MSC  $M$  is **linked** if  $M = AB$  and  $A \parallel B$  imply  $M = A$  or  $M = B$ .

Assumption: nodes of HMSCs we consider are labeled by **linked** MSCs

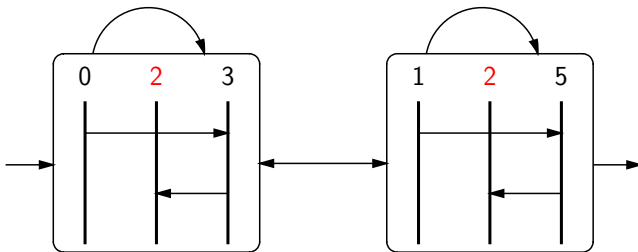
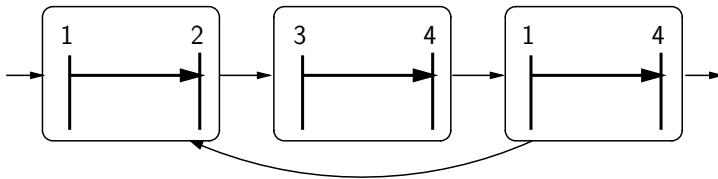
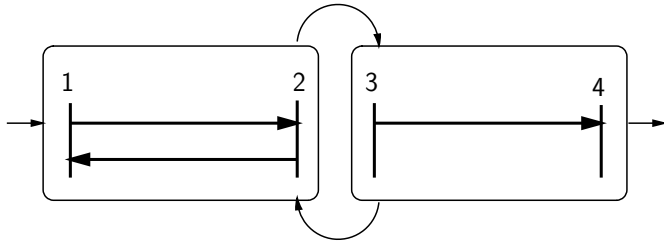
An HMSC  $G = \langle V, R, v^0, v^f, \lambda \rangle$  is

- **Locally cooperative** if for all edge  $v \rightarrow w$ ,  $\lambda(v)\lambda(w)$  is linked.
- **Globally-cooperative** if all loops are labeled by linked MSCs

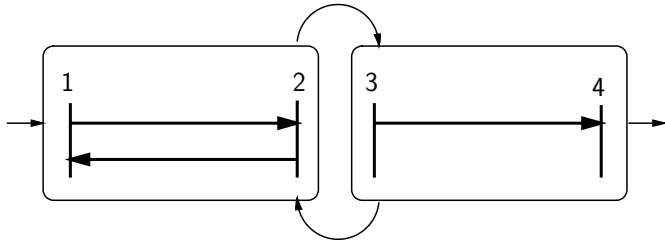
Testing whether an HMSC is

- globally-cooperative is co-NP complete [MP99]
- locally-cooperative is polynomial

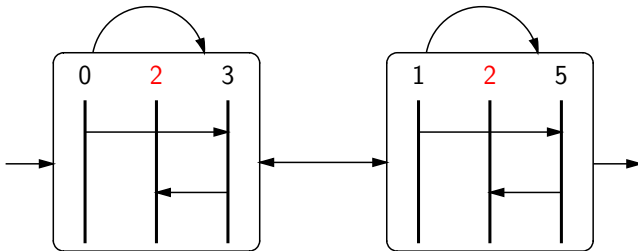
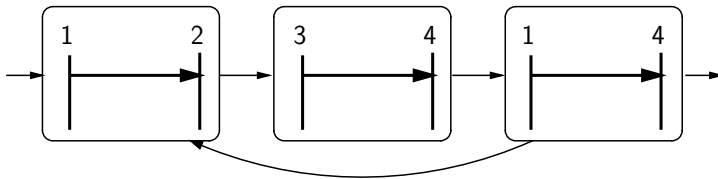
# Cooperative HMSCs: examples



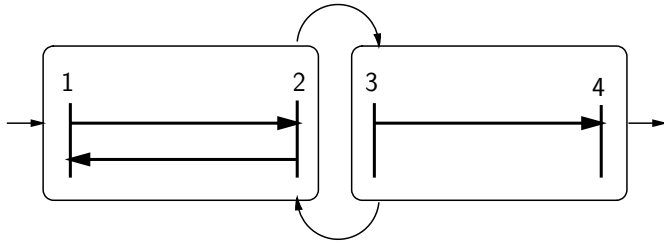
# Cooperative HMSCs: examples



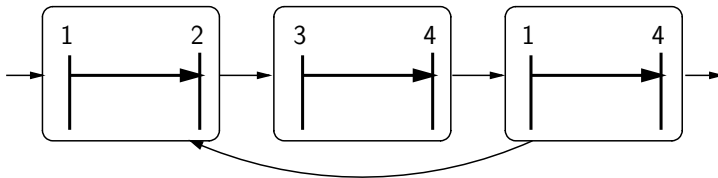
not cooperative



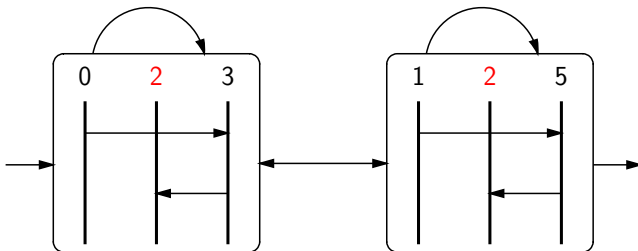
# Cooperative HMSCs: examples



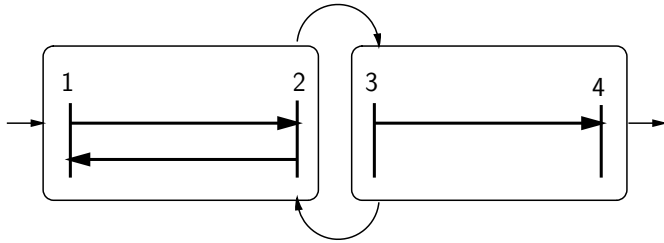
not cooperative



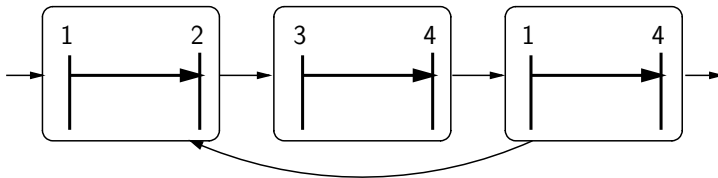
not locally cooperative  
globally cooperative



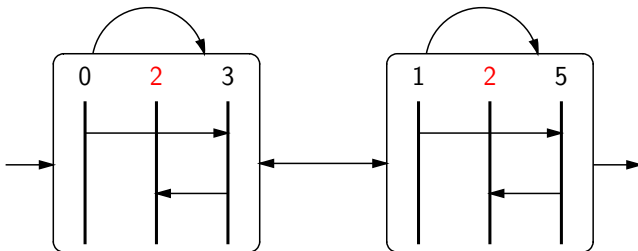
# Cooperative HMSCs: examples



not cooperative



not locally cooperative  
globally cooperative



locally cooperative

# Local-choice HMSCs [HJ00]

**Choice node:** node having at least 2 successors

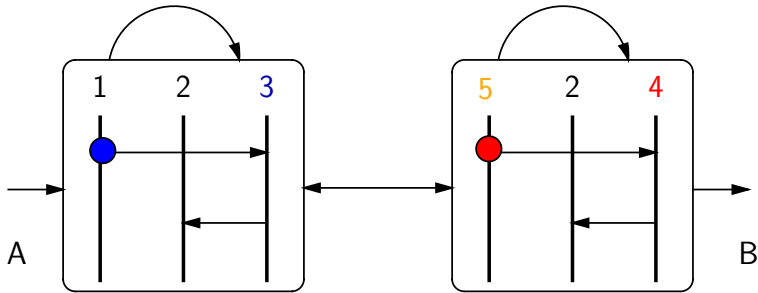
Local-choice HMSC: each choice is controlled by a **single** process

$G = \langle V, R, v^0, v^f, \lambda \rangle$  is **local-choice** iff

1. For each choice node  $v$ , there is a process  $root(v)$  such that every path  $w_1 w_2 \dots$ , starting in a node  $w_1$  successor of  $v$ , has a **unique** minimal event located on  $root(v)$ .
2. Every path starting in  $v^0$  has a **unique** minimal event

Local-choice and locally-cooperative not comparable.

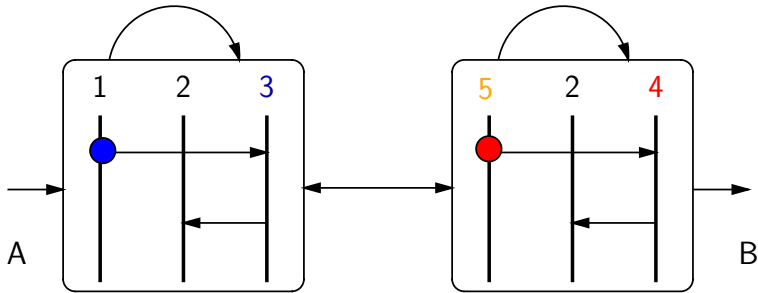
# Local-choice HMSCs



is not local-choice.

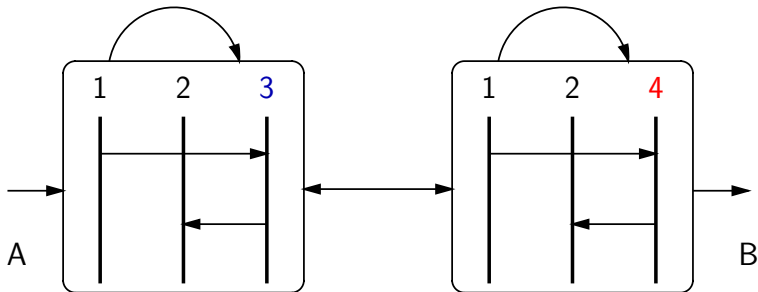
● and ● are minimal on AB

# Local-choice HMSCs



is not local-choice.

● and ● are minimal on AB



is local-choice.

root(A) and root(B) is 1

# Results: Realizability

# Extended implementation

Via extra data, processes are able to

- transmit a bounded vision of their past
- transmit **anticipations** (the approach we have chosen)

HMSC  $G \rightarrow$  Addition of extra data in messages:  $\tilde{G}$

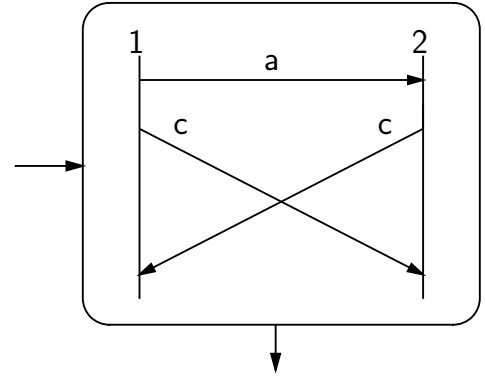
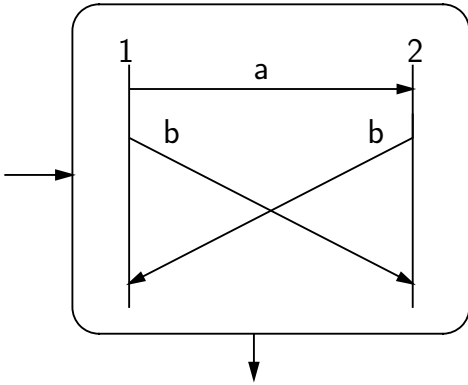
$\rightarrow \tilde{C}$ : standard CFM of  $\tilde{G}$

$\rightarrow$  we may have  $L(\tilde{C}) = L(\tilde{G})$  even if  $L(C) \neq L(G)$

$\rightarrow$  forget extra data for simulation purpose

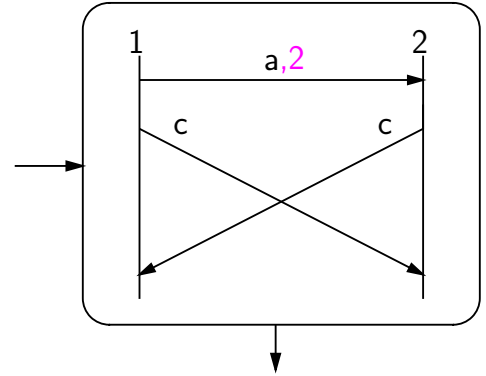
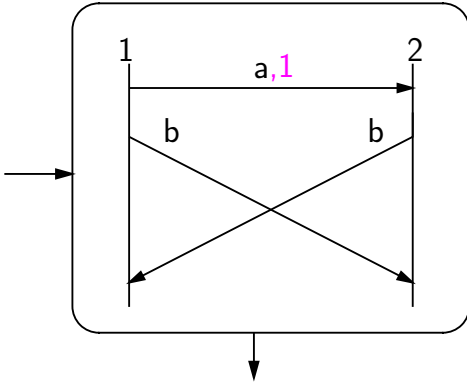
Non implementable languages may become implementable.

# Extended implementation: idea



Not deadlock-free implementable

# Extended implementation: idea



Not deadlock-free implementable

Becomes deadlock-free implementable with **extra data**

# Extended implementation: results

## Theorem

1. Locally-cooperative HMSCs are **always** realizable with renaming

# Extended implementation: results

## Theorem

1. Locally-cooperative HMSCs are **always** realizable with renaming
2. Local-choice HMSCs are also **always** realizable with renaming, and
  - deadlock-free implementation,
  - implementation of linear size.

# Extended implementation: results

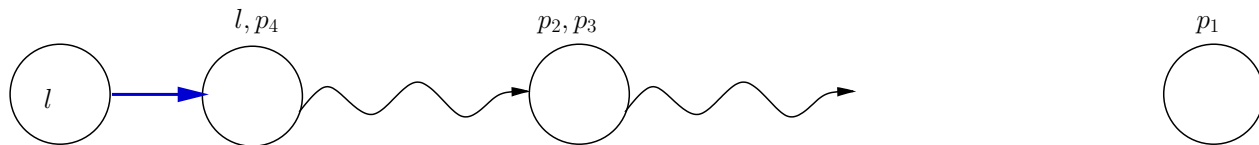
## Theorem

1. Locally-cooperative HMSCs are **always** realizable with renaming
2. Local-choice HMSCs are also **always** realizable with renaming, and
  - deadlock-free implementation,
  - implementation of linear size.
3. Globally-cooperative HMSCs are always realizable using renaming (more difficult)

Implementations of locally-cooperative HMSCs **may have deadlocks**.

# Extended implementation: loc. cooperative case

- each process **guesses** the first nodes in which other processes reappear,



- local cooperation makes possible to **transmit** predictions along edges.
- consistency of all predictions inside a node is ensured (**linked** node),
- system remains consistent: any bad choice is detected and deadlocks
- precomputation of predictions (fixpoint)

# Results: Model-checking

# Model-checking cooperative HMSCs: results

## Theorem

Model-checking  $\text{negative}(\cap)$ / $\text{positive}(\sqsubseteq)$  properties of

- a. locally cooperative  $\text{atom-labeled}$  HMSCs is  $\text{NLOG-complete}$ / $\text{PSPACE-complete}$ .
- b. locally-cooperative HMSCs is  $\text{PSPACE-compl}$ / $[\text{PSPACE-EXPSPACE}]$
- c. globally-cooperative HMSCs is  $\text{PSPACE-compl}$ / $\text{EXPSPACE-complete}$

# Model-checking cooperative HMSCs: upper bounds

$$\text{Lin}^a(G) = \text{Lin}(G) \cap [\text{Lin}(\text{Atom}(G))]^*$$

- $\text{Lin}^a(G)$  is a set of representatives of  $\text{Lin}(G)$ .
- $\implies \text{Lin}^a(G) \cap \text{Lin}^a(H) \neq \emptyset$  iff  $L(G) \cap L(H) \neq \emptyset$
- Use uniqueness of the decomposition in atoms up to commutation.  
For locally-cooperative **atom-labeled** HMSCs, no commutation between MSCs labeling adjacent transitions

# Model-checking cooperative HMSCs: upper bounds

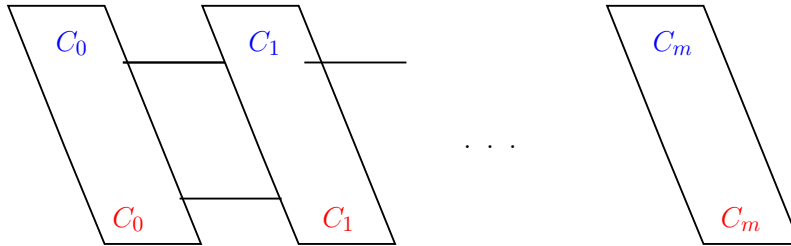
$$\text{Lin}^a(G) = \text{Lin}(G) \cap [\text{Lin}(\text{Atom}(G))]^*$$

- $\text{Lin}^a(G)$  is a set of representatives of  $\text{Lin}(G)$ .
- $\implies \text{Lin}^a(G) \cap \text{Lin}^a(H) \neq \emptyset$  iff  $L(G) \cap L(H) \neq \emptyset$
- Use uniqueness of the decomposition in atoms up to commutation.  
For locally-cooperative **atom-labeled** HMSCs, no commutation between MSCs labeling adjacent transitions
- $\text{Lin}^a(G)$  is regular (due to the cooperative condition) and one can construct an automaton for it (exponential size) [MP99]
- Use complexity of intersection and inclusion on automata.
- Using the same idea, any CFM-implementable class has decidable MC.

# Model-checking cooperative HMSCs: lower bounds

Intersection for loc. cooperative: simulation of a PSPACE TM.

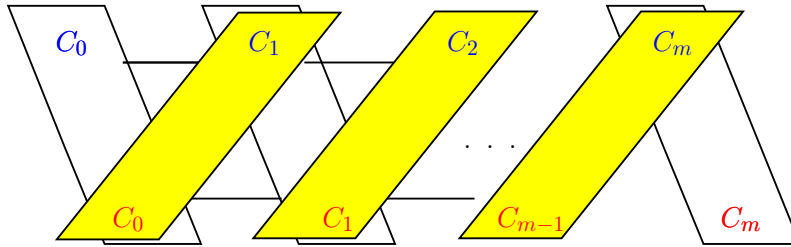
Computation  $C_0 \vdash C_1 \vdash \dots \vdash C_m$ ,  $|C_i| = k = O(|w|^n)$



# Model-checking cooperative HMSCs: lower bounds

Intersection for loc. cooperative: simulation of a PSPACE TM.

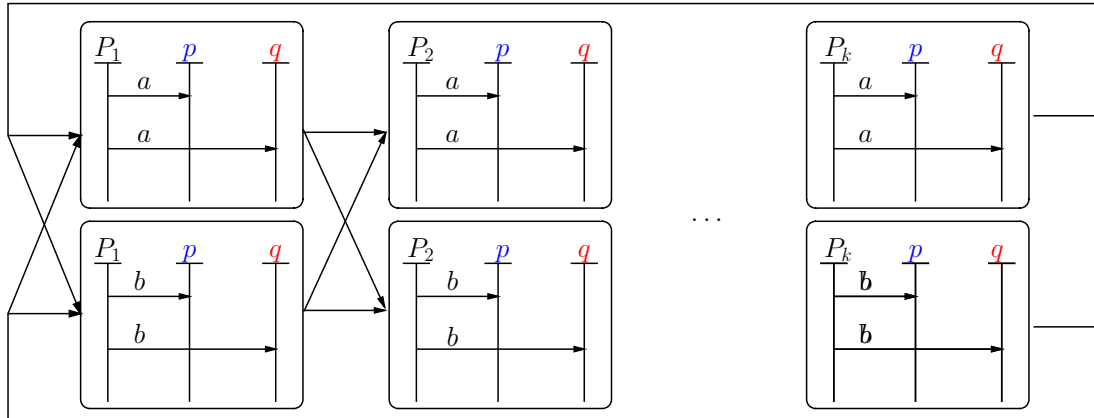
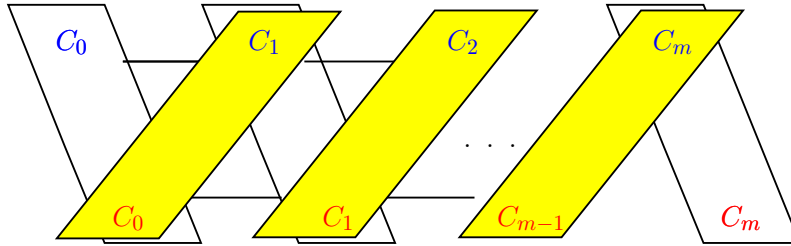
Computation  $C_0 \vdash C_1 \vdash \dots \vdash C_m$ ,  $|C_i| = k = O(|w|^n)$



# Model-checking cooperative HMSCs: lower bounds

Intersection for loc. cooperative: simulation of a PSPACE TM.

Computation  $C_0 \vdash C_1 \vdash \dots \vdash C_m$ ,  $|C_i| = k = O(|w|^n)$

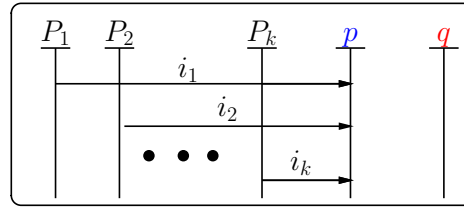


Enforces reading two identical tape symbols sequences (1 blue, 1 red)

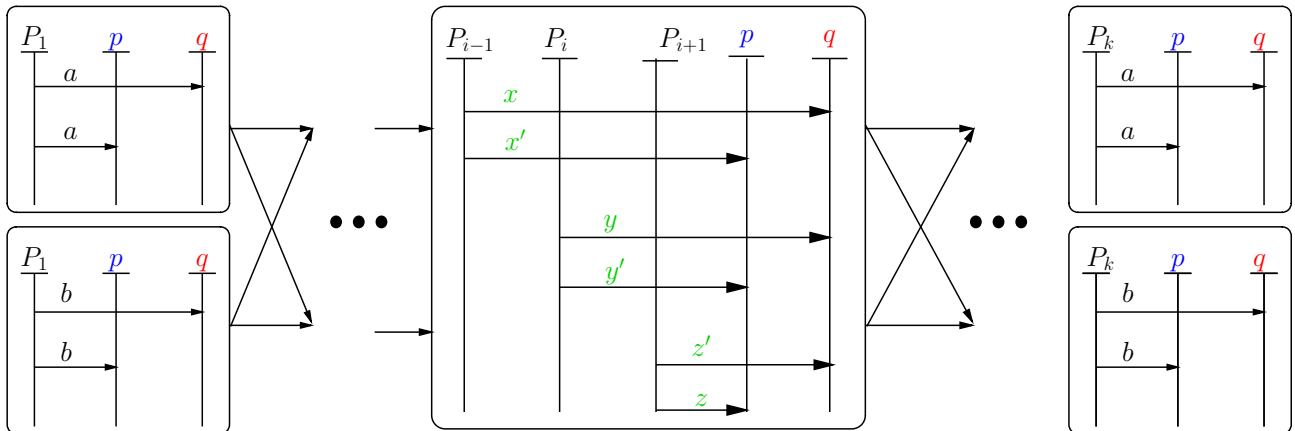
# Enforcing a successful computation $C_i \vdash C_{i+1}$

The language of the yellow HMSC is **Init** **Trans**\* **Accept**.

Let  $C_0 = i_1 \cdots i_k$ . Then Init =



**Trans** non-deterministically makes a transition  $xyz \rightarrow x'y'z'$  at a position



# Local HMSCs and non-decomposable HMSCs

An HMSC is **local** iff

- it is locally-cooperative,
- it is local-choice,
- all of its nodes are triangles, ie of the form  $e\uparrow$ .

A  **$p$ -decomposition** of an MSC  $M$  is a factorization  $M = Ne\uparrow$ , where

- $e\uparrow$  is an MSC,
- $e$  is located on  $p$ .

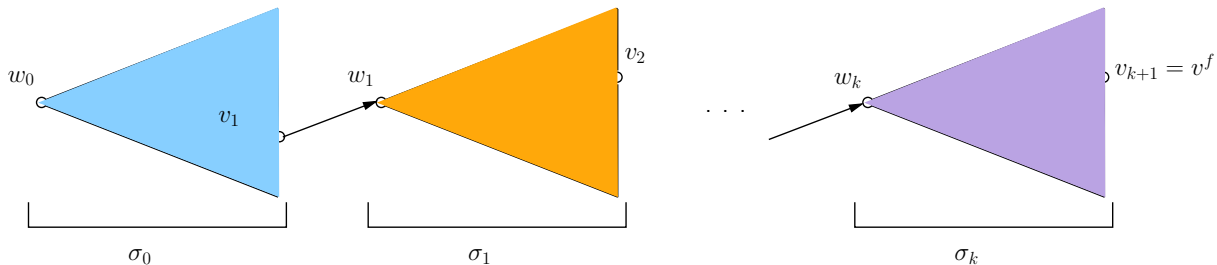
A local HMSC is **non-decomposable** if no node  $v$  is  $\text{root}(v)$  decomposable

# From local-choice to local HMSCs

## Fact

For every local-choice HMSC one can construct an equivalent locally-cooperative + local-choice HMSC of quadratic size.

Idea Factorize a path in the HMSC between consecutive choice nodes

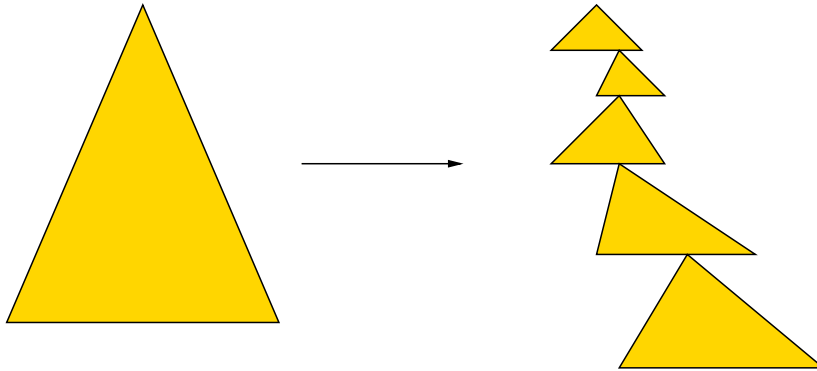


Nodes of the new HMSC labeled by “single-choice paths”  $\sigma_i$

## Consequence

MC for local-choice  $\leq_P$  to MC for local-choice+locally-cooperative

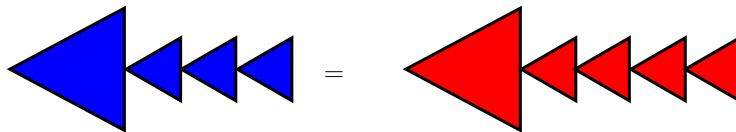
# From local to local+non-decomposable HMSCs



- Bottom-up,
- choosing at each step a minimal triangle
- with respect to the current root

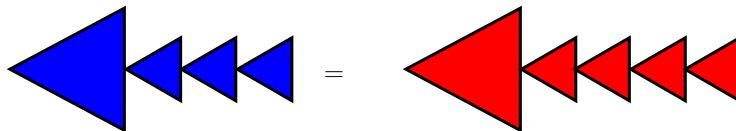
# Model-checking local, non-decomposable HMSCs

$$M_1 M_2 \cdots M_k = N_1 N_2 \cdots N_\ell$$



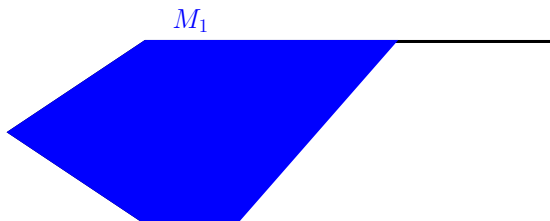
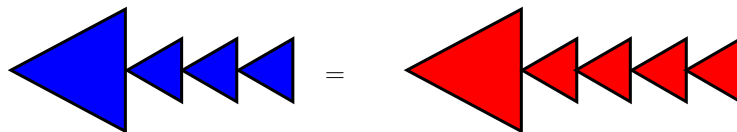
# Model-checking local, non-decomposable HMSCs

$$M_1 M_2 \cdots M_k = N_1 N_2 \cdots N_\ell$$



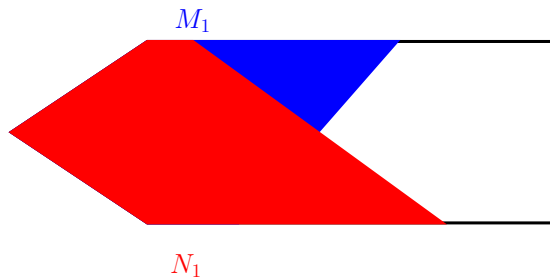
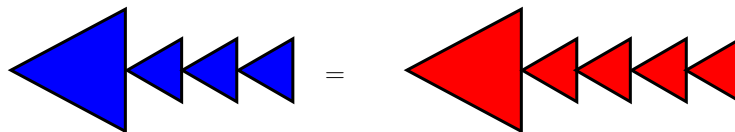
# Model-checking local, non-decomposable HMSCs

$$M_1 M_2 \cdots M_k = N_1 N_2 \cdots N_\ell$$



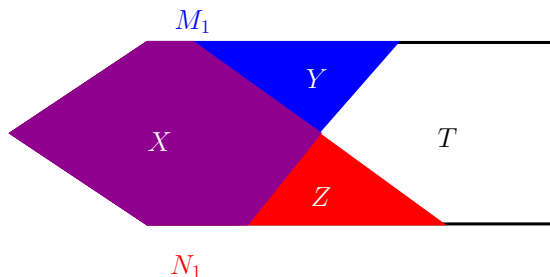
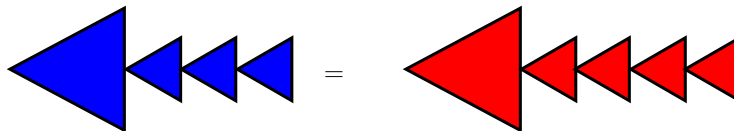
# Model-checking local, non-decomposable HMSCs

$$M_1 M_2 \cdots M_k = N_1 N_2 \cdots N_\ell$$



# Model-checking local, non-decomposable HMSCs

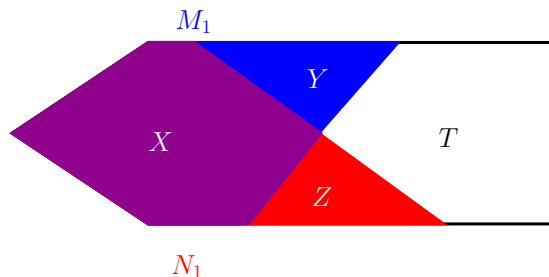
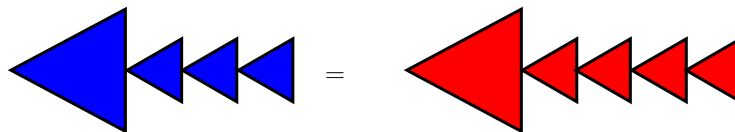
$$M_1 M_2 \cdots M_k = N_1 N_2 \cdots N_\ell$$



$$X = M_1 \cap N_1$$
$$Y \parallel Z$$

# Model-checking local, non-decomposable HMSCs

$$M_1 M_2 \cdots M_k = N_1 N_2 \cdots N_\ell$$



$$X = M_1 \cap N_1$$
$$Y \parallel Z$$

**Lemma** Both  $Y, Z$  are empty, or  $T$  is empty.

- First case:  $M_1 = N_1 \rightarrow$  easy to deal with
- Second case:  $M_1 = X N_2 \cdots N_\ell$  and  $N_1 = X M_2 \cdots M_k \rightarrow$  possible precomputation of such  $(M_1, N_1)$ .

# Model-checking local-choice HMSCs

**Theorem** Let  $G, H$  be local-choice.

1. Deciding whether  $L(G) \cap L(H) = \emptyset$  can be decided in  $O((|G| + |H|)^2)$ .
2. Deciding whether  $L(G) \subseteq L(H)$  is PSPACE-complete

**Idea** Local-choice HMSC  $\rightarrow$  Local HMSC

$\rightarrow$  Local-non-decomposable HMSC

$\rightarrow$  Model-checking these HMSCs is easy

# Further work

- Find syntactic conditions for **safe** implementability
- ▷ Extended local-choice [GM03]
- Extend results to **recursive** MSCs
- Other **representations** of MSC languages (asynchronous automata)

# Summary (FIFO)

Class $\mathcal{C}$	G.Coop	L.Coop	L.Coop + atom	Loc.Choice
$\in \mathcal{C}?$	co-NP <sup>0</sup>	P	P	P
$L \cap K \neq \emptyset?$	PSPACE	NLOG <sup>1</sup>	NLOG	NLOG
$L \subseteq K?$	EXPSPACE	PSPACE <sup>1</sup>	PSPACE	PSPACE
Implem. weak	Undec <sup>2,3</sup>			
Implem. safe	EXPSPACE <sup>2</sup>	PSPACE <sup>2</sup>		
Implem.+data	Yes	$ G ^{O(\wp)}$	$ G ^{O(\wp)}$	$ G $

0) [MP99] 1) if  $\wp$  constant 2) [Loh02] 3) [AEY]

- Model-checking and realizability can be **decidable** (even efficiently)
- Partial order methods are very helpful

# Next?

# Next?

Green, local, non-decomposable, decorated MSCs.

