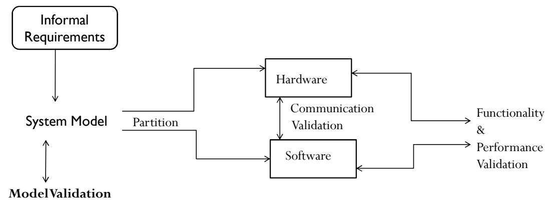
# Modeling Notations
# CS 4271 lecture 2

Abhik Roychoudhury
National University of Singapore
http://www.comp.nus.edu.sg/~abhik/

## Different kinds of ES Validation



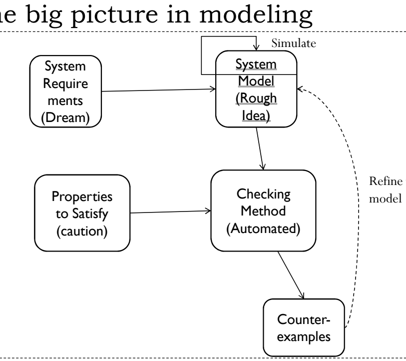

 

## What is a system design model?

- We first clarify the following terms
  - System Architecture: Inter-connection among the system components.
  - System behavior: How the components change state, by communicating among themselves.
- System Design Model = Architecture + Behavior
  - More precise definition later.

## The big picture in modeling



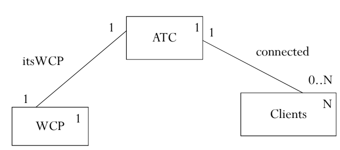

 

## Criteria for a Design Model

- Provides structure as well as behavior for the system components.
- Complete
  - Complete description of system behavior.
- Based on well-established modeling notations.
  - We use UML.
- Preferably executable
  - Can simulate the model, and get a feel for how the constructed system will behave!

## Running Example - ATC




**Overall System Structure, Behavior not shown.**

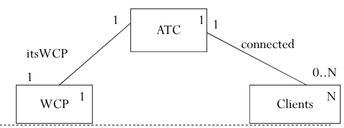
## On system behavior

- Consider a "scenario"
  - Client1 sends "connect" request to ATC
  - Client2 sends "connect" request to ATC
  - ATC sends weather information to Client1, Client2.
- No need to capture "weather info." in model.
- OK to abstract this info. from the requirements while constructing the model, provided
  - No decisions are made in the system based on weather info.
- Model is "complete" at a certain level of abstraction.




## ATC – the example control sys.

- NASA CTAS
  - Automation tools for managing large volume arrival air traffic in large airports.
  - Final Approach Spacing Tool
    - Determine speed and trajectory of incoming aircrafts on their final approach.
    - Master controller updates weather info. to "clients"
      - **controllers using inputs to compute aircraft trajectories.**







## ATC – the example control sys.

- Part of the *Center TRACON Automation System (CTAS)* by NASA
  - manage high volume of arrival air traffic at large airports
  - **http://ctas.arc.nasa.gov**
- Control weather updating to all weather-aware clients
  - A weather control panel (WCP)
  - Many weather-aware clients
  - A communication manager (CM)

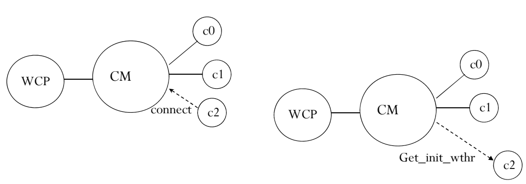



## Behavior of ATC example

- Two standard behaviors
  - Client initialization
  - Weather update
- Abstracted Information
  - Weather information types
  - Clients types
  - Internal computation on weather information
- For simplified requirements: textbook Chap 2.3



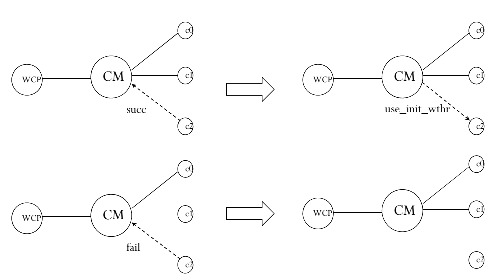

## Client Initialization



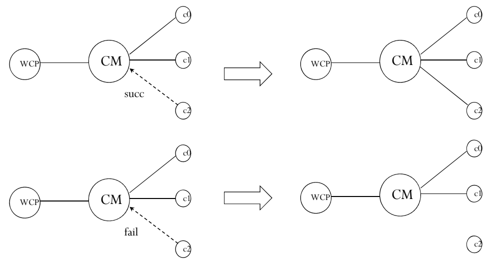




## Client - Initialization

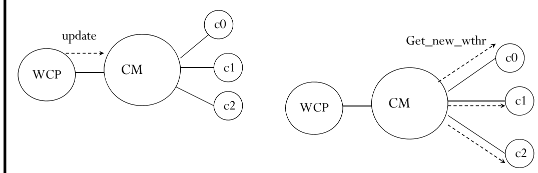


## Client - Initialization



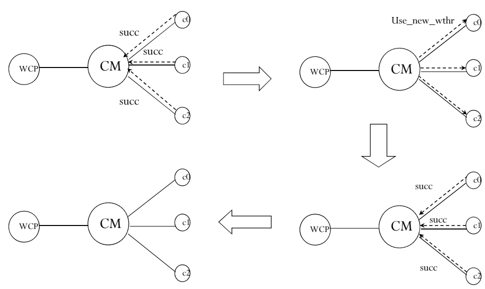

 

## Client – Weather Update



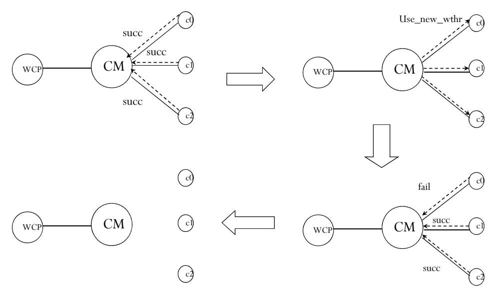

 

## Client Update – Case 1



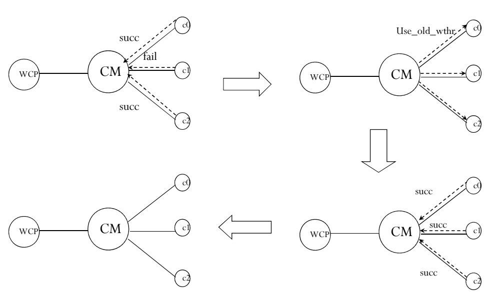

 

## Client Update – Case 2



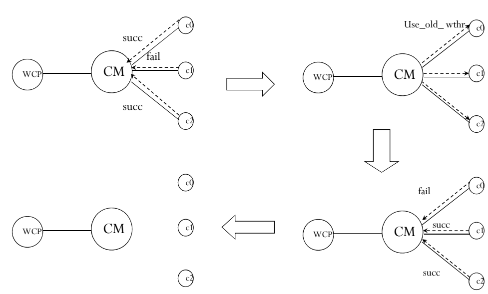

 

## Client Update – Case 3



## Client Update – Case 4



## What do the requirements

- … look like ?

A weather update controller is consist of a weather control panel (WCP), a number of weather-aware clients, and a communication manager (ATC) which controls the interactions between the WCP and all connected clients. Initially, the WCP is enabled for manually weather updating, the ATC is at its idle status, and all the clients are disconnected. Two standard behaviors of this system are as follows.

## Sample Initialization Requirements

- A disconnected weather-aware client can establish a connection by sending a connecting request to the CM.
- If the ATC's status is idle when the connecting request is received, it will set both its own status and the connecting client's status to preinitializing, and disable the weather control panel so that no manual updates can be made by the user during the process of client initialization.
  - Otherwise (ATC's status is not idle), the ATC will send a message to the client to refuse the connection, and the client remains disconnected.

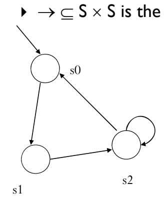
## **Organization**

- So Far
  - What is a Model?
  - ATC – Running Example
    - Informal Req. at a lab scale.
    - Has subtle deadlock error (see textbook chap 2.3)
- Now, how to model/validate such requirements
  - Modeling Notations
    - Finite State Machines

## Finite State Machines

- $M = (S, I, \rightarrow)$
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\rightarrow \subseteq S \times S$ is the transition relation.




$S = \{s0, s1, s2\}$

$I = \{s0\}$

$\rightarrow = \{(s0,s1), (s1,s2), (s2,s2), (s2,s0)\}$

## Issues in system modeling …

- … using FSMs
  - Unit step: How much computation does a single transition denote?
  - Hierarchy: How to visualize a FSM model at different levels of details?
  - Concurrency: How to compose the behaviors of concurrently running subsystems (of a large sys.)
    - Each subsystem is modeled as an FSM!

## What's in a step?

- For hardware systems
  - A single clock cycle
- For software systems
  - Atomic execution of a "minimal" block of code
    - A statement or an instruction?
    - Depends on the level at which the software system is being modeled as an FSM !

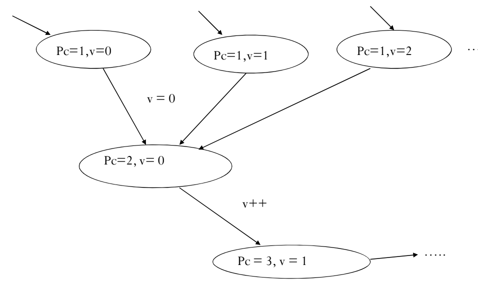
 

## Example

- 1 v = 0;
- 2 v++;
- 3 …
  - What are the states ?
    - (value of pc, value of v)
  - How many initial states are there ?
    - No info, depends on the type of v
- Draw the states and transitions corresponding to this program.

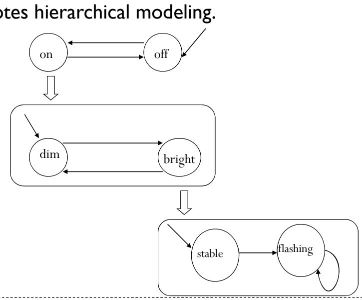
## Example



## Hierarchy

- Choice of steps at different levels of details also promotes hierarchical modeling.



## Basic Concurrent Composition

- $M1 = (S1, I1, \rightarrow_1)$ $M2 = (S2, I2, \rightarrow_2)$
- Define
  - $M1 \times M2 = (S1 \times S2, I1 \times I2, \rightarrow)$
  - Where $(s1,s2) \rightarrow (t1, t2)$ provided
    - $s1 \in S1, t1 \in S1,$
    - $s2 \in S2, t2 \in S2,$
    - $(s1 \rightarrow_1 t1)$ OR $(s2 \rightarrow_2 t2)$
    - Defines control flow of the composed FSM as an arbitrary interleaving of flows from components.
  - *Interleaving of independent flows, what about comm.?*

## Communicating FSM

<u>Basic FSM</u>

- $M = (S, I, \rightarrow)$
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\rightarrow \subseteq S \times S$ is the transition relation.

<u>Communicating FSM</u>

- $M = (S, I, \Sigma, \rightarrow)$
  - S is a finite set of states
  - $I \subseteq S$ is the set of initial states
  - $\Sigma$ is the set of action names that it takes part in
  - $\rightarrow \subseteq S \times \Sigma \times S$ is the transition relation.

Communication across FSMs via action names.

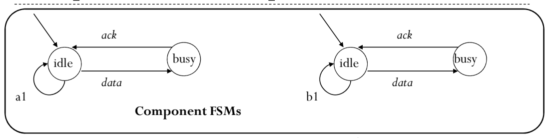
 

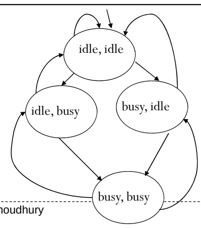
## Composition of comm. FSMs

- $M1 = (S1, I1, \Sigma_1, \rightarrow_1)$ $M2 = (S2, I2, \Sigma_2, \rightarrow_2)$
- Define
  - $M1 \times M2 = (S1 \times S2, I1 \times I2, \Sigma_1 \cup \Sigma_2, \rightarrow)$
  - And $(s1,s2) \xrightarrow{a} (t1,t2)$ provided
    - $s1 \in S1, t1 \in S1,$ and
    - $s2 \in S2, t2 \in S2,$ and
    - If $a \in \Sigma_1 \cap \Sigma_2$ we have $(s1 \xrightarrow{a} t1)$ and $(s2 \xrightarrow{a} t2)$
    - If $a \in \Sigma_1 - \Sigma_2$ we have $(s1 \xrightarrow{a} t1)$
    - If $a \in \Sigma_2 - \Sigma_1$ we have $(s2 \xrightarrow{a} t2)$

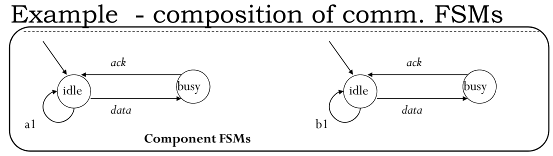
## Example - basic composition

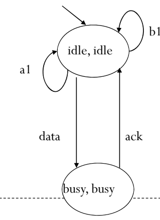


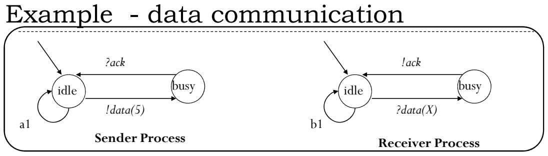
## Example - composition of comm. FSMs

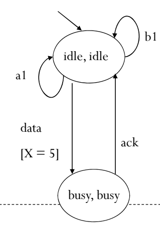


## Example - data communication



## Example: Concurrent Program

P0 || P1

- l0: while true do
- l1: wait(turn = 0);
- l2: turn := 1;
- l3: endwhile

- m0: while true do
- m1: wait(turn = 1);
- m2: turn := 0;
- m3: endwhile

Models a crude protocol for entry/exit to critical section without modeling the critical section itself.

## Example Concurrent Program: States

- Global State = (pc0, pc1, turn)
  - pc0 $\in$ { l0, l1, l2, l3 }
  - pc1 $\in$ { m0, m1, m2, m3 }
  - turn $\in$ { 0, 1 }
- Total = 4 * 4 * 2 = 32 possible states
  - Not all of them might be reachable from the initial states.
  - How many are reachable – try it!

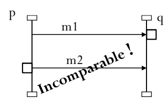
## Wrap-up of FSMs

- FSMs denote an intra-component style of modeling
  - Given a large system – identify its components
  - Model each component as FSM – M1, M2, M3
  - Overall system modeled as concurrent composition
    - M1 || M2 || M3
- Alternate style of modeling
  - Inter-component style
  - Emphasize communication over computation.
  - Sequence Diagrams are basic snippets for describing communication.

## MSC based Models

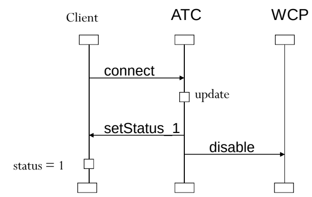
- MSC = Message Sequence Chart
- Labeled partial order of events
  - Highlights inter-process communications
  - While, FSMs highlight intra-process control flow.






## Conventional use of MSCs

- Describe sample scenarios of system interaction
  - Appears in requirement documents
  - Do not describe "complete" system behavior

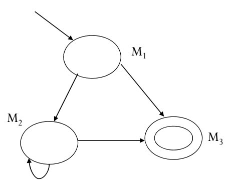



Sample MSC from ATC example

**Exercise:** Find two incomparable events in this MSC

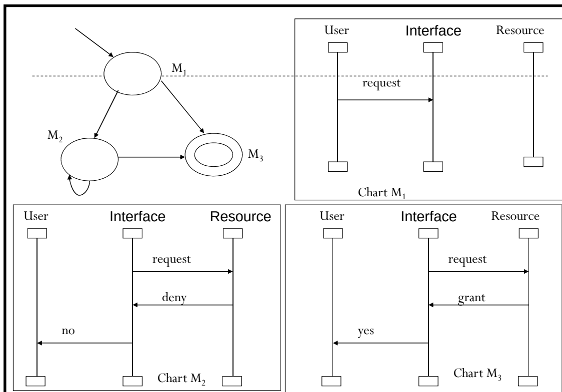


## MSC-based design model



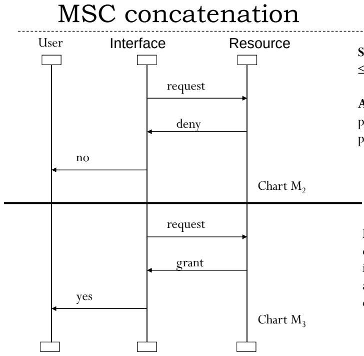

Connect MSCs into a graph – Message Sequence Graph (MSG)
Each node of the graph is a MSC.
Need to define the meaning of concatenation of MSCs








## MSC concatenation




**Synchronous:** All events in M2 ≤ All events in M3

**Asynchronous**: All events in process p of M2 ≤ All events in process p of M3

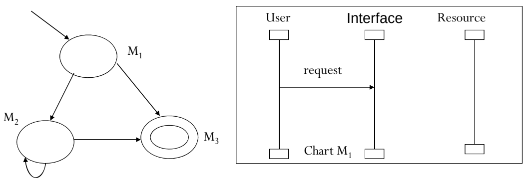
Interface and Resource processes can finish M3 while User process is still in M2 – provided asynchronous concatenation is considered.



## MSC-based design model?

- Complete
  - Complete description of system behavior.
  - MSG achieves this criterion.
- Based on well-established modeling notations.
  - We use UML Sequence Diagrams, which is OK.
- Preferably executable
  - Can simulate the model, and get a feel for how the constructed system will behave!
  - Global simulation of MSG is possible.
  - But not per-process execution !!

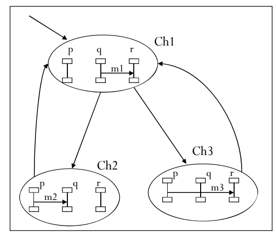
## Why not executable?




At the end of M1, all the processes agree together to execute either M2 or M3.
One process may go ahead of the others (under asynchronous concatenation).
However, the **decision** of which MSC to execute next must be consistent.
Difficult to generate per-process code to capture this **joint decision**.

## Example MSG

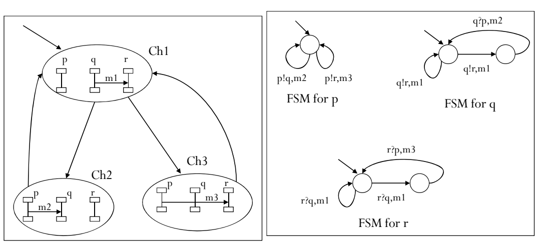



Generates behavior of the form

**(Ch1 o (Ch2 + Ch3))$^{\omega}$**

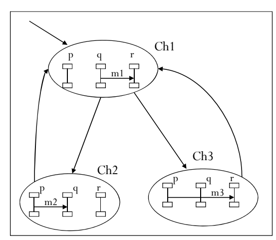
 

## Per-process FSMs



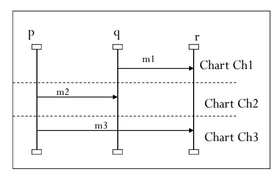

 

## Implied Scenario




Supposed to generates behavior of the form

**(Ch1 o (Ch2 + Ch3))$^{\omega}$**

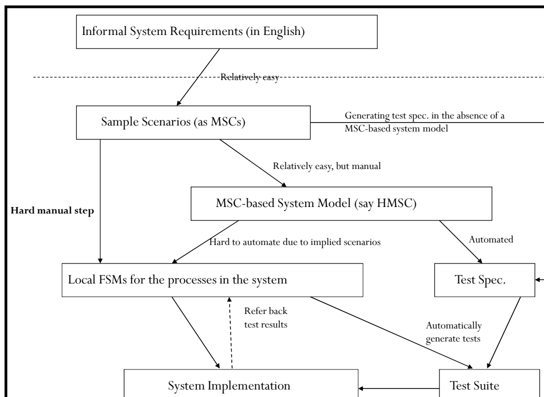


## Putting the notations together

- So, far we have studied 2 notational styles
  - Intra-process style FSM modeling notations
  - Inter-process style MSC-based modeling notation.
- In actual system modeling from English requirements
  - How do they fit together?
  - What roles do they play?
  - Are they both used in parallel?



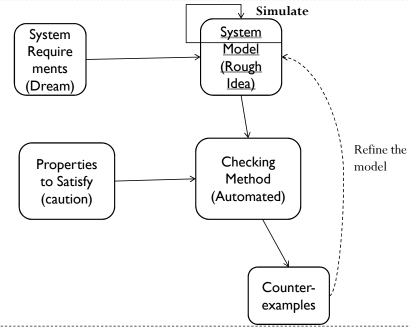

 

## Organization

- So Far
  - What is a Model?
  - ATC – Running Example
    - Informal Req. at a lab scale.
    - Has subtle deadlock error (see textbook chap 2.3)
  - How to model such requirements
    - Modeling Notations
      - Finite State Machines
      - MSC based models
- Now, how to validate the models
  - Simulations

## The big picture - recapitulate



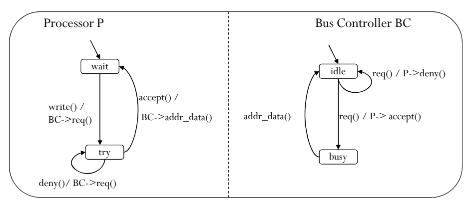

 

## FSM Simulations

- Monolithic FSM simulation
  - A random walk through the FSM's graph.
- Simulating a composition of FSMs
  - Need to consider the definition of concurrent composition.
  - Keep track of local states of the individual processes.
- Simulating more complex notations
  - UML State Diagrams
  - MSC-based models

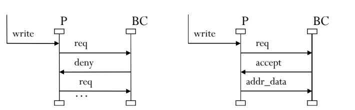
## Example – State Diagrams




Processor and Bus Controller – what does the example do?

## This is what the example does



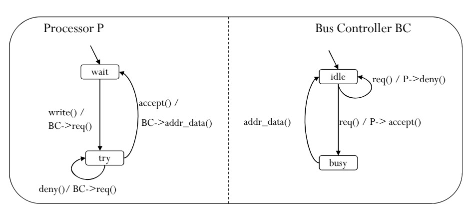

Sample scenarios of the State Diagram shown in the previous slide.

**Super-step:**
On encountering a write, the sequence of method calls executed is
write, req, (deny, req)*, accept, addr_data

**How?**

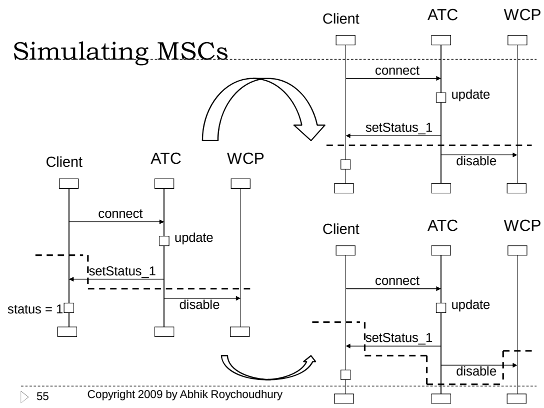
## Simulation – State Diagrams




req(), deny(), req(), accept(), addr_data()

## Simulating MSCs



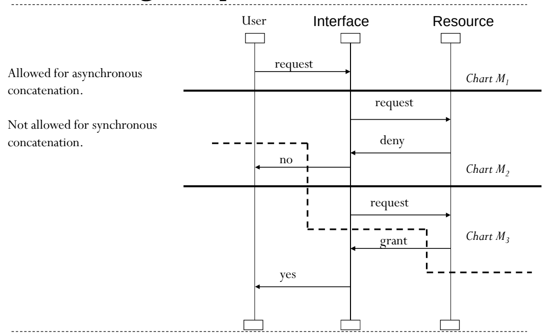

 

## Recap on MSC semantics

- For a sequence of MSCs --- M1, M2
  - **Synchronous concatenation:** All events in M1 $\leq$ All events in M2
  - **Asynchronous concatenation**: All events in process p of M1 $\leq$ All events in process p of M2
- For any msg. m sent from process p to process q
  - **Synchronous message passing**: Send and receive happens in the form of a hand-shake.
  - **Asynchronous message passing**: Sender sends message which is stored in a queue, picked up by receiver later.
- Simulating a sequence of MSCs will need to follow the concatenation & message passing semantics.

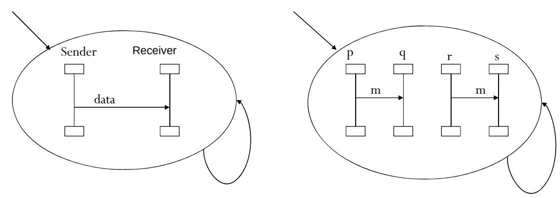
 

## Simulating a sequence of MSCs



## Simulation requires unbounded memory?




Simulation requires unbounded memory under asynchronous concatenation and asynchronous message passing

Simulation requires unbounded memory under asynchronous concatenation and synchronous / asynchronous message passing.

## Avoiding unbounded memory

- Spot Exercise:
  - How can we avoid spending unbounded memory while simulating Message Sequence Graphs ?

## **In the next lecture**

- So Far
  - What is a Model?
  - ATC – Running Example
  - How to model such requirements
- How to validate the models
  - So far: Simulations
  - In the next lecture
    - : Model-based testing