Verification of Message Sequence Charts

Madhavan Mukund

Chennai Mathematical Institute

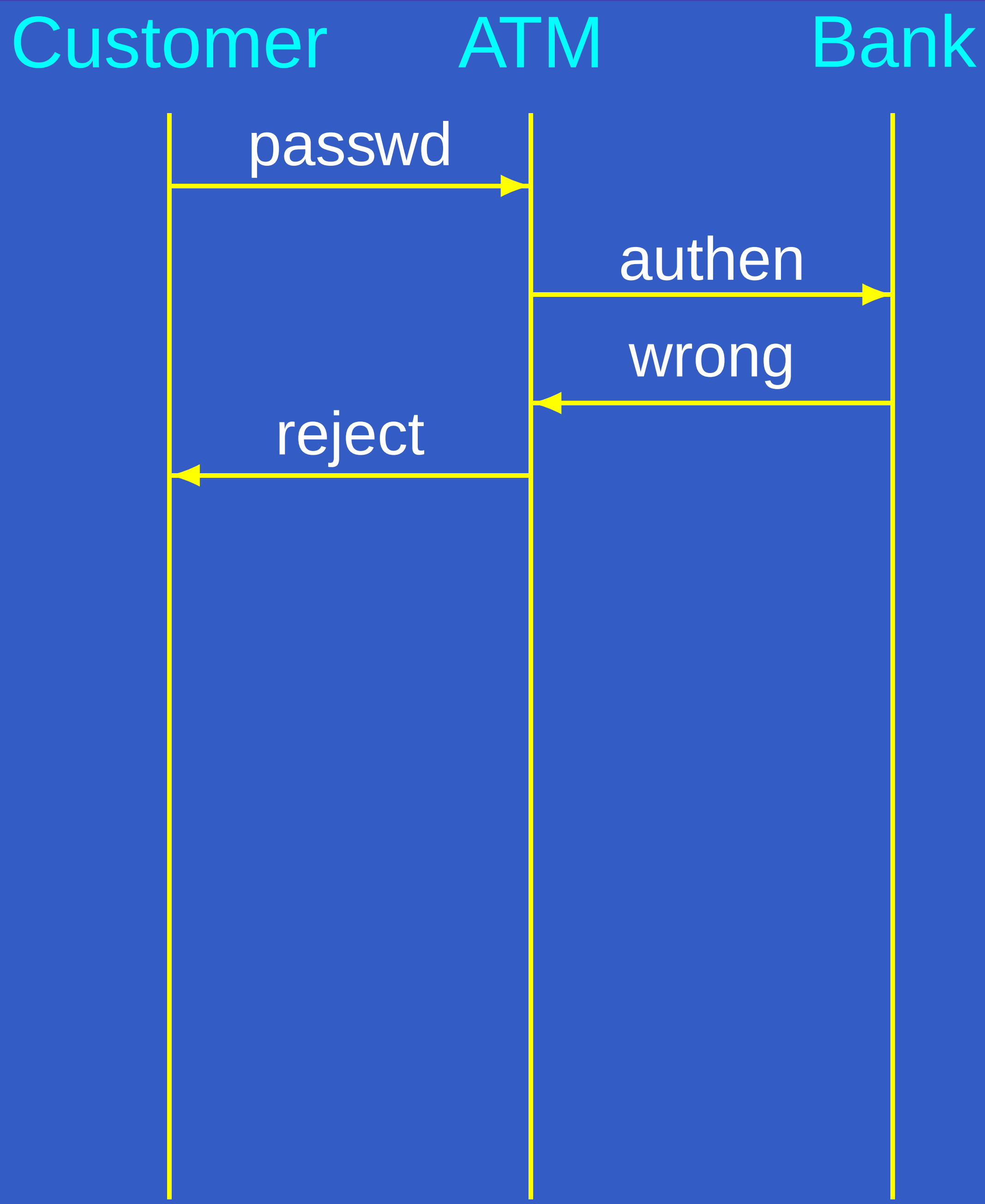
92 G N Chetty Rd, Chennai 600 017, India

`http://www.cmi.ac.in/~madhavan`

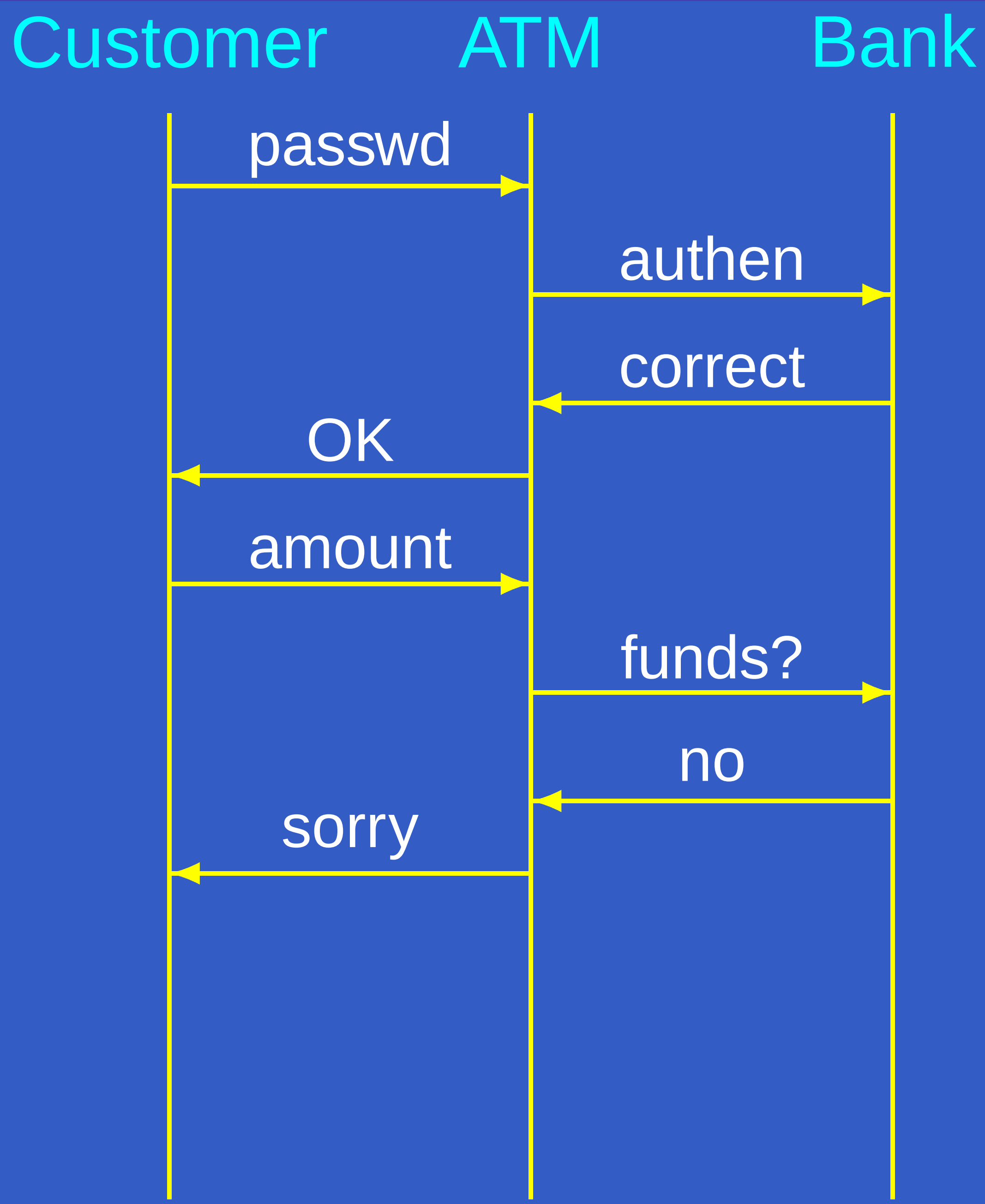
Scenarios

- A scenario describes a pattern of interaction
- Attractive visual formalism
- Telecommunications
  - Message sequence charts (MSC)
  - Messages sent between communicating agents
- UML
  - Sequence diagrams
  - Interaction between objects
    e.g., method invocations etc

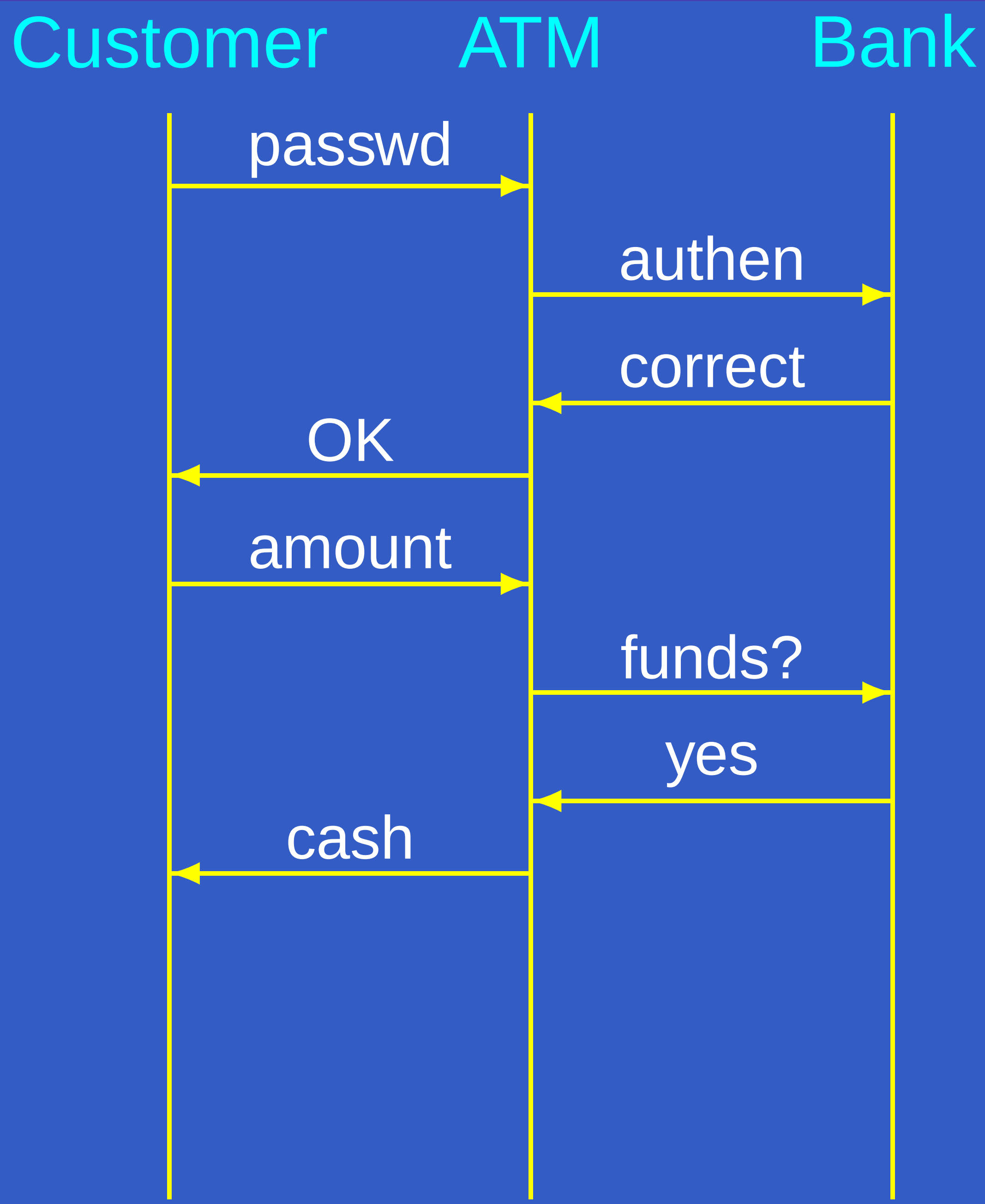
An ATM



An ATM

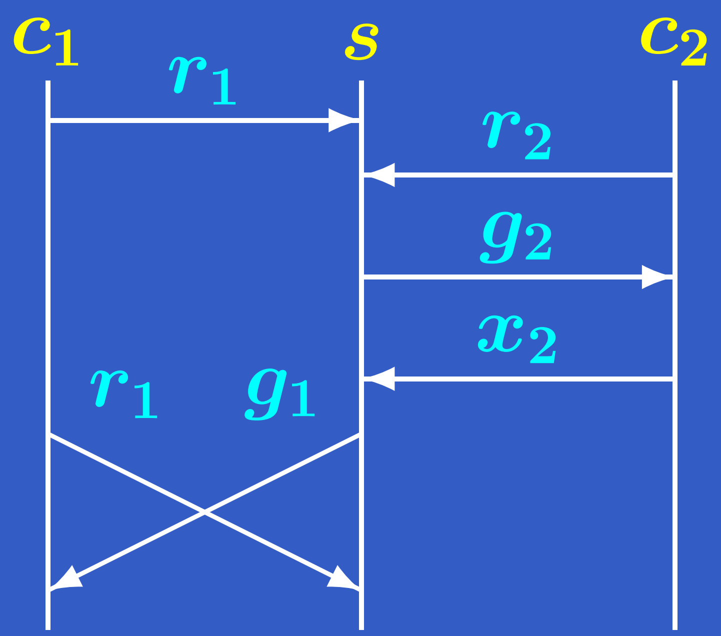

Customer
ATM
Bank
passwd
authen
correct
OK
amount
funds?
no
sorry

An ATM


Customer
ATM
Bank
passwd
authen
correct
OK
amount
funds?
yes
cash

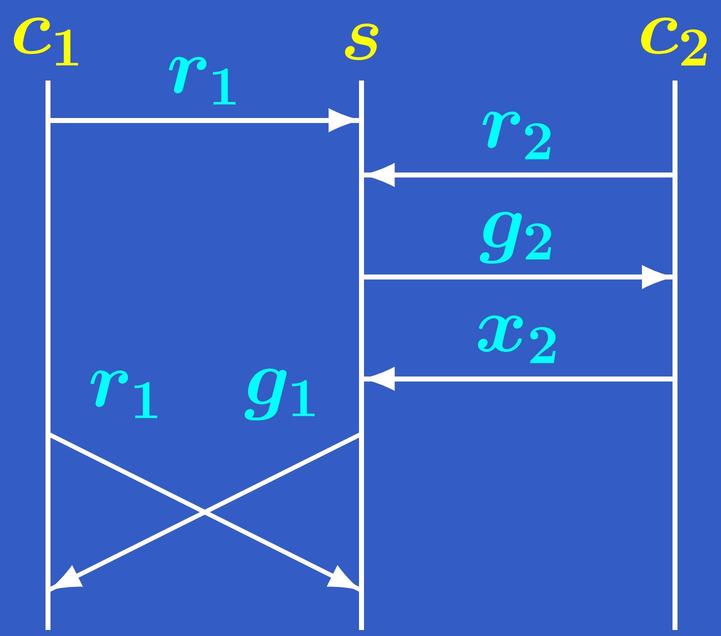
Message sequence charts

Two clients and a server

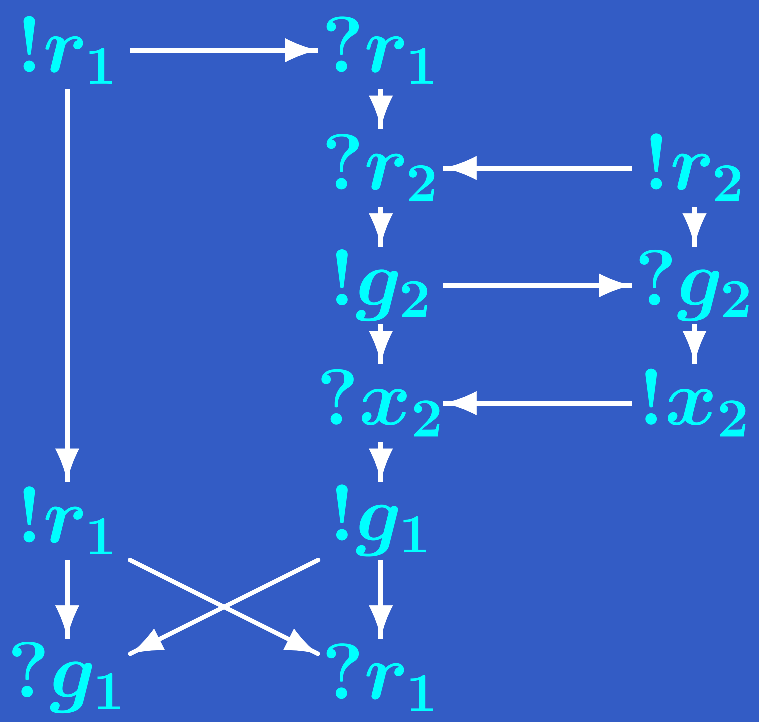


Message sequence charts

Two clients and a server,

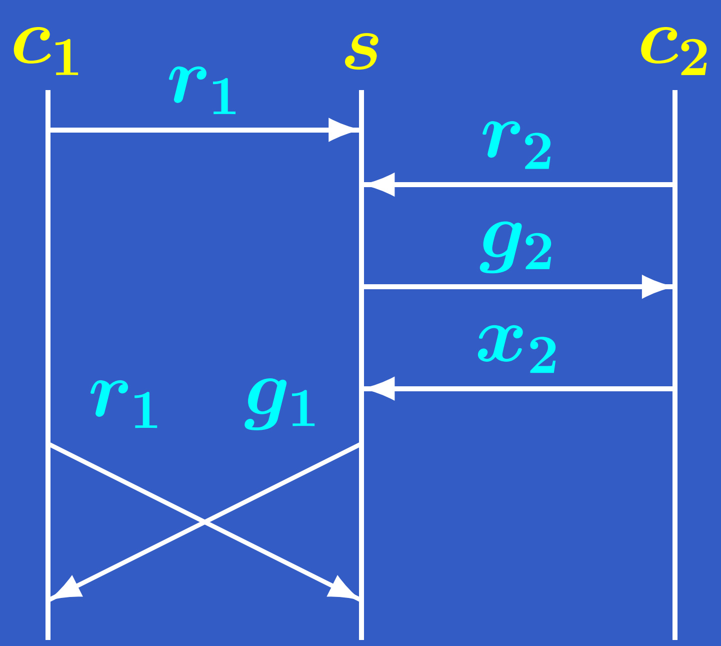


and a partial order representation

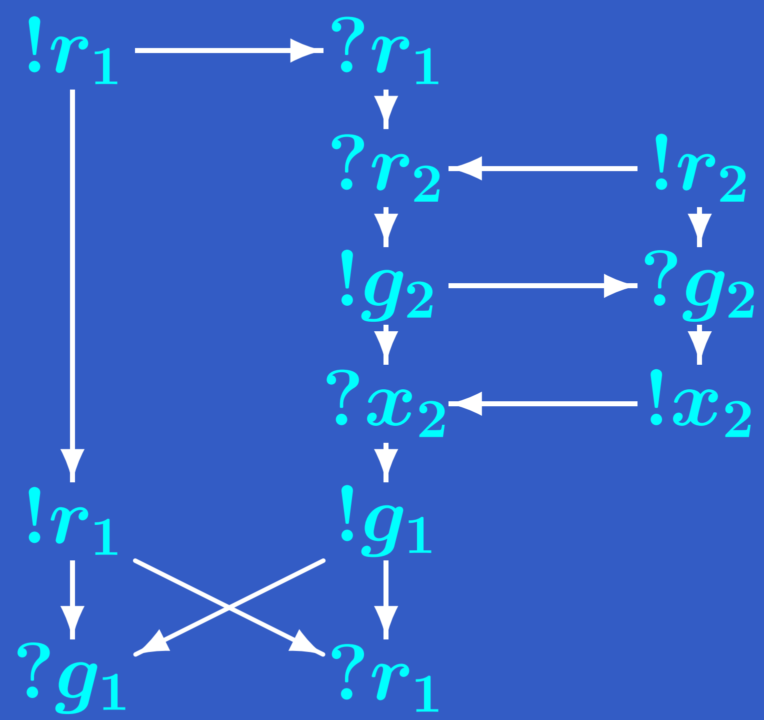


Message sequence charts

Two clients and a server,



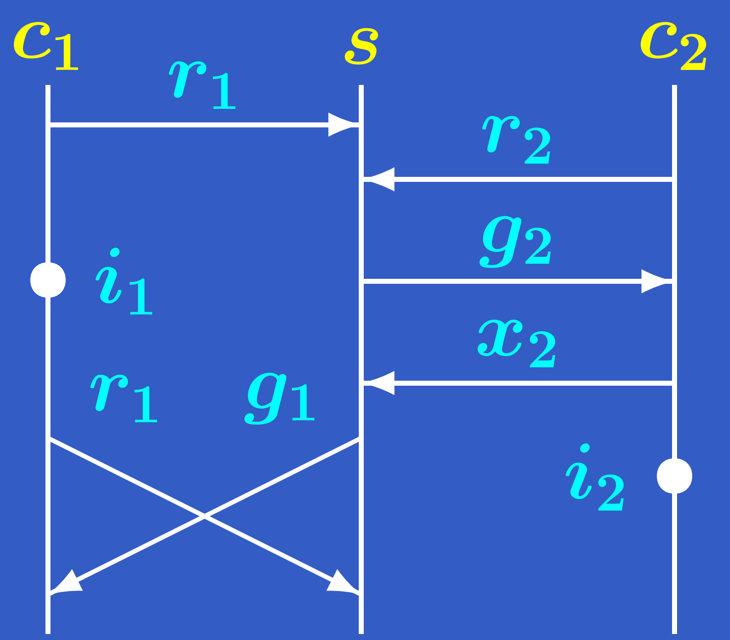
and a partial order representation



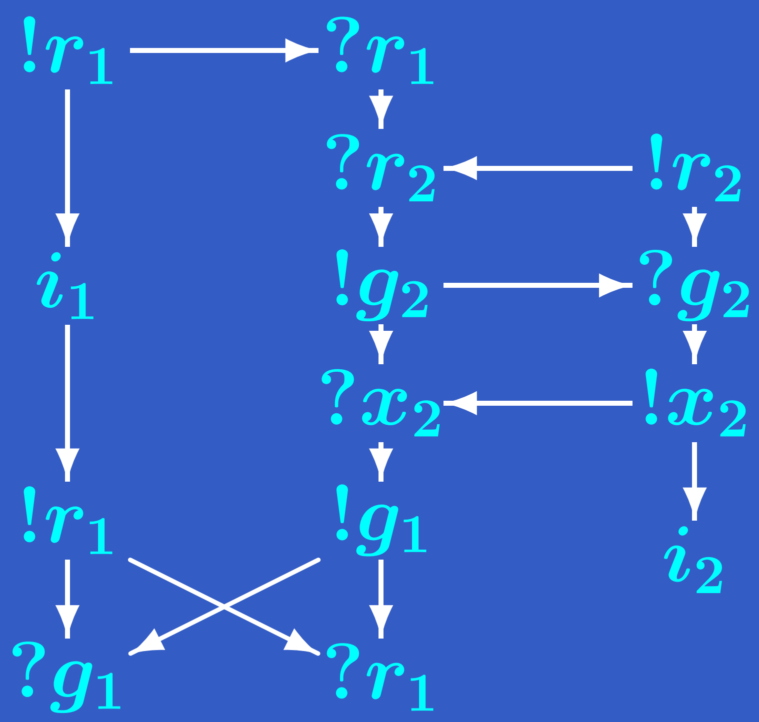
Assume, in general, that channels are fifo

Message sequence charts

Two clients and a server,



and a partial order representation



Assume, in general, that channels are fifo

Can add *internal* events, local to processes

Collections of MSCs

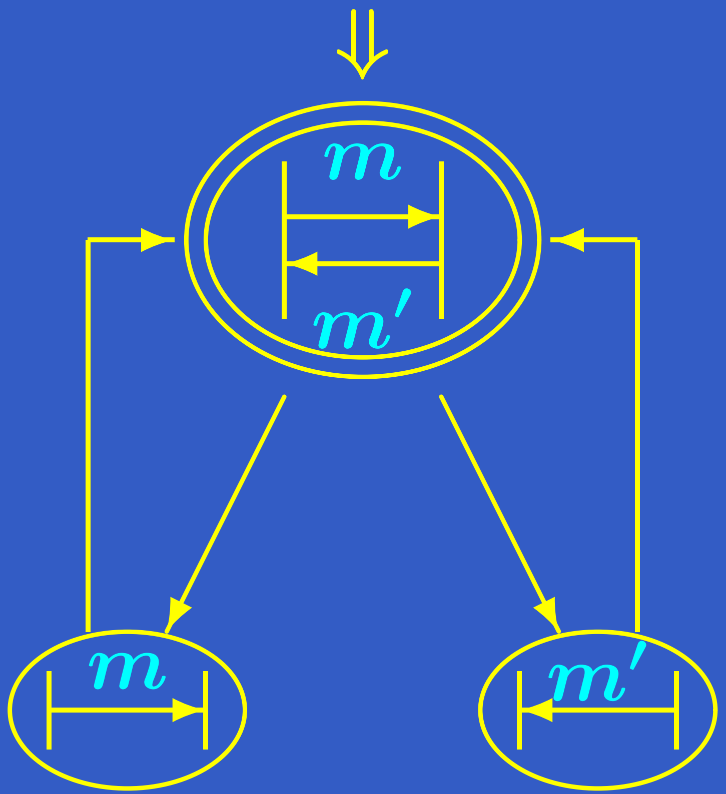
- Often need to specify a collection of scenarios
- Finite collection can be exhaustively enumerated
- Infinite collection needs a generating mechanism

High level MSCs (HMSCs)

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

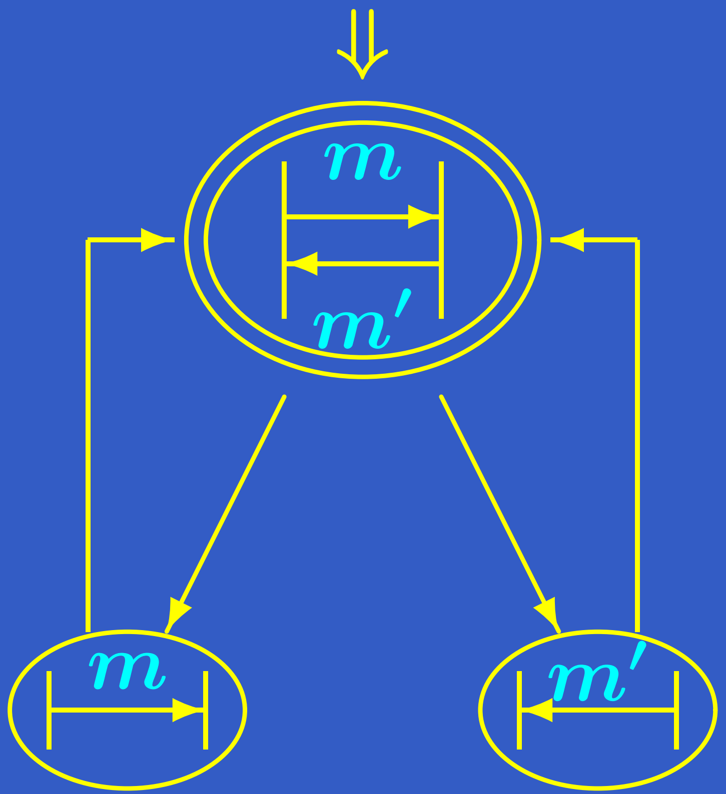
High level MSCs (HMSCs)

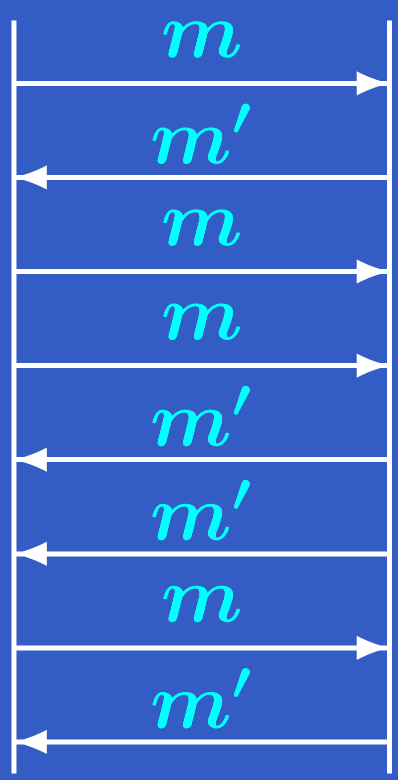
- A finite state automaton
- Each state is labelled by an MSC
- Each (legal) path in the automaton generates an MSC



High level MSCs (HMSCs)

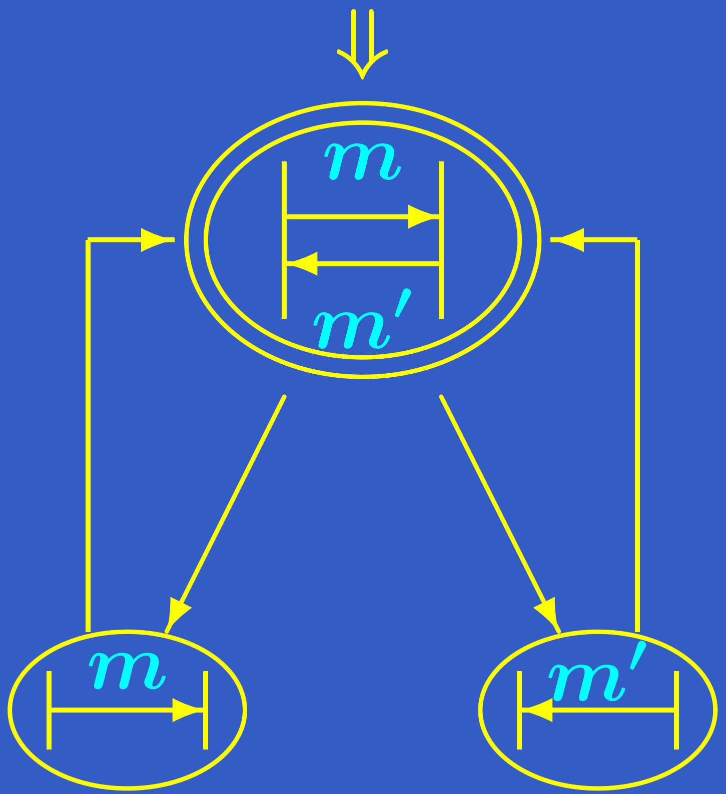
- A finite state automaton
- Each state is labelled by an MSC
- Each (legal) path in the automaton generates an MSC

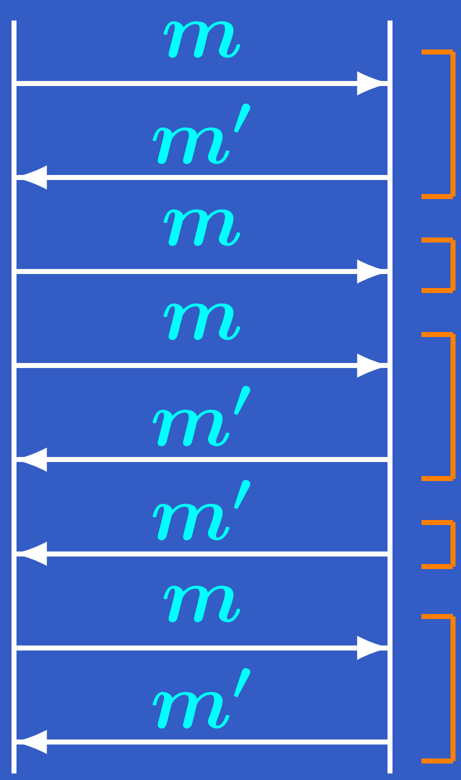




High level MSCs (HMSCs)

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





Concatenation of MSCs

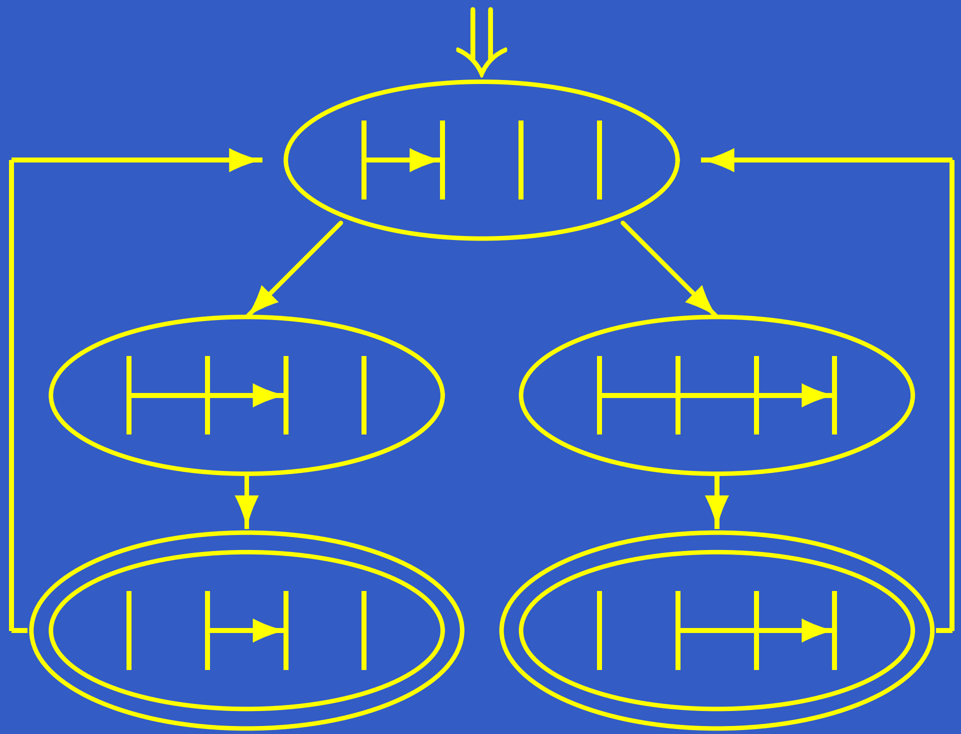
- First MSC finishes before second starts : *synchronous*

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

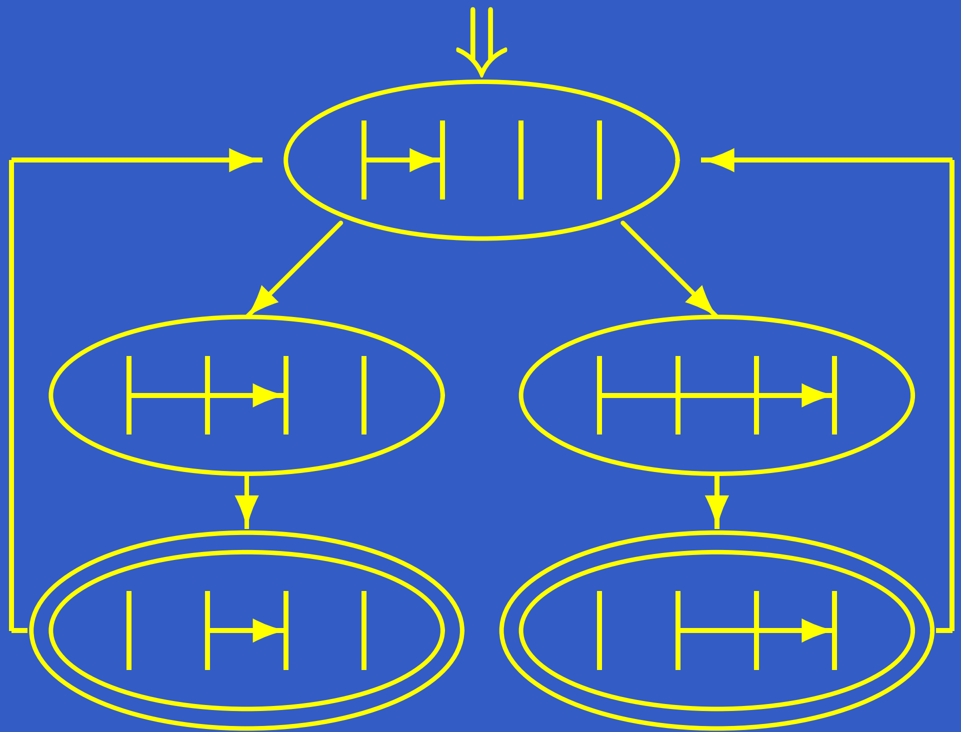
Concatenation of MSCs

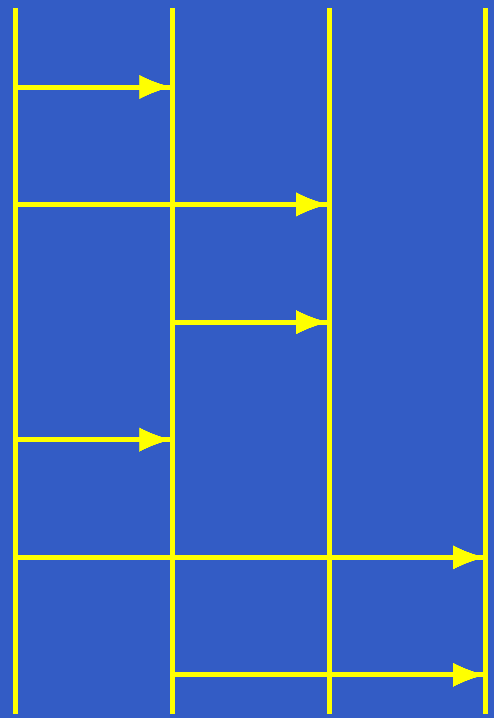
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC



Concatenation of MSCs

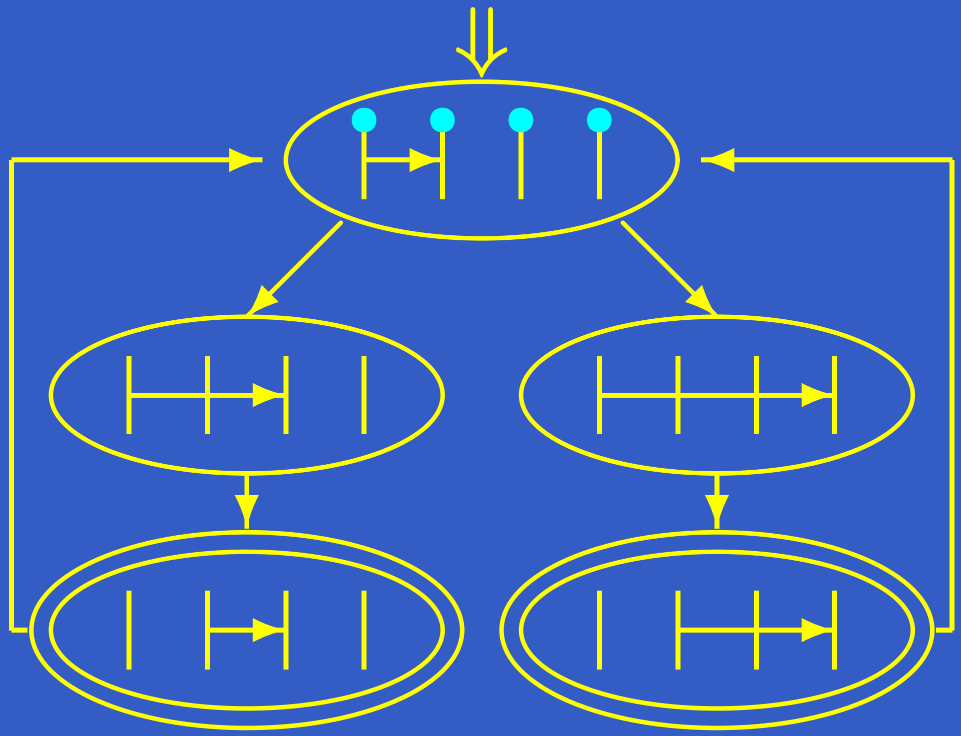
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

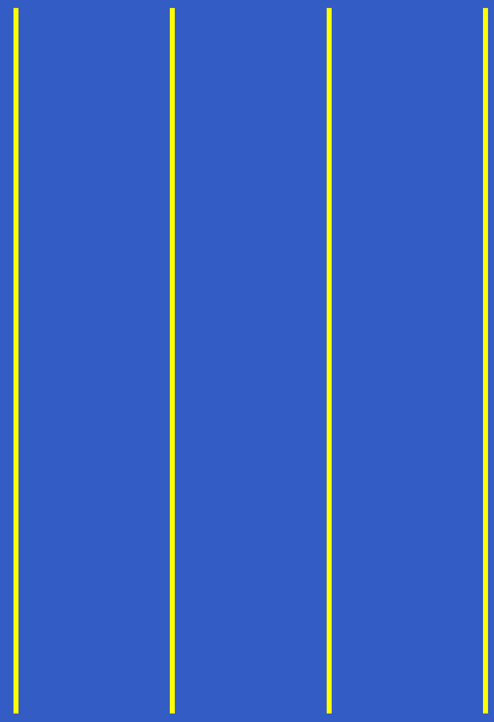




Concatenation of MSCs

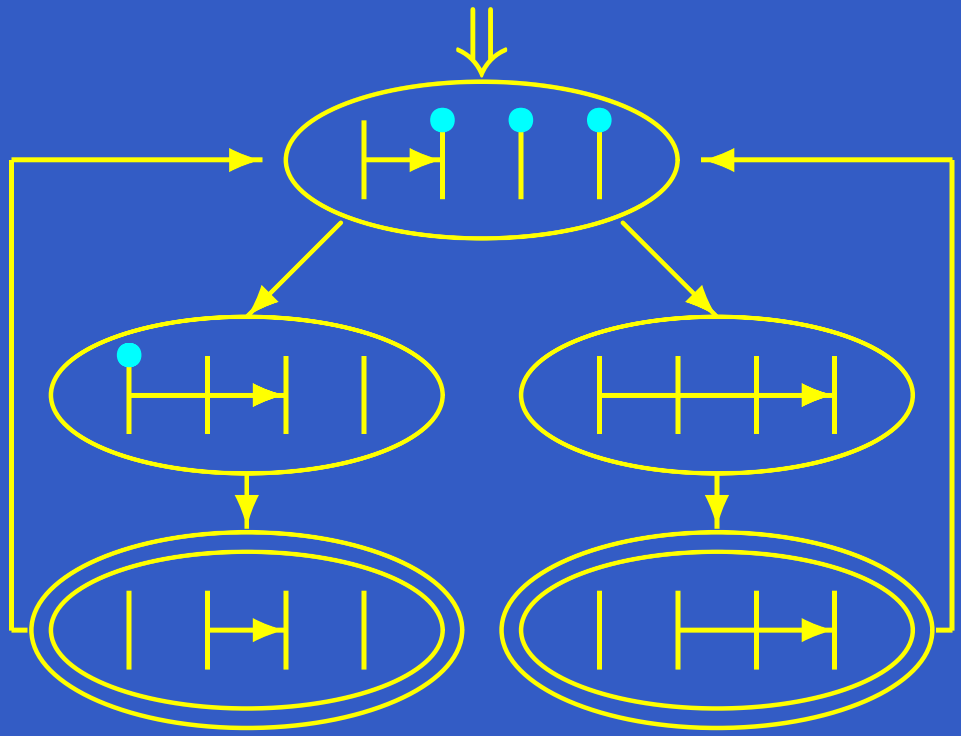
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

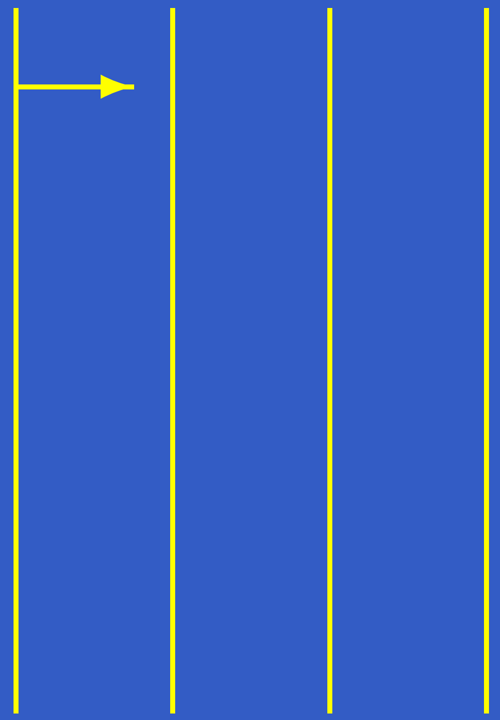




Concatenation of MSCs

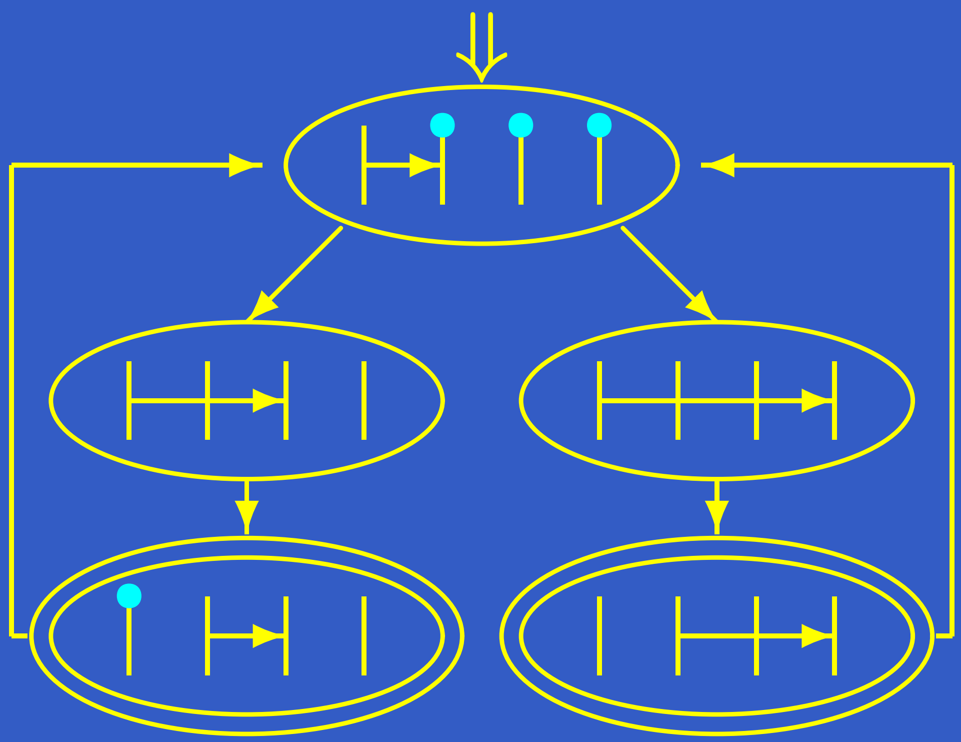
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

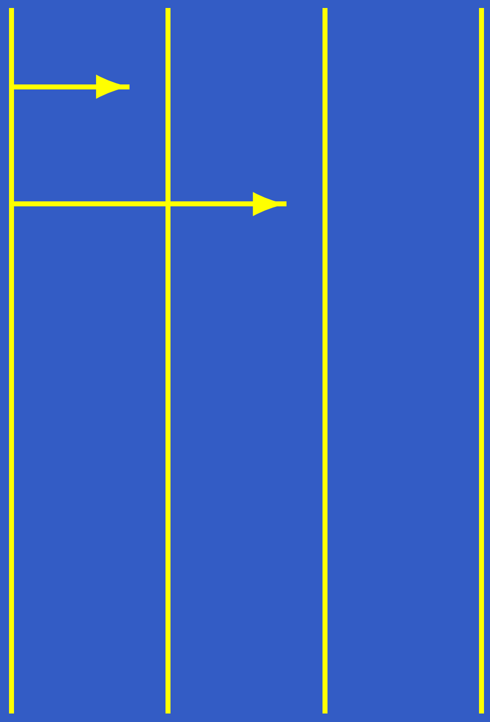




Concatenation of MSCs

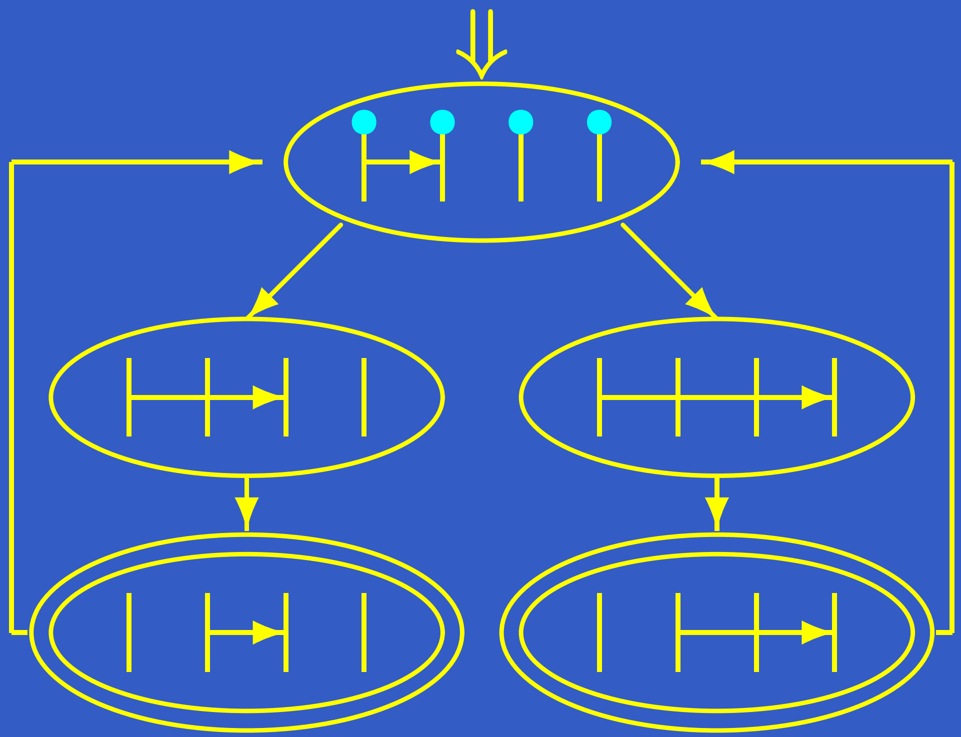
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

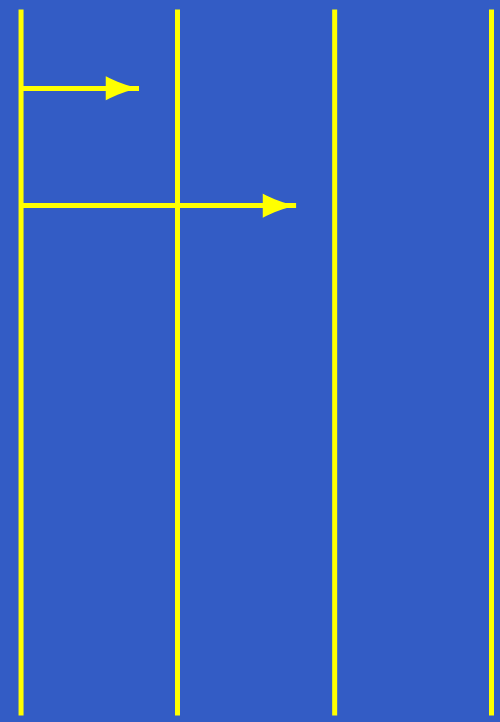




Concatenation of MSCs

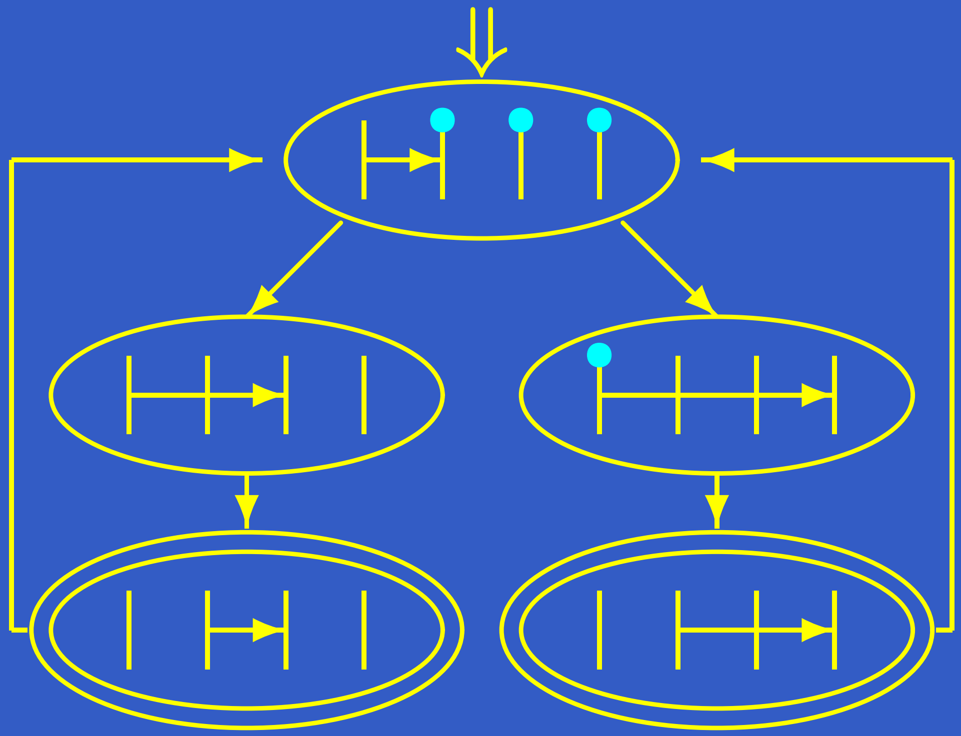
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

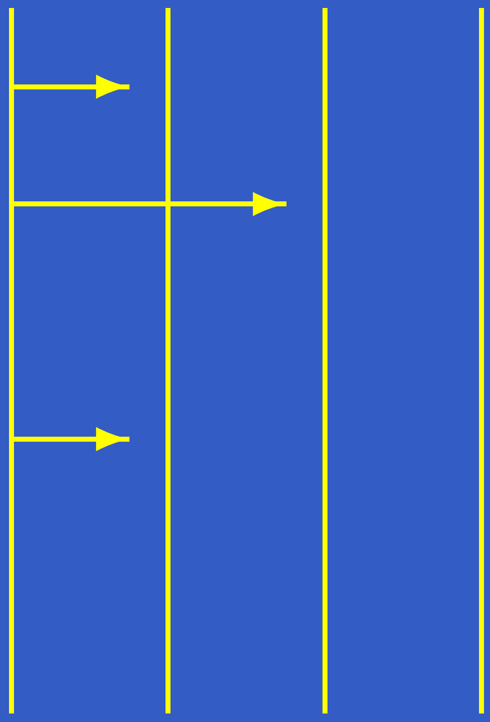




Concatenation of MSCs

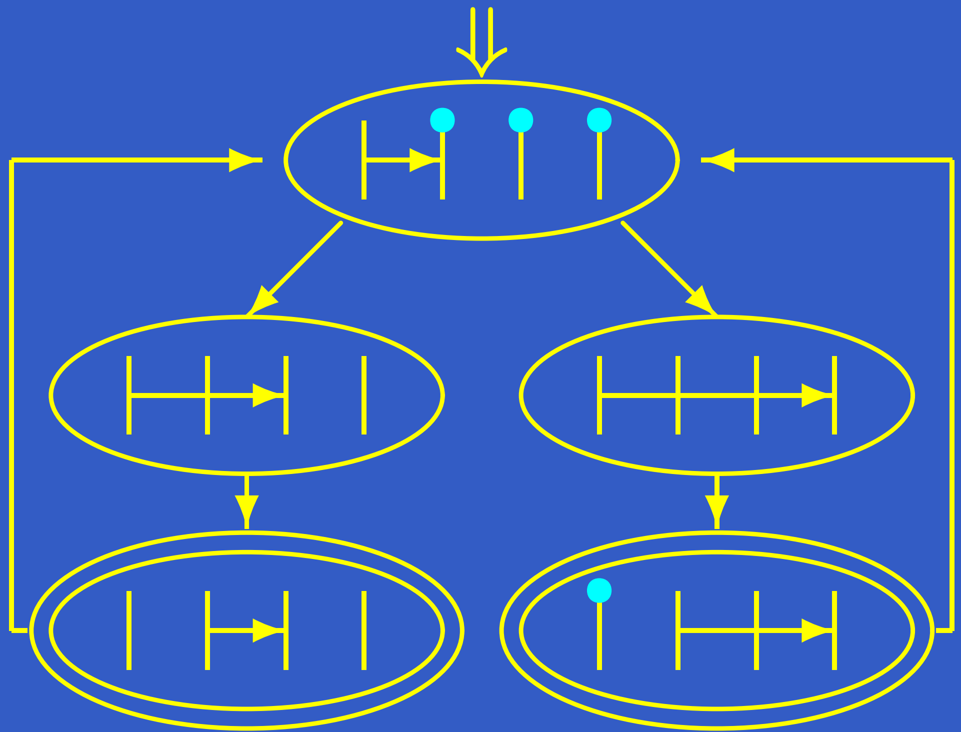
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

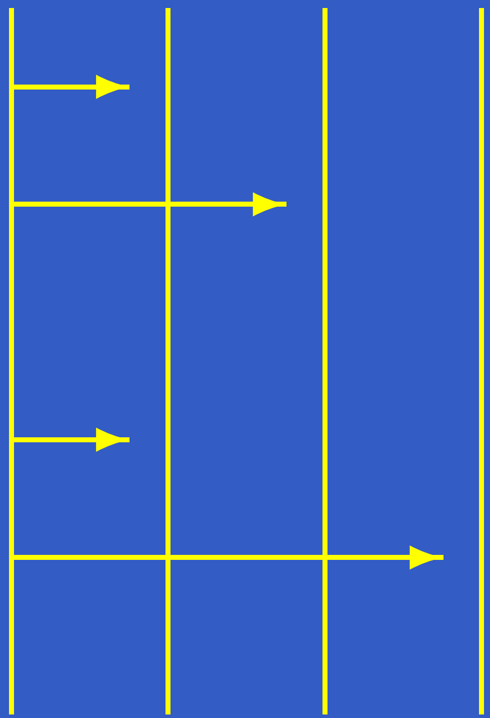




Concatenation of MSCs

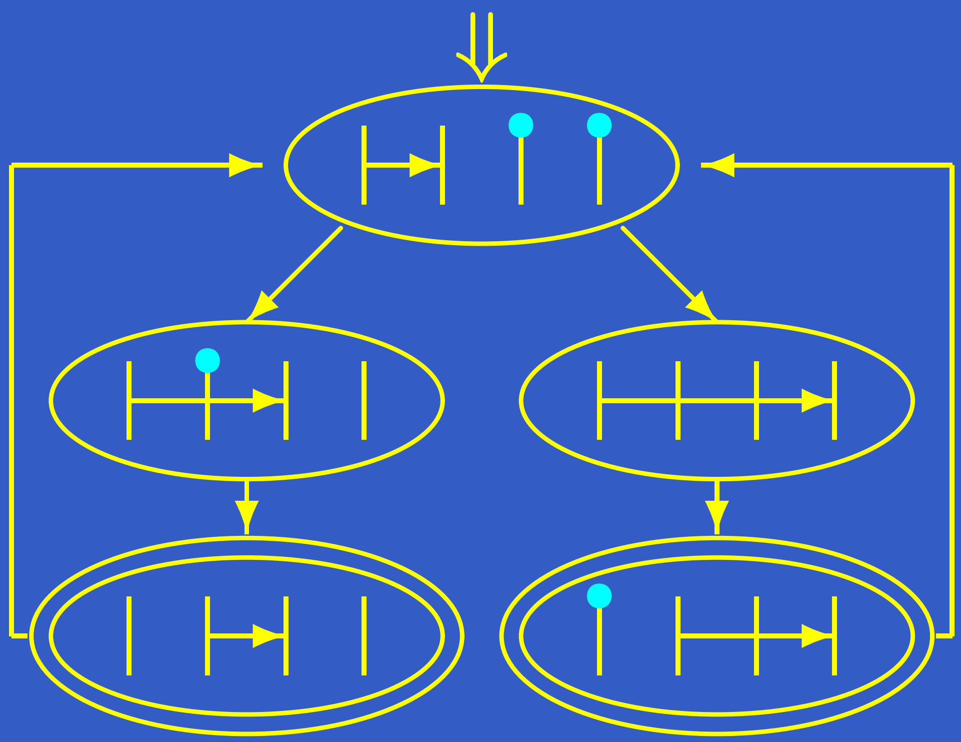
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

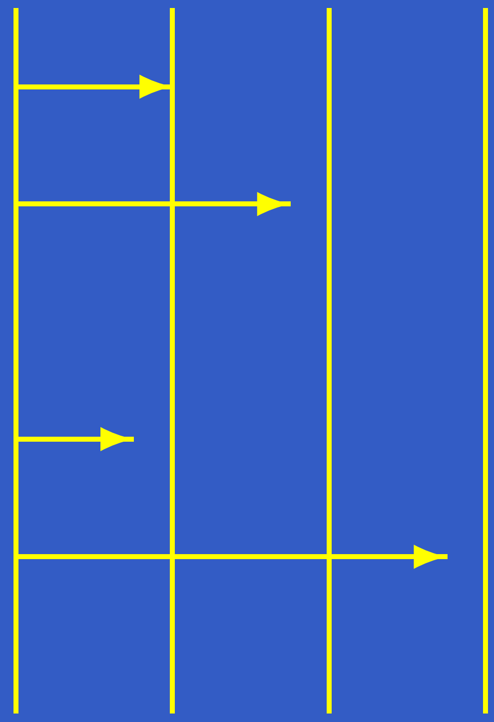




Concatenation of MSCs

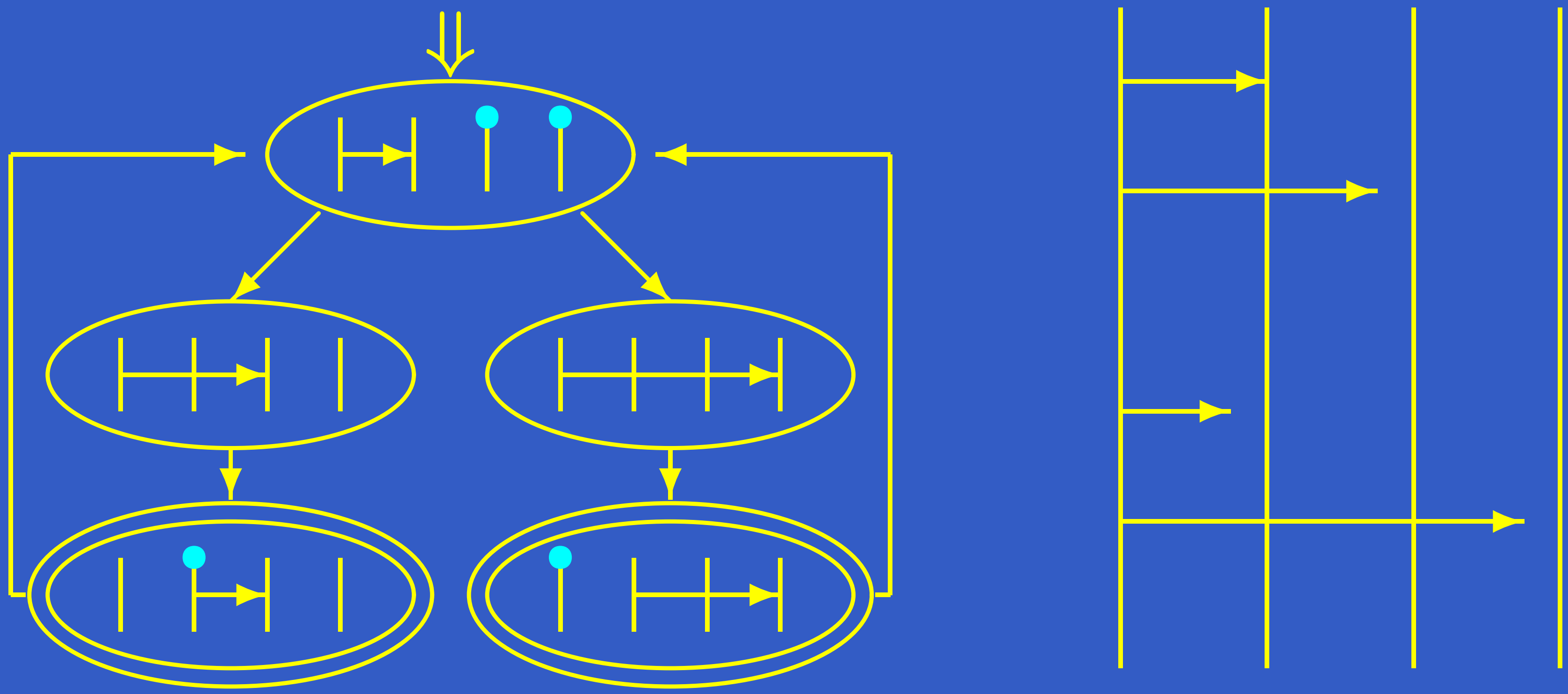
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC





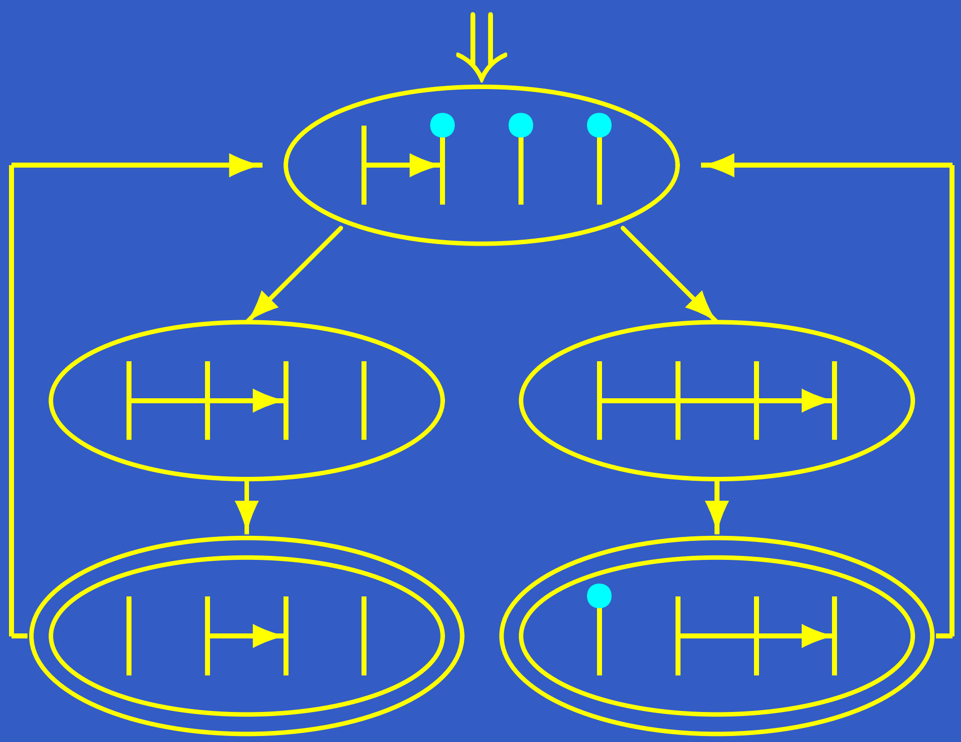
Concatenation of MSCs

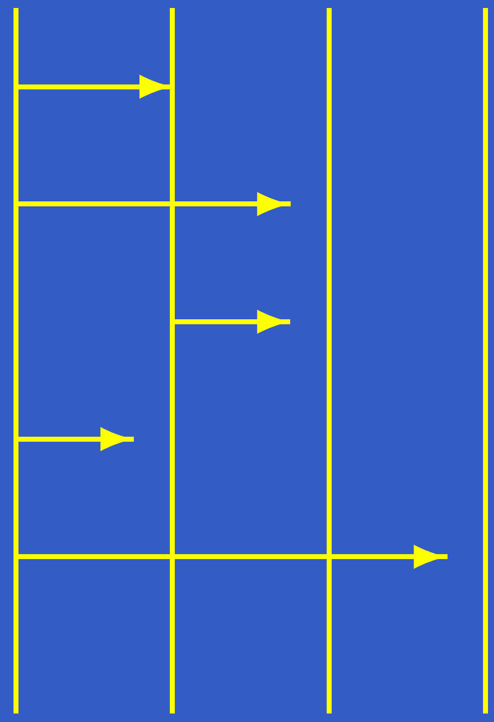
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC



Concatenation of MSCs

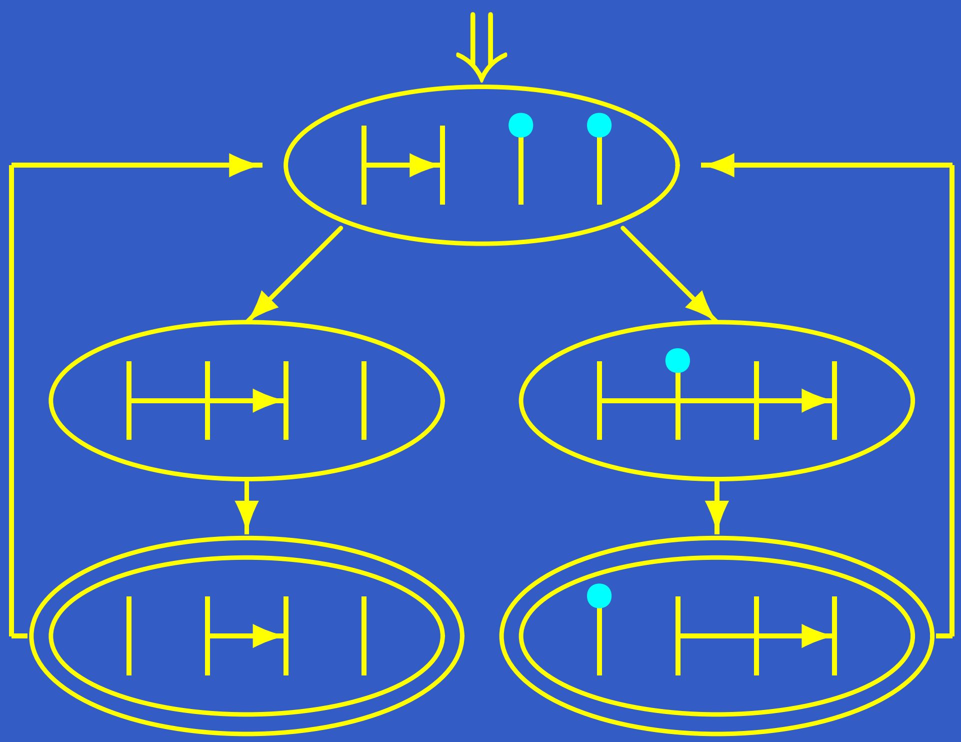
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

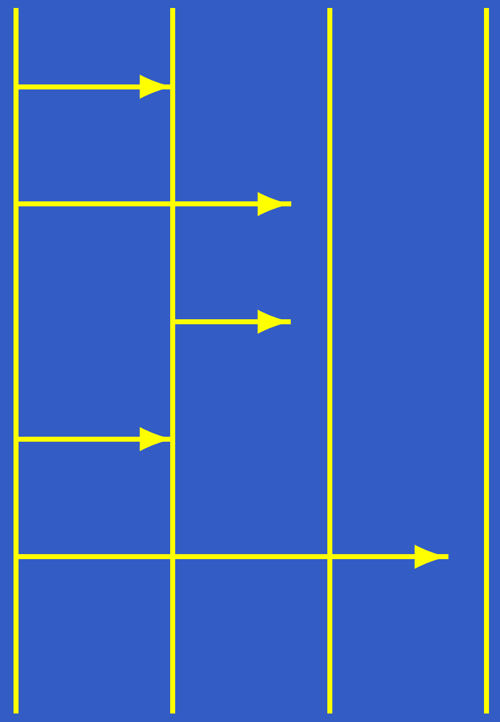




Concatenation of MSCs

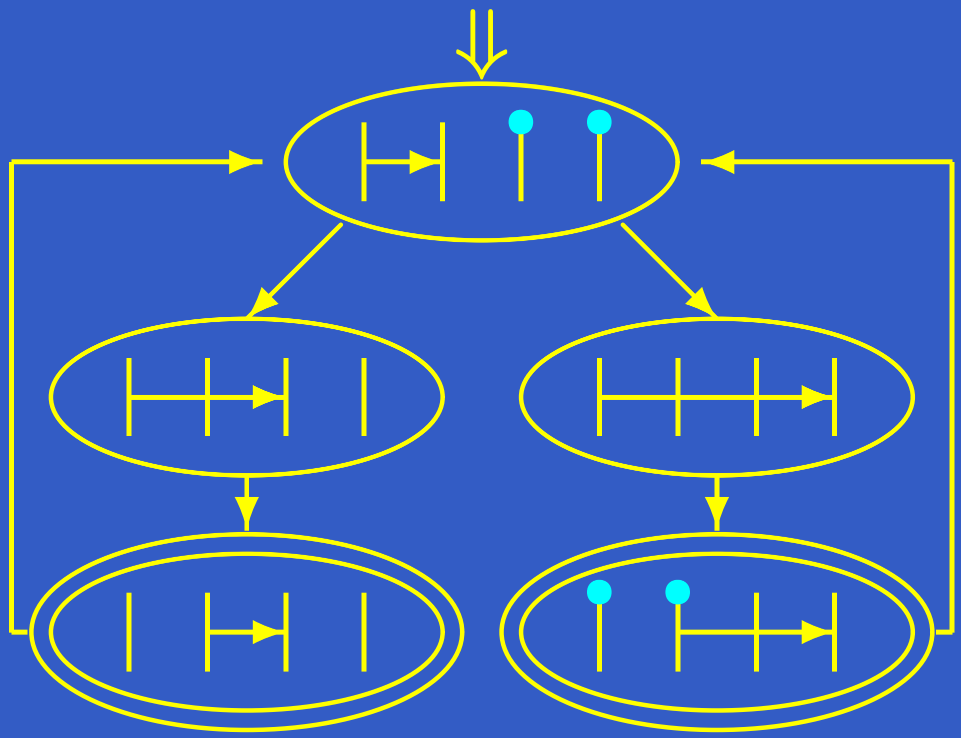
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

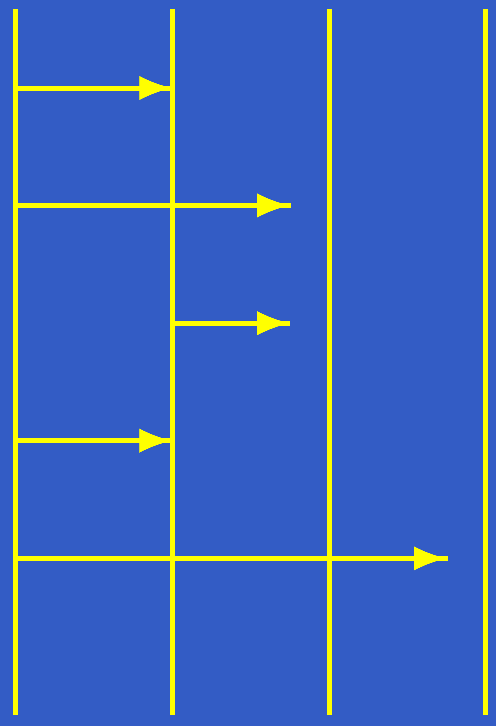




Concatenation of MSCs

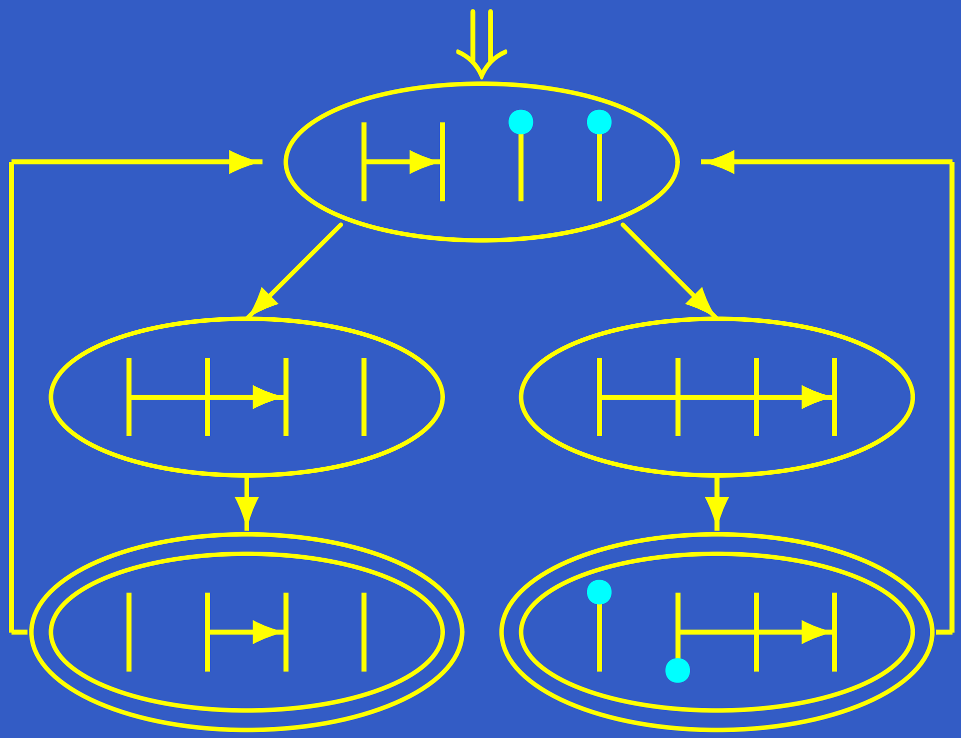
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

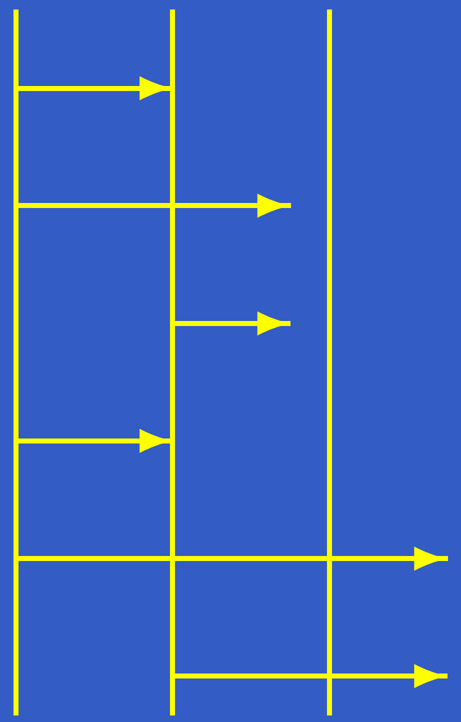




Concatenation of MSCs

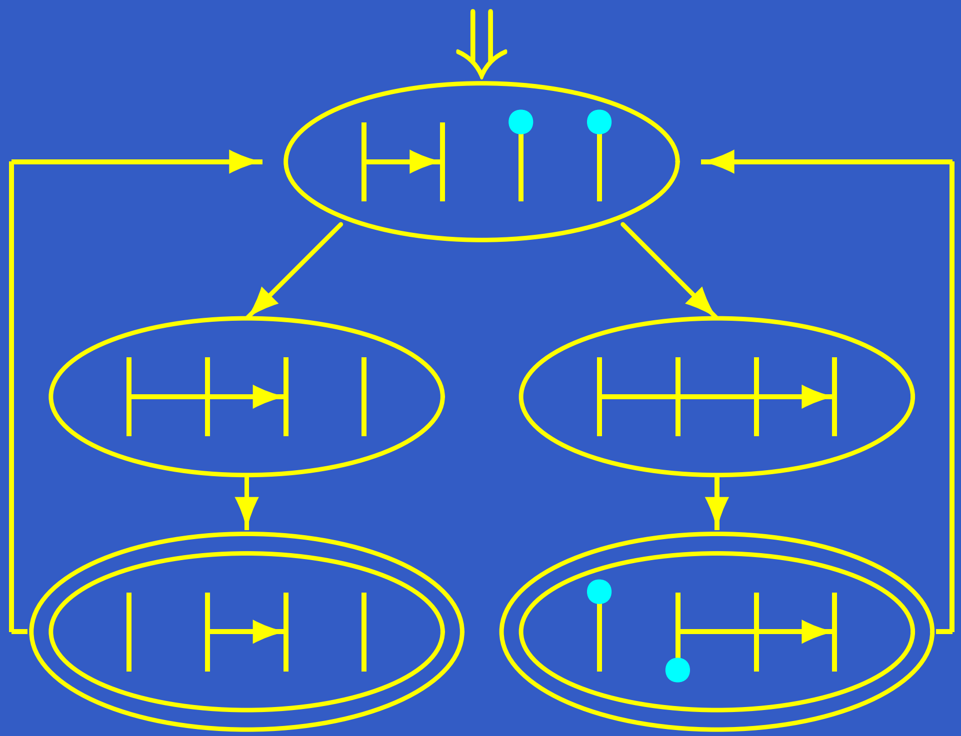
- First MSC finishes before second starts : *synchronous*
- Join MSCs along each process line : *asynchronous*
  - Some processes may proceed to second MSC before others complete actions of first MSC

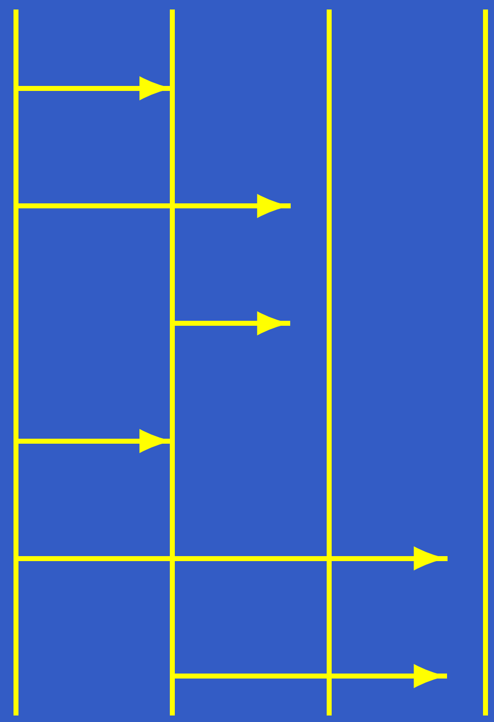




Concatenation of MSCs

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





Verification using scenarios

- A set of scenarios is a behavioural specification
  - May be negative or positive

Verification using scenarios

- A set of scenarios is a behavioural specification
  - May be negative or positive
- Check if an implementation (e.g., set of communicating FSMs) satisfies the specification
- Can use scenarios to filter out (un)interesting executions

Verification using scenarios

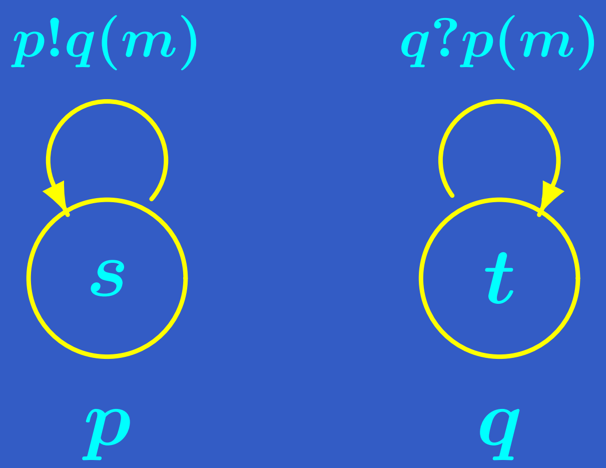
- A set of scenarios is a behavioural specification
  - May be negative or positive
- Check if an implementation (e.g., set of communicating FSMs) satisfies the specification
- Can use scenarios to filter out (un)interesting executions
- Set of scenarios specify both the system and its desired properties
- Checking positive specifications:
  Inclusion of MSC languages
- Checking negative specifications:
  Is the intersection of MSC languages empty

Verification using scenarios . . .

- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels

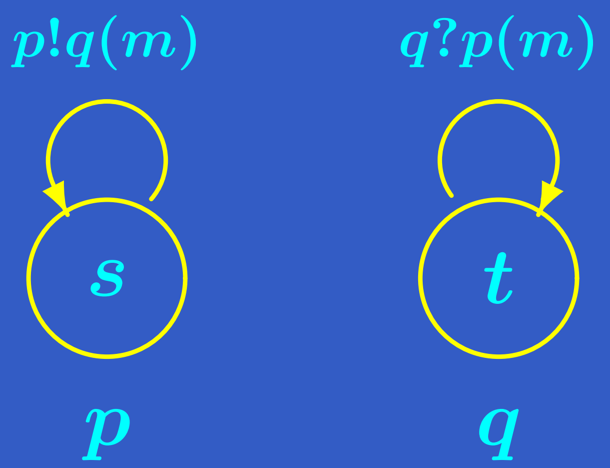
Verification using scenarios . . .

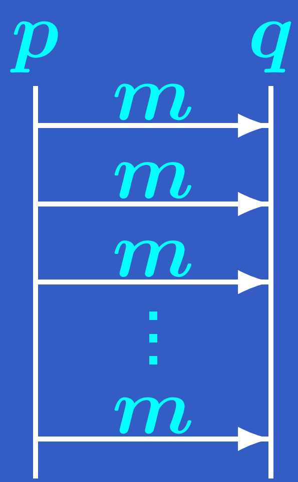
- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels



Verification using scenarios . . .

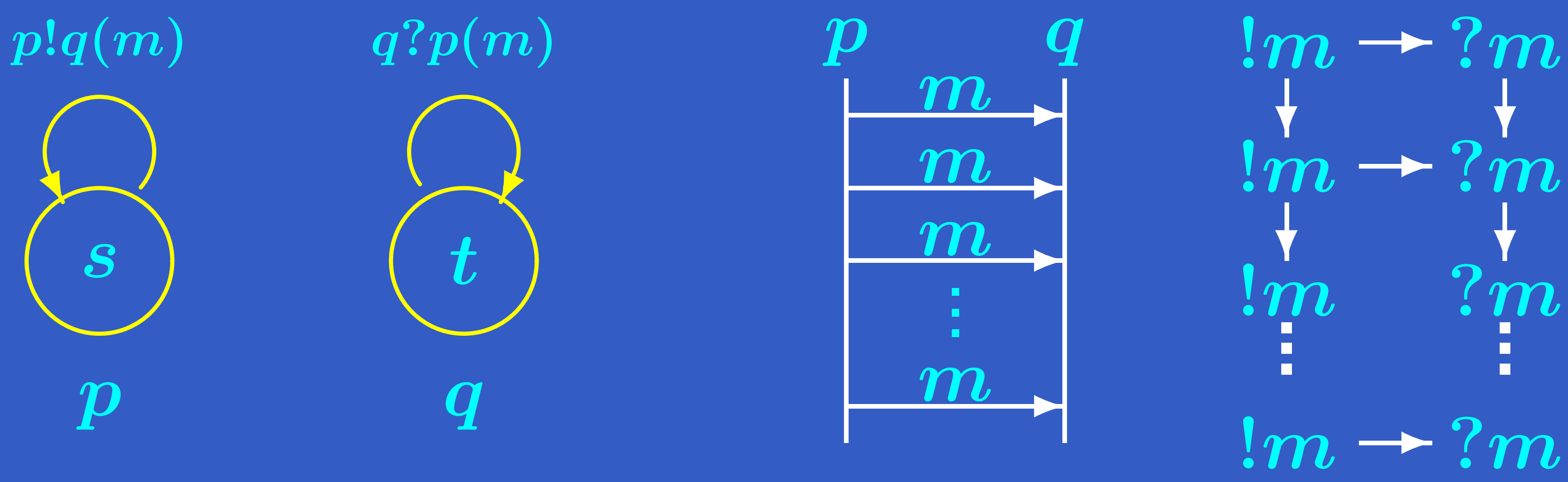
- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels





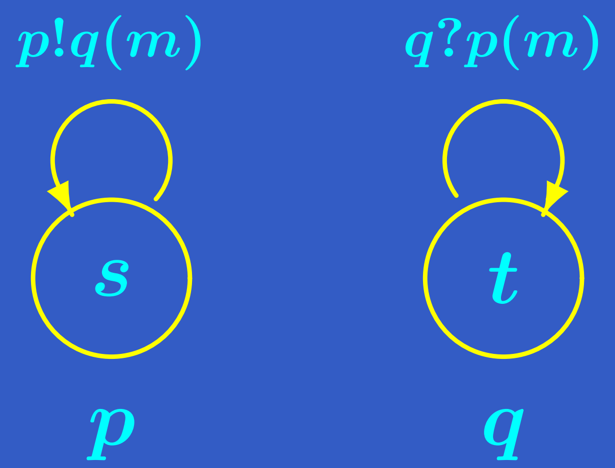
Verification using scenarios ...

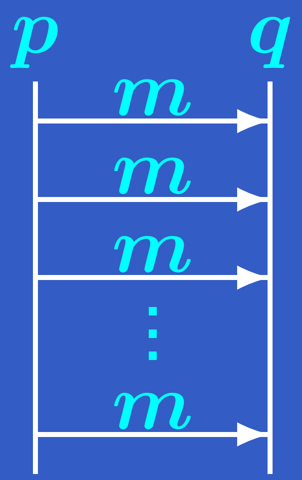
- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels

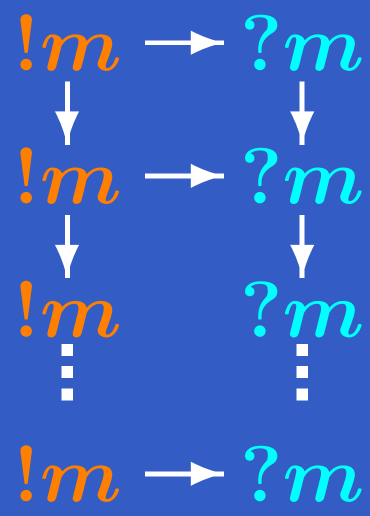


Verification using scenarios . . .

- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels







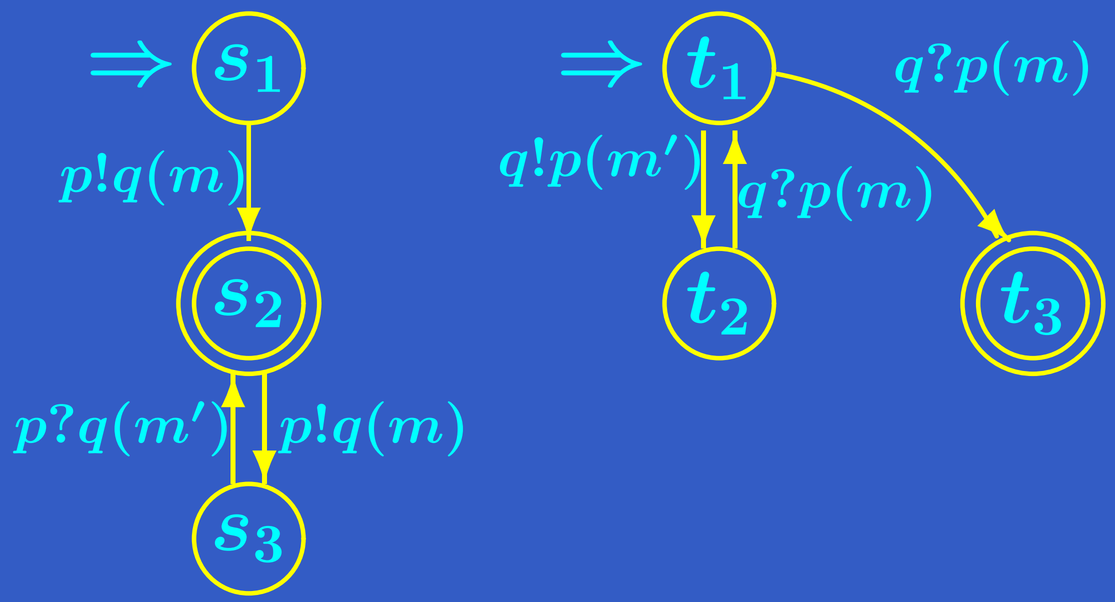
  - No bound on number of messages in channel

Verification using scenarios . . .

- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels
  - Globally finite state $\Rightarrow$ channels are bounded

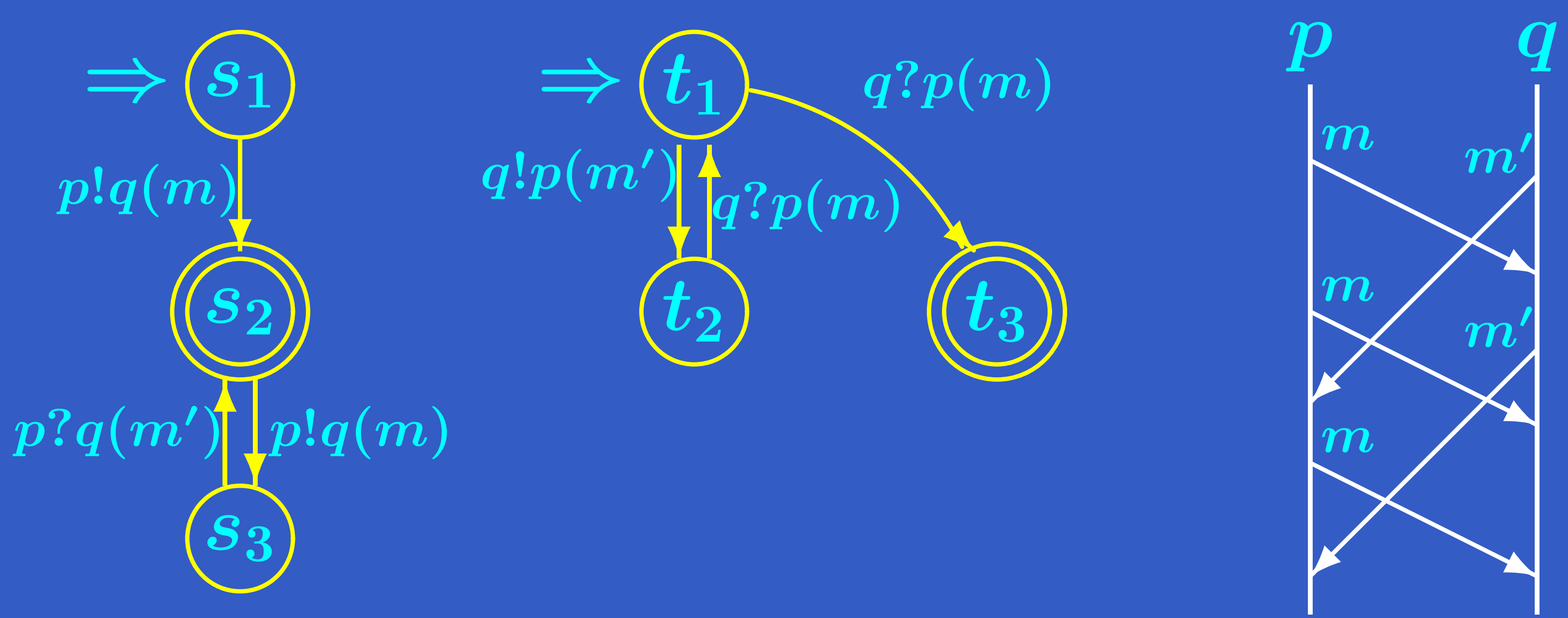
Verification using scenarios . . .

- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels
  - Globally finite state $\Rightarrow$ channels are bounded



Verification using scenarios . . .

- Simplest way is to convert scenarios into executable form — set of communicating finite state-machines
- Execution model
  - Each component is finite state
  - Communication is via (fifo) channels
  - Globally finite state $\Rightarrow$ channels are bounded



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 $p?q(m)$
- Collection of MSCs $\Leftrightarrow$
  word language over send/receive actions

Regular MSC languages

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

Regular MSC languages

- An MSC is (uniquely) determined by its linearizations
  - Set of strings over send actions $p!q(m)$ and receive actions $p?q(m)$
- Collection of MSCs $\Leftrightarrow$
  word language over send/receive actions
- Regular collection of MSCs $\triangleq$
  linearizations form a regular language
- Communicating finite-state machines with bounded channels generate only regular MSC languages

Regular MSC languages

- An MSC is (uniquely) determined by its linearizations
  - Set of strings over send actions $p!q(m)$ and receive actions $p?q(m)$
- Collection of MSCs $\Leftrightarrow$
  word language over send/receive actions
- Regular collection of MSCs $\triangleq$
  linearizations form a regular language
- Communicating finite-state machines with bounded channels generate only regular MSC languages
- Converse is also true [MNS, CONCUR '00]

Regular MSC languages

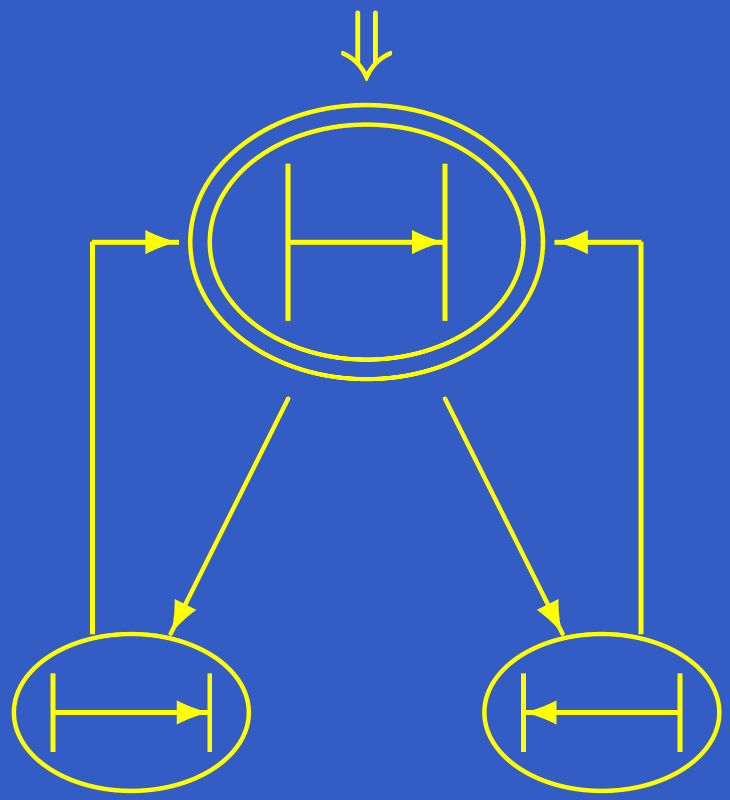
- An MSC is (uniquely) determined by its linearizations
  - Set of strings over send actions $p!q(m)$ and receive actions $p?q(m)$
- Collection of MSCs $\Leftrightarrow$
  word language over send/receive actions
- Regular collection of MSCs $\triangleq$
  linearizations form a regular language
- Communicating finite-state machines with bounded channels generate only regular MSC languages
- Converse is also true [MNS, CONCUR '00]
- Regular MSC languages have a nice theory
  [HMNST, I&C '0?]

HMSCs and regularity

- HMSC specifications may not be regular

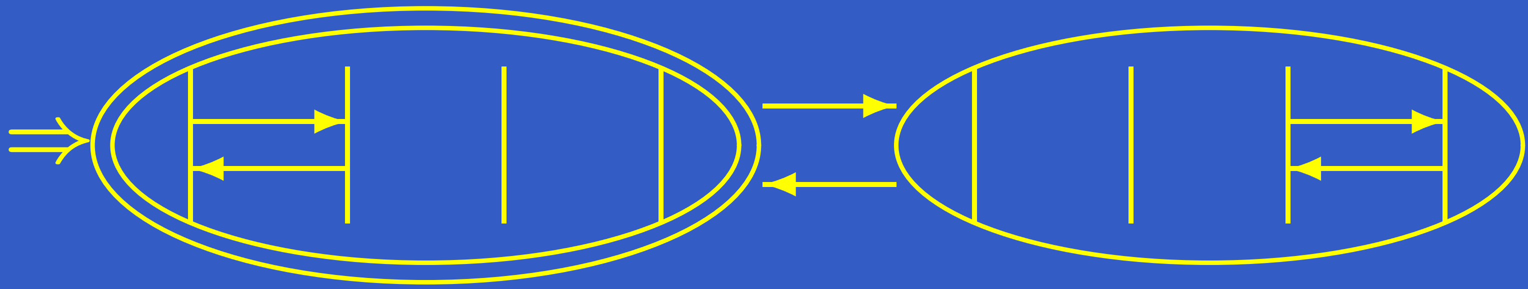
HMSCs and regularity

- 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



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]

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]
- . . . but checking if an HMSC specification is regular is undecidable [HMNT00]

Locally synchronized HMSCs

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

Locally synchronized HMSCs

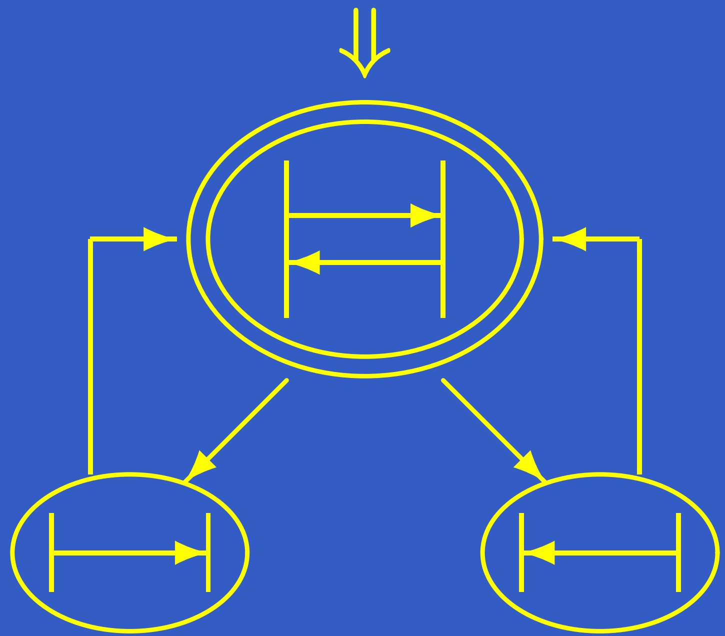
- Construct communication graph for an MSC
  $p \longrightarrow q$ iff p sends a message to q
- For each loop, communication graph is one strongly connected component plus isolated vertices

Locally synchronized HMSCs

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

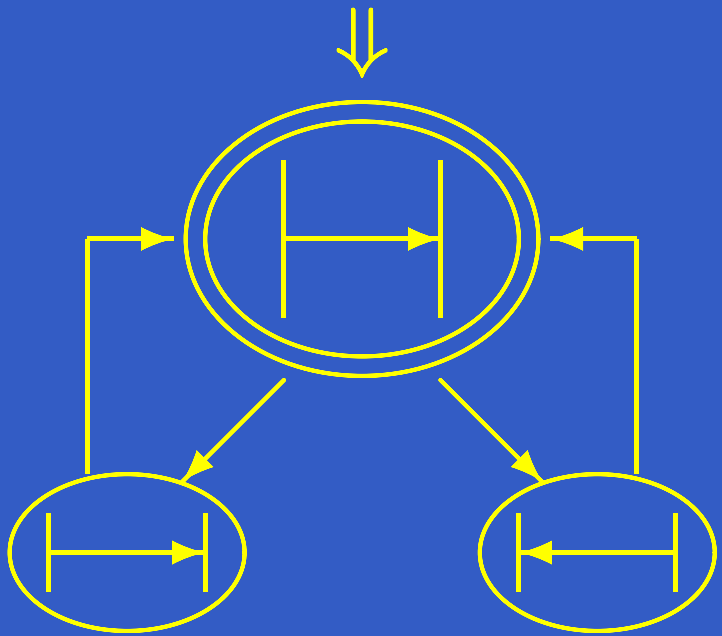
Locally synchronized HMSCs

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



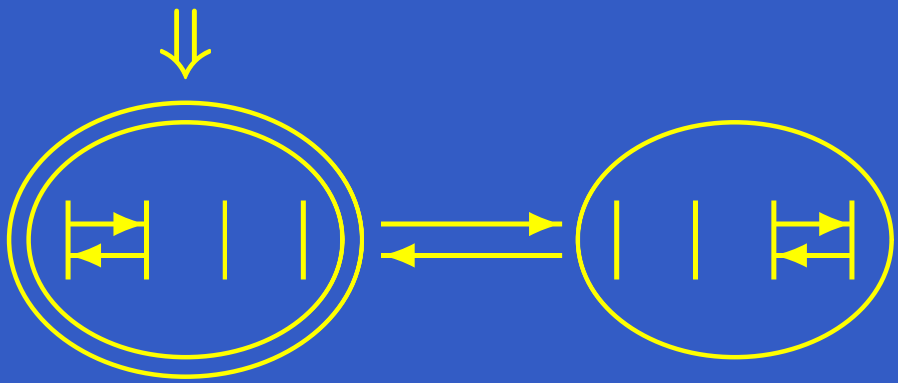
Locally synchronized HMSCs

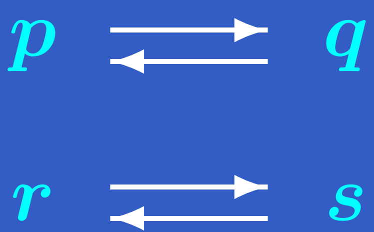
- Construct communication graph for an MSC
  $p \longrightarrow q$ iff p sends a message to q
- For each loop, communication graph is one strongly connected component plus isolated vertices
- In each loop, every message is "acknowledged"



Locally synchronized HMSCs

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





Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages

Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages
- Regular MSC languages can be translated into message-passing automata with bounded channels

Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages
- Regular MSC languages can be translated into message-passing automata with bounded channels
- Use standard automata-theoretic techniques to check $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} = \emptyset$

Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages
- Regular MSC languages can be translated into message-passing automata with bounded channels
- Use standard automata-theoretic techniques to check $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} = \emptyset$
  - Note that for positive scenarios, we want to check $L_{pos} \subseteq L_{sys}$

Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages
- Regular MSC languages can be translated into message-passing automata with bounded channels
- Use standard automata-theoretic techniques to check $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} = \emptyset$
  - Note that for positive scenarios, we want to check $L_{pos} \subseteq L_{sys}$
  - Normal temporal logic verification asks if $L_{sys} \subseteq L_{\varphi}$

Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages
- Regular MSC languages can be translated into message-passing automata with bounded channels
- Use standard automata-theoretic techniques to check $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} = \emptyset$
  - Note that for positive scenarios, we want to check $L_{pos} \subseteq L_{sys}$
  - Normal temporal logic verification asks if $L_{sys} \subseteq L_{\varphi}$
  - $L_1 \subseteq L_2 \Leftrightarrow L_1 \cap \overline{L_2} = \emptyset$

Verification for MSCs

- Locally synchronized HMSCs define regular MSC languages
- Regular MSC languages can be translated into message-passing automata with bounded channels
- Use standard automata-theoretic techniques to check $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} = \emptyset$
  - Note that for positive scenarios, we want to check $L_{pos} \subseteq L_{sys}$
  - Normal temporal logic verification asks if $L_{sys} \subseteq L_{\varphi}$
  - $L_1 \subseteq L_2 \Leftrightarrow L_1 \cap \overline{L_2} = \emptyset$
  - For scenarios, we have to complement L_{sys}, not L_{spec}

Verification for MSCs . . .

- Model checking possible for larger classes than regular HMSC languages [GMSZ, ICALP ’02 GMK, DLT ’04]

Verification for MSCs . . .

- Model checking possible for larger classes than regular HMSC languages [GMSZ, ICALP '02 GMK, DLT '04]
  - *Existentially* bounded channels

Verification for MSCs . . .

- Model checking possible for larger classes than regular HMSC languages [GMSZ, ICALP '02 GMK, DLT '04]
  - *Existentially* bounded channels
  - For every MSC in L, there is an ordering of events which bounds the channels

Verification for MSCs . . .

- Model checking possible for larger classes than regular HMSC languages [GMSZ, ICALP '02 GMK, DLT '04]
  - *Existentially* bounded channels
  - For every MSC in $\boldsymbol{L}$, there is an ordering of events which bounds the channels
- Construct a regular set of representative linearizations that cover the MSC language

Interpreting MSCs

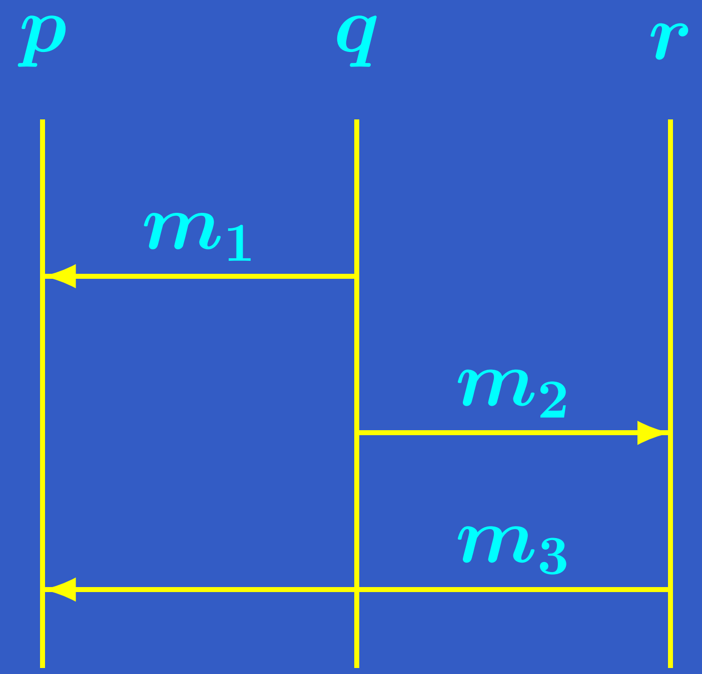
- The visual notation of MSCs is appealing

Interpreting MSCs

- The visual notation of MSCs is appealing . . .
- . . . but can also be misleading

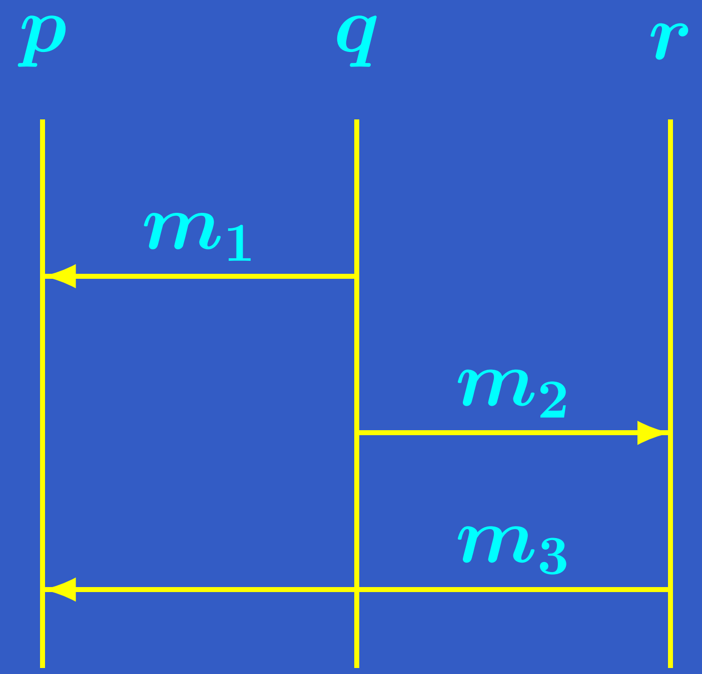
Interpreting MSCs

- The visual notation of MSCs is appealing . . .
- . . . but can also be misleading



Interpreting MSCs

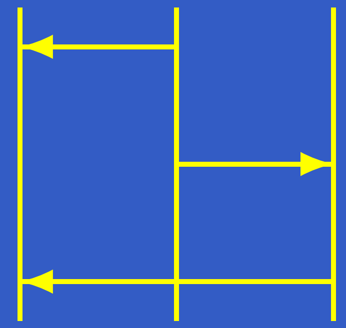
- The visual notation of MSCs is appealing . . .
- . . . but can also be misleading



- Is it reasonable to insist that m_1 arrives before m_3?

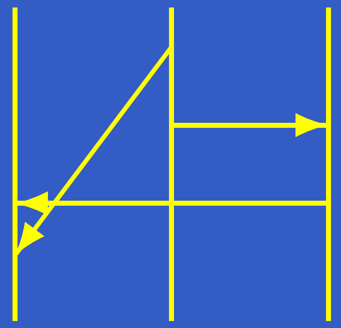
Race Conditions [AHP, TACAS '96]

- Independent receive events can be interchanged



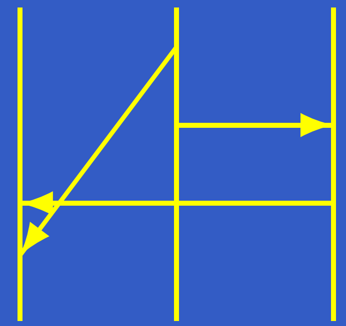
Race Conditions [AHP, TACAS '96]

- Independent receive events can be interchanged

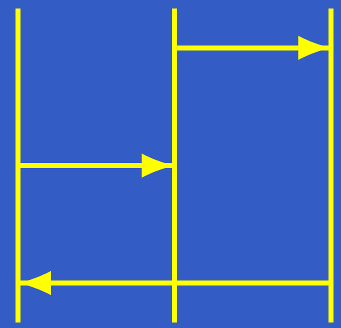


Race Conditions [AHP, TACAS '96]

- Independent receive events can be interchanged

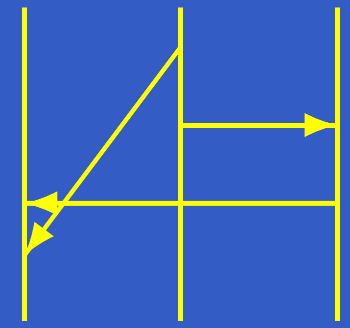


- Send event before a receive event can be swapped (but not vice versa!)

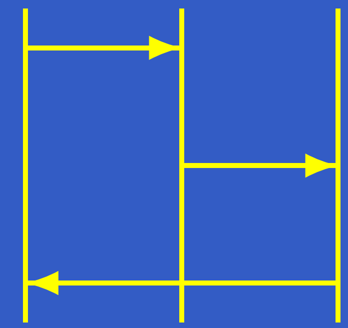


Race Conditions [AHP, TACAS '96]

- Independent receive events can be interchanged



- Send event before a receive event can be swapped (but not vice versa!)



Verification for MSCs . . .

- Implementation may add additional messages

Verification for MSCs . . .

- Implementation may add additional messages
- MSC M *matches* MSC M' if events of M can be embedded injectively in M'

Verification for MSCs . . .

- Implementation may add additional messages
- MSC M *matches* MSC M' if events of M can be embedded injectively in M'
- Scenario matching with embedding

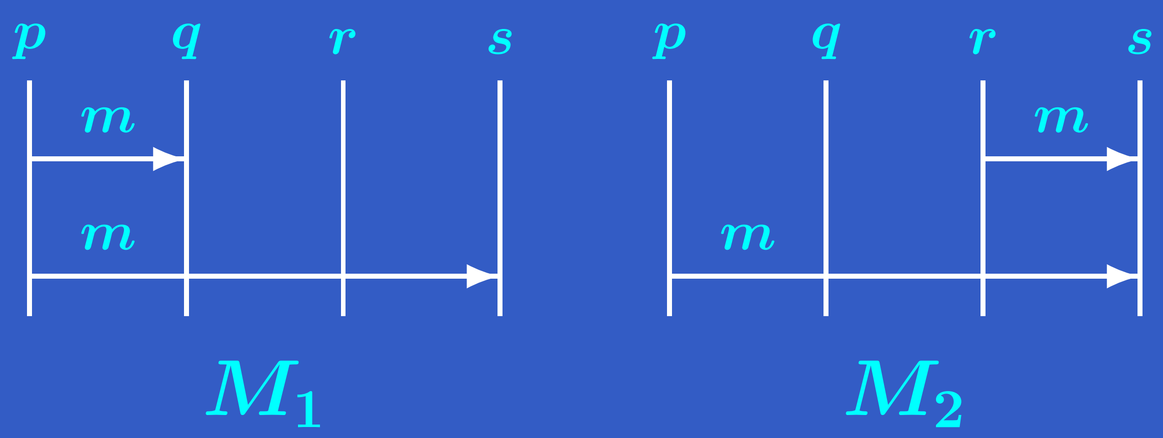
Verification for MSCs . . .

- Implementation may add additional messages
- MSC M *matches* MSC M' if events of M can be embedded injectively in M'
- Scenario matching with embedding
  - Greedy algorithm works with closure under race conditions [MPS, FOSSACS '98]

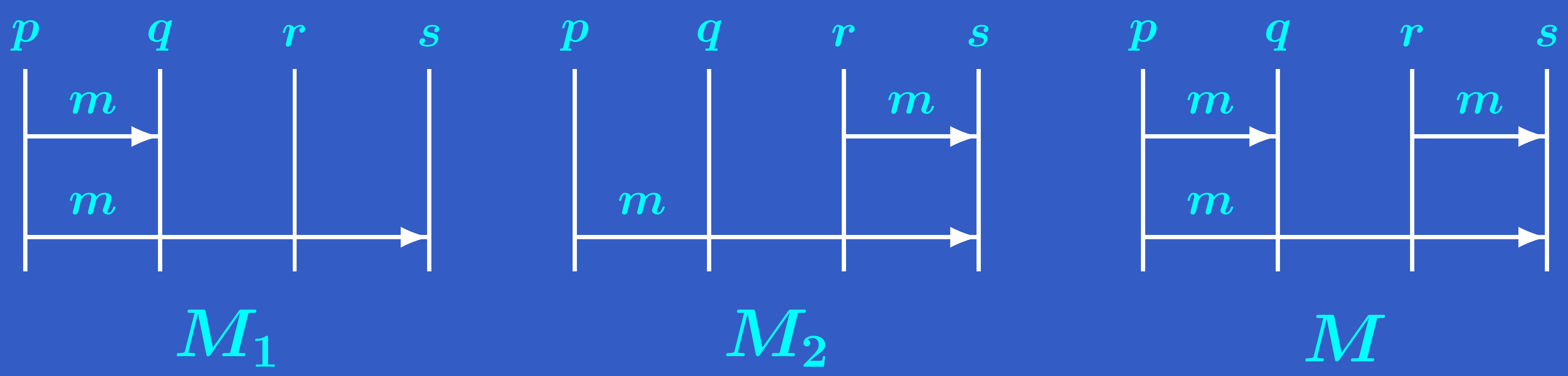
Verification for MSCs . . .

- Implementation may add additional messages
- MSC M *matches* MSC M' if events of M can be embedded injectively in M'
- Scenario matching with embedding
  - Greedy algorithm works with closure under race conditions [MPS, FOSSACS '98]
  - Without race condition closure, backtracking appears unavoidable [DM, SPIN '03]

Implied scenarios [AEY, ICSE ’00]

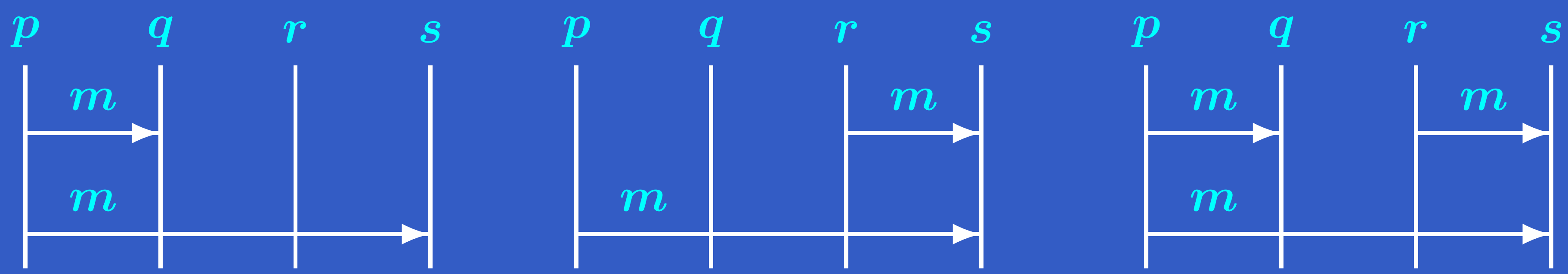


Implied scenarios [AEY, ICSE '00]



- p and q believe M is M_1
- r and s believe M is M_2

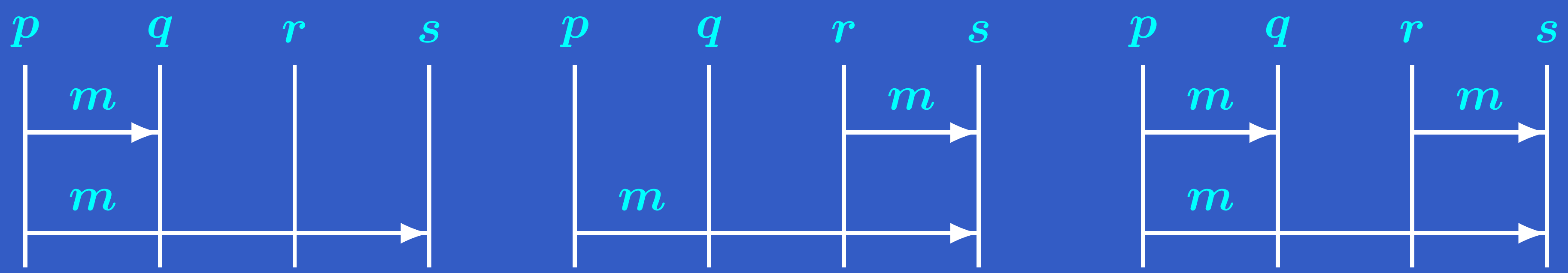
Implied scenarios [AEY, ICSE '00]



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]



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
- An MSC language is weakly realizable if it is closed with respect to implied MSCs

Implied scenarios . . .

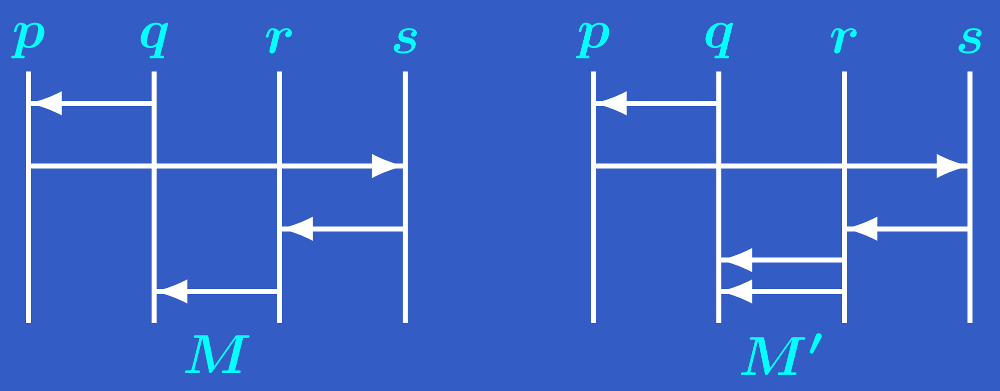
- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP ’01]

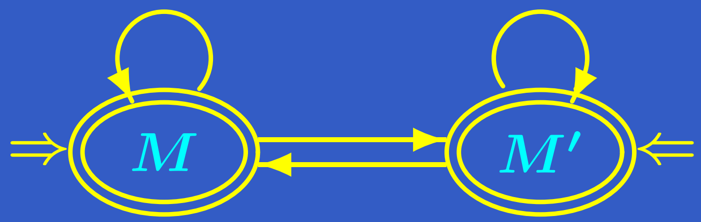
Implied scenarios . . .

- 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 . . .

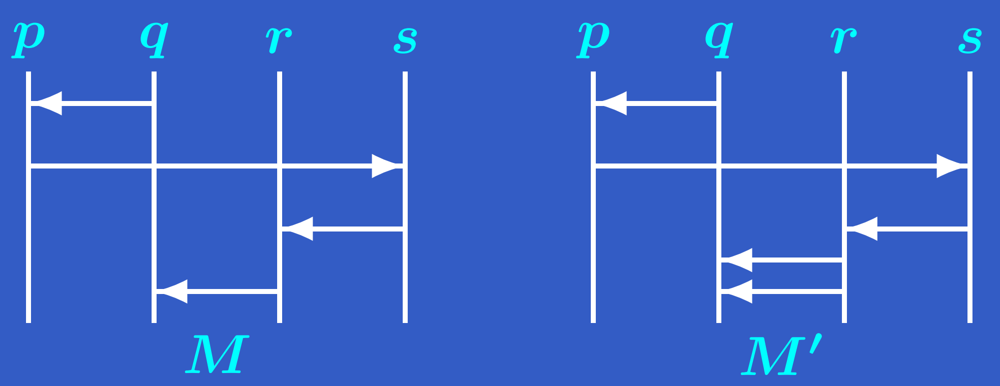
- Even for regular MSC languages, checking weak realizability is undecidable! [AEY, ICALP '01]
- Even if the original language has bounded channels, its weak closure may not

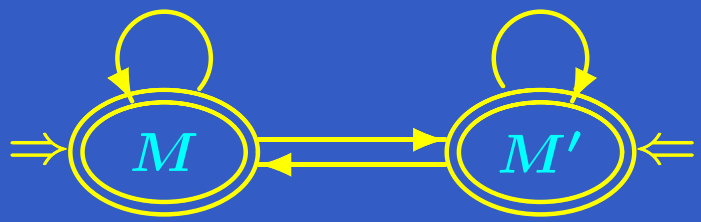


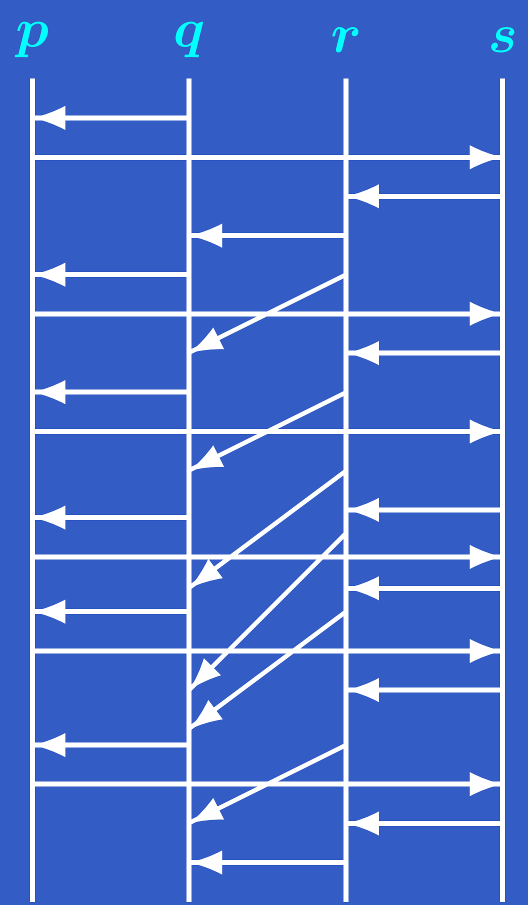


Implied scenarios . . .

- 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

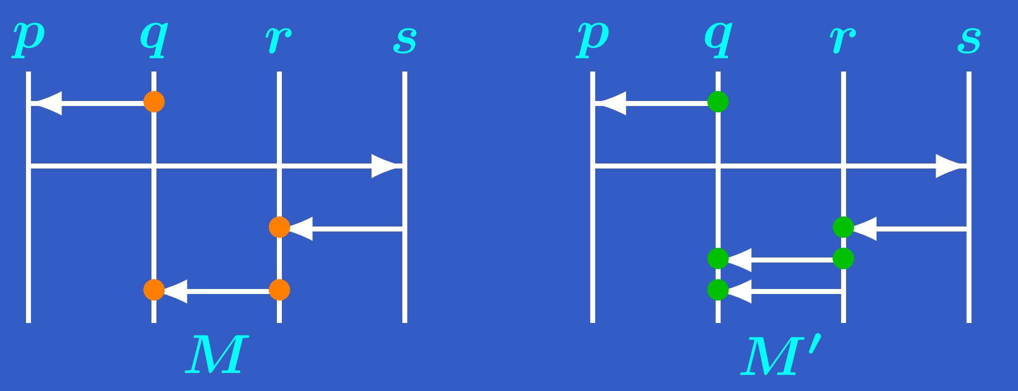


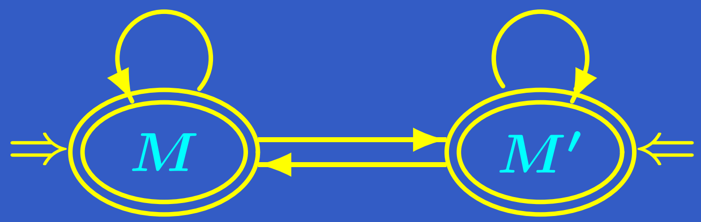


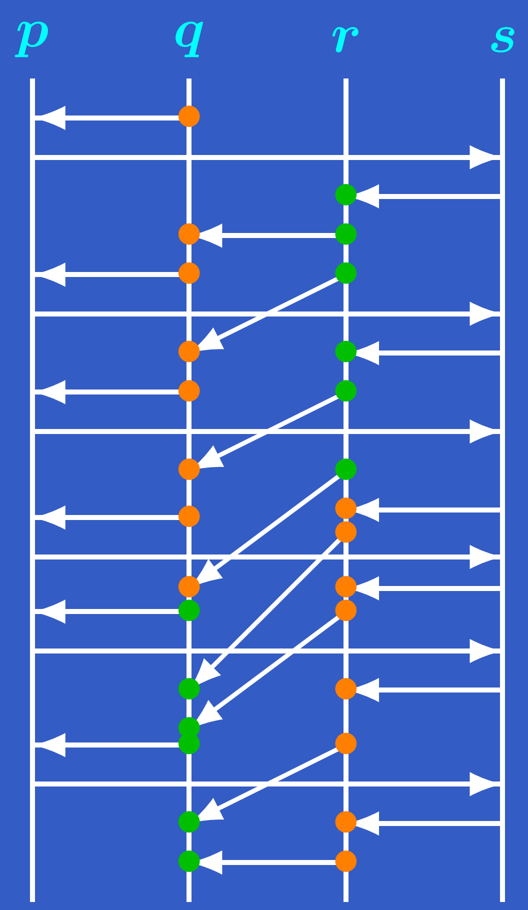


Implied scenarios . . .

- 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

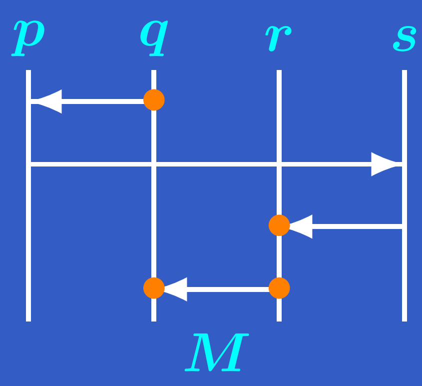


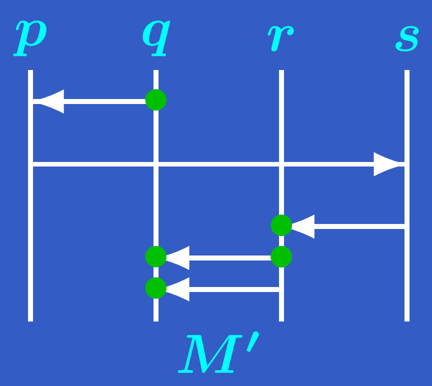


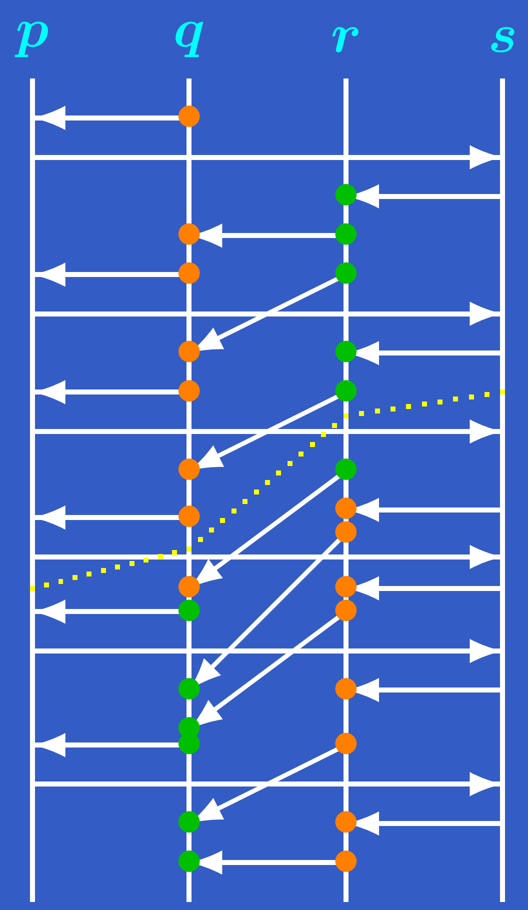


Implied scenarios . . .

- 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







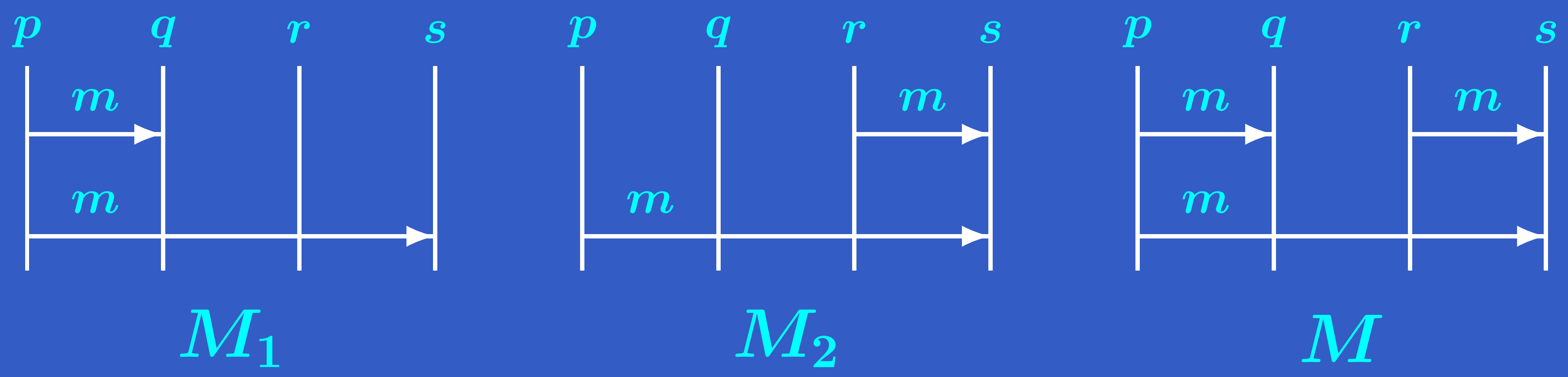
Confusing $M^{2k}M'^{k}$ and $M'^{k}M^{2k}$ generates upto k messages in $p \longrightarrow s$ channel

Beyond weak realizability

- Assumption underlying weak realizability
  The only information that a process can maintain locally is its own action history

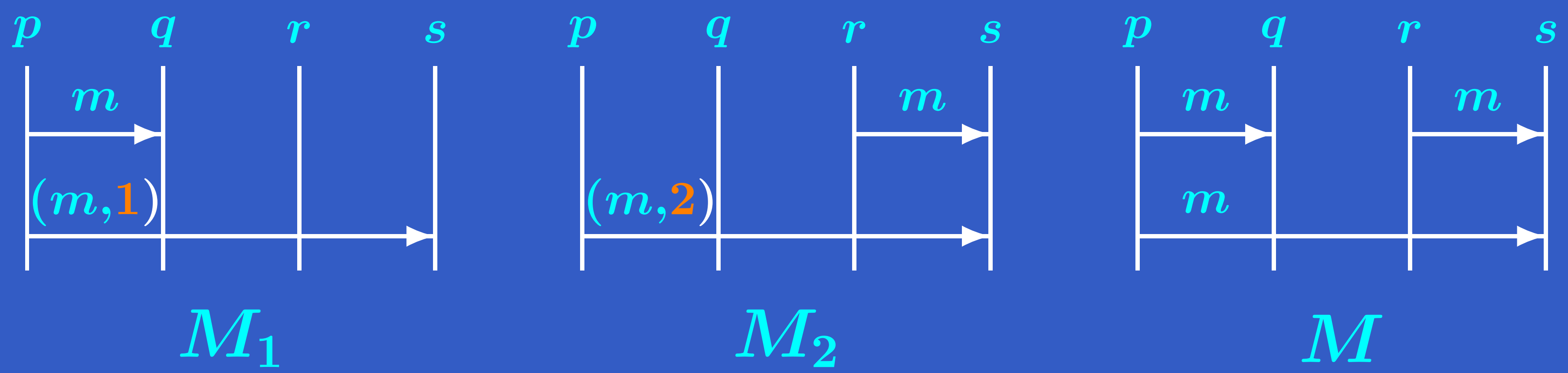
Beyond weak realizability

- Assumption underlying weak realizability
  The only information that a process can maintain locally is its own action history



Beyond weak realizability

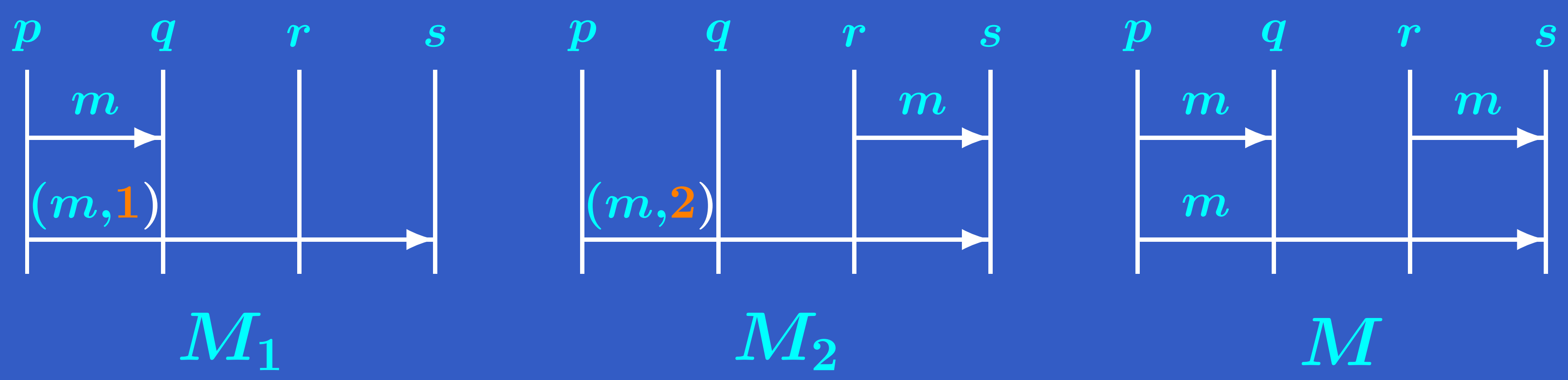
- Assumption underlying weak realizability
  The only information that a process can maintain locally is its own action history



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

Beyond weak realizability

- Assumption underlying weak realizability
  The only information that a process can maintain locally is its own action history



- 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 [AMNN, CMI-TR 05]

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

Causal closure [AMNN, CMI-TR 05]

- Recall that an MSC is a partially ordered set of events
- For a process p and an 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 [AMNN, CMI-TR 05]

- Recall that an MSC is a partially ordered set of events
- For a process p and an 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 [AMNN, CMI-TR 05]

- Recall that an MSC is a partially ordered set of events
- For a process p and an 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 realizable if it is closed with respect to causal implication

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 recognizing the causal closure

HMSCs and causal closure

- The causal closure of a regular HMSC language is not always HMSC definable

HMSCs and causal closure

- The causal closure of a regular HMSC language is not always HMSC definable
- Every HMSC language L is finitely generated

HMSCs and causal closure

- The causal closure of a regular HMSC language is not always HMSC definable
- Every HMSC language L is finitely generated
  - Finite set $A = \{A_1, \ldots, A_k\}$ of (atomic) MSCs

HMSCs and causal closure

- The causal closure of a regular HMSC language is not always HMSC definable
- Every HMSC language L is finitely generated
  - Finite set $A = \{A_1, \ldots, A_k\}$ of (atomic) MSCs
  - Every MSC in L is a concatenation of MSCs from A

HMSCs and causal closure

- The causal closure of a regular HMSC language is not always HMSC definable
- Every HMSC language L is finitely generated
  - Finite set $A = \{A_1, \ldots, A_k\}$ of (atomic) MSCs
  - Every MSC in L is a concatenation of MSCs from A
- In general, regular MSC languages may not be finitely generated

HMSCs and causal closure

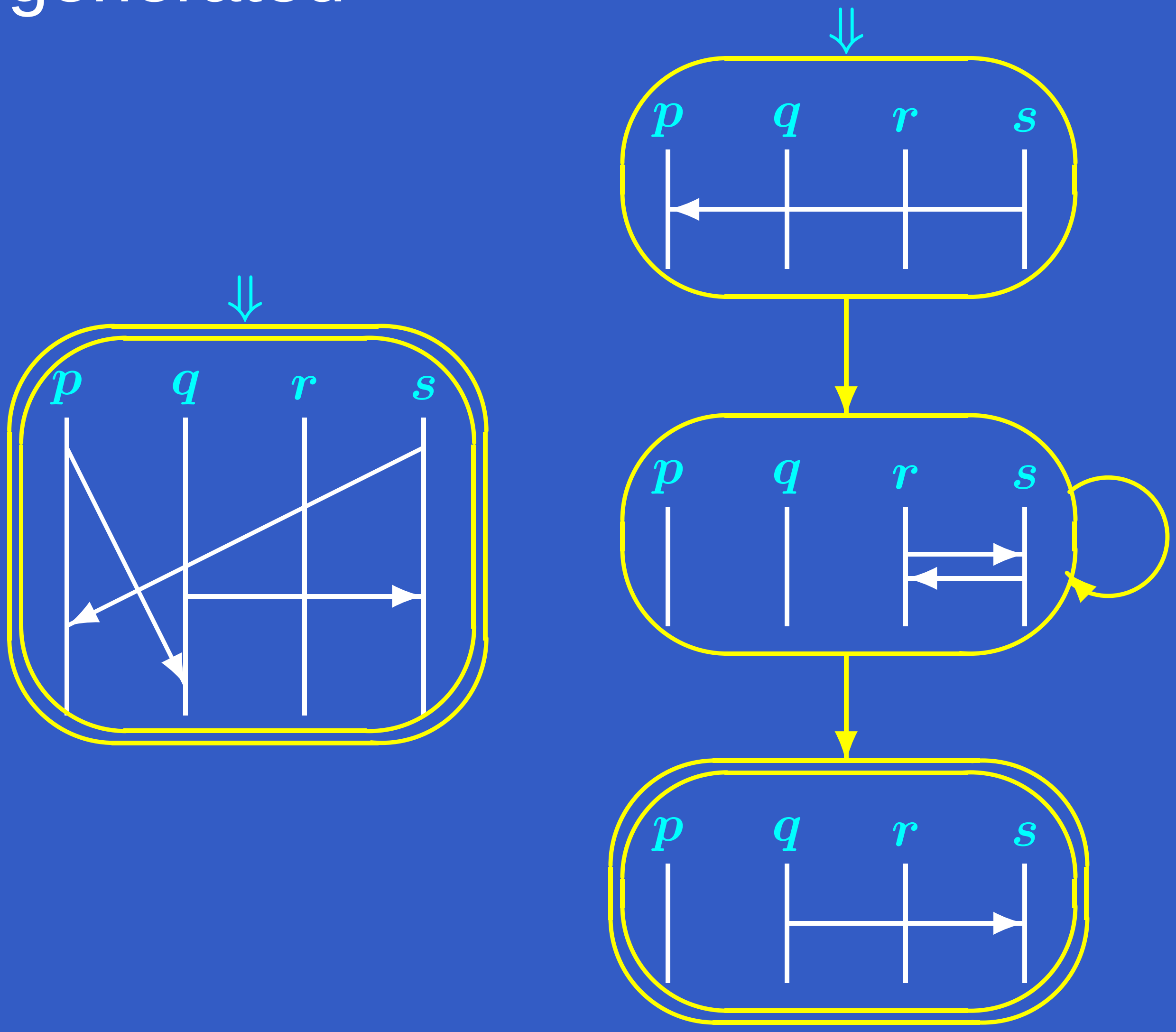
- The causal closure of a regular HMSC language is not always HMSC definable
- Every HMSC language L is finitely generated
  - Finite set $A = \{A_1, \ldots, A_k\}$ of (atomic) MSCs
  - Every MSC in L is a concatenation of MSCs from A
- In general, regular MSC languages may not be finitely generated
- Causal closure of a regular HMSC language may contain an infinite collection of atoms

HMSCs and causal closure

Causal closure of an HMSC language may not be finitely generated

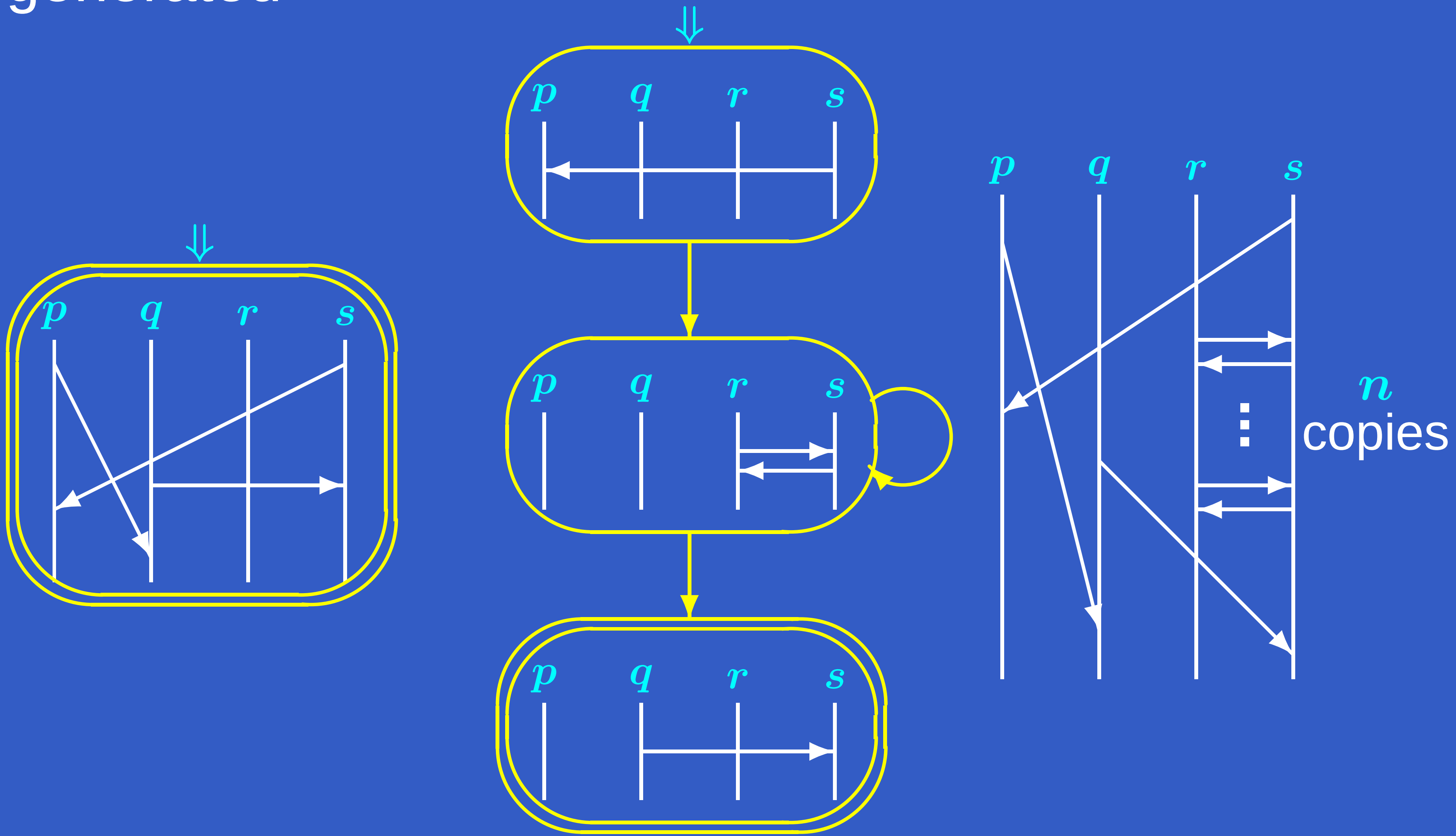
HMSCs and causal closure

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



Verification using scenarios . . .

- Semantics in terms of causal closure makes HMSCs notation more expressive

Verification using scenarios . . .

- Semantics in terms of causal closure makes HMSCs notation more expressive
- “Reasonable” implementations will be causally closed

Verification using scenarios . . .

- Semantics in terms of causal closure makes HMSCs notation more expressive
- “Reasonable” implementations will be causally closed
- Verifying positive scenarios P against system behaviours S:

Verification using scenarios . . .

- Semantics in terms of causal closure makes HMSCs notation more expressive
- "Reasonable" implementations will be causally closed
- Verifying positive scenarios P against system behaviours S:
  - Both P and S should be causally closed to avoid missing out some scenarios when checking $P \subseteq S$.

Verification using scenarios . . .

- Semantics in terms of causal closure makes HMSCs notation more expressive
- "Reasonable" implementations will be causally closed
- Verifying positive scenarios P against system behaviours S:
  - Both P and S should be causally closed to avoid missing out some scenarios when checking $P \subseteq S$.
- Verifying that a negative property N is not present in S:

Verification using scenarios . . .

- Semantics in terms of causal closure makes HMSCs notation more expressive
- “Reasonable” implementations will be causally closed
- Verifying positive scenarios P against system behaviours S:
  - Both P and S should be causally closed to avoid missing out some scenarios when checking $P \subseteq S$.
- Verifying that a negative property N is not present in S:
  - N should be causally closed so no forbidden scenario goes undetected.

Summary

- Scenario based specifications are intuitively easy to use

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$
  - Can be solved for regular MSC languages

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$
  - Can be solved for regular MSC languages
  - Extended to existentially bounded MSC languages

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$
  - Can be solved for regular MSC languages
  - Extended to existentially bounded MSC languages
- Semantics of MSCs is not completely straightforward

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$
  - Can be solved for regular MSC languages
  - Extended to existentially bounded MSC languages
- Semantics of MSCs is not completely straightforward
  - Race conditions

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$
  - Can be solved for regular MSC languages
  - Extended to existentially bounded MSC languages
- Semantics of MSCs is not completely straightforward
  - Race conditions
  - Implied scenarios

Summary

- Scenario based specifications are intuitively easy to use
- Verification requires computing $L_{pos} \subseteq L_{sys}$ and $L_{neg} \cap L_{sys} \neq \emptyset$
  - Can be solved for regular MSC languages
  - Extended to existentially bounded MSC languages
- Semantics of MSCs is not completely straightforward
  - Race conditions
  - Implied scenarios
- To what extent can verification be done with enriched semantics for MSCs?

Related approaches

Related approaches not covered in this talk

- *Live sequence charts* by Harel et al
- Verification of *Lamport diagrams* by Meenakshi and Ramanujam

References

B Adsul, M Mukund, K Narayan Kumar and Vasumathi Narayanan
Causal closure for MSC languages
Internal Report, Chennai Mathematical Institute (2005)

R Alur, K Etessami and M Yannakakis
Realizability and Verification of MSC Graphs
Theoretical Computer Science, 331(1), (2005) 97–114.

R Alur, G Holzmann and D Peled
An analyzer for message sequence charts
Software Concepts and Tools, 17(2) (1996) 70–77.

R Alur and M Yannakakis
Model checking of message sequence charts
CONCUR 1999, Springer LNCS 1664 (1999) 114–129.

B Genest, A Muscholl and D Kuske
A Kleene Theorem for a Class of Communicating Automata with Effective Algorithms
Proc DLT 2004, Springer LNCS 3340 (2004) 30–48.

B Genest, A Muscholl, H Seidl and M Zeitoun
Infinite-State High-Level MSCs: Model-Checking and Realizability
ICALP 2002, Springer LNCS 2380 (2002) 657–668.

J G Henriksen, M Mukund, K Narayan Kumar, M Sohoni and P S Thiagarajan
A Theory of Regular MSC Languages
Information and Computation (to appear).

A Muscholl and D Peled
Message sequence graphs and decision problems on Mazurkiewicz traces
MFCS 1999, Springer LNCS 1672 (1999) 81–91.

A Muscholl, D Peled and Z Su
Deciding properties for message sequence charts
FOSSACS'98, Springer LNCS 1378 (1998) 226–242.