

# Embedded Systems



**Arijit Mondal**

Dept. of Computer Science & Engineering  
Indian Institute of Technology Patna

[arijit@iitp.ac.in](mailto:arijit@iitp.ac.in)

# Composition of State Machines

# Introduction

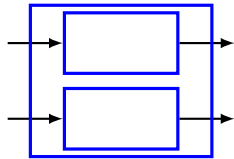
- A system can have large number of states
- Dividing large state machine into smaller can ease modeling
- Multiple state machines interact with each other to realize bigger functionality
- Known as concurrent composition
- Hybrid systems are example of sequential composition

# Concurrent composition

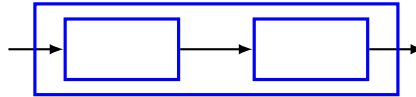
- Two or more state machines react either simultaneously or independently
- Synchronous composition — if the reaction occurs simultaneously
- Asynchronous composition — if the reaction occurs independently
- Need to define semantics of such composition

# Spatial composition

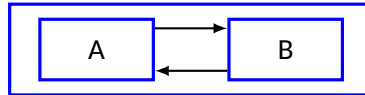
- Side-by-side composition
- Cascade composition
- Feedback composition



side-by-side

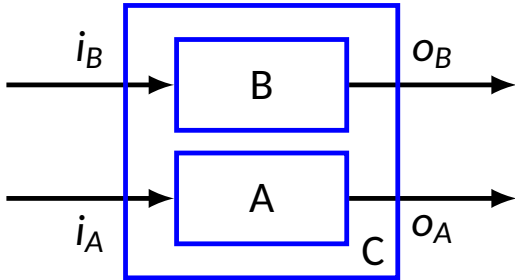


cascade



feedback

# Side-by-side synchronous composition



# Synchronous composition

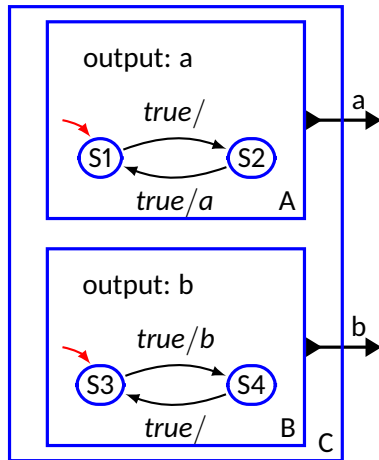
- A reaction in the composition system is a simultaneous reaction of its constituents
- Recall, environment decides when state machine reacts
- Synchronous side-by-side composition is easy
- Such compositions are modular
- If A and B are deterministic, then so is synchronous side-by-side composition
- Compositional — if a property held by the components is also a property of the composition
  - **Determinism is compositional**

# Mathematical description

- Let  $SM_A = \langle S_A, I_A, O_A, update_A, S_A^0 \rangle$
- Let  $SM_B = \langle S_B, I_B, O_B, update_B, S_B^0 \rangle$ 
  - $S$  - states,  $I$  - inputs,  $O$  - outputs,  $S^0$  - initial state
- Composition machine  $C$  will have
  - $S_C = S_A \times S_B$
  - $I_C = I_A \times I_B$
  - $O_C = O_A \times O_B$
  - $S_C^0 = S_A^0 \times S_B^0$
  - $update_C((s_A, s_B), (i_A, i_B)) = ((s'_A, s'_B), (o_A, o_B))$  where
    - $(s'_A, o_A) = update_A(s_A, i_A)$  and
    - $(s'_B, o_B) = update_B(s_B, i_B)$



# Side-by-side synchronous composition: example



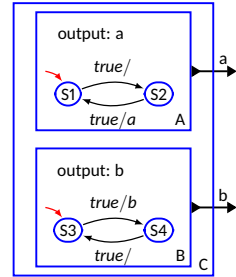
# Side-by-side synchronous composition: example

S1,S3

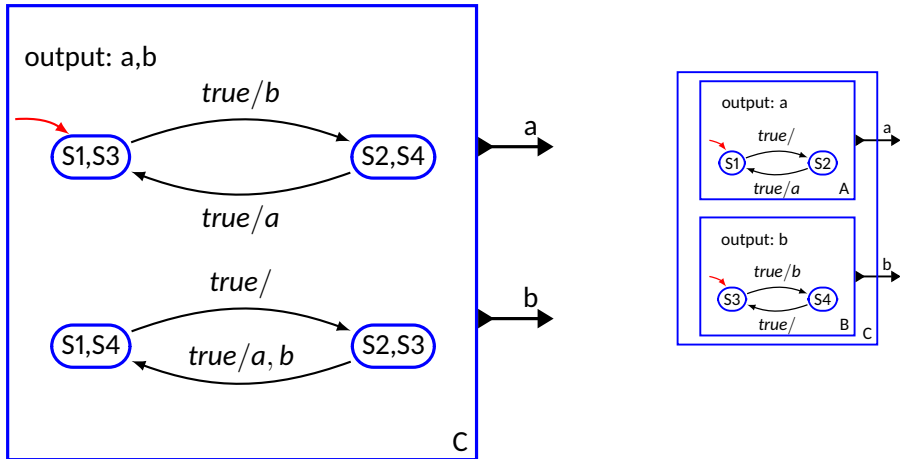
S2,S4

S1,S4

S2,S3



# Side-by-side synchronous composition: example



# Side-by-side asynchronous composition

- Component machine reacts independently
- A reaction of composition machine C is a reaction of **one of A or B** where the choice is nondeterministic
  - **Interleaving in nature as A or B never react simultaneously**
- A reaction of composition machine C is a reaction of **A or B or both A and B** where the choice is nondeterministic
- Composition machine C will have
  - $update_C((s_A, s_B), (i_A, i_B)) = ((s'_A, s'_B), (o'_A, o'_B))$  **where**  
 $(s'_A, o'_A) = update_A(s_A, i_A)$  **and**  $s'_B = s_B$  **and**  $o'_B = absent$   
**OR**  
 $(s'_B, o'_B) = update_B(s_B, i_B)$  **and**  $s'_A = s_A$  **and**  $o'_A = absent$

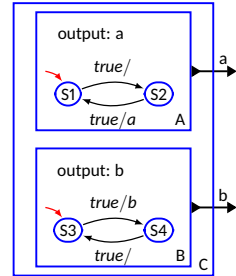
# Side-by-side asynchronous composition:example

S1,S3

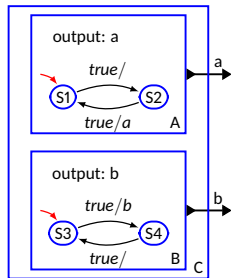
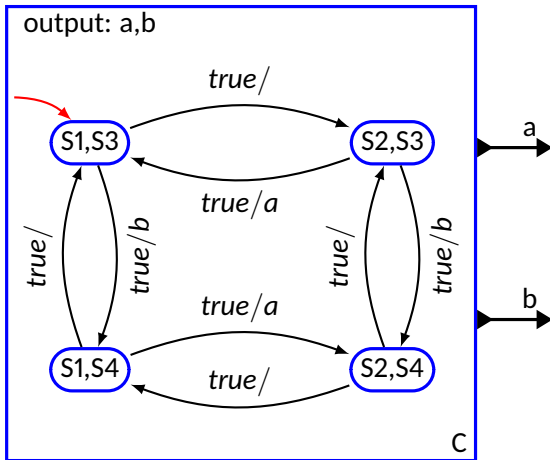
S2,S3

S1,S4

S2,S4



# Side-by-side asynchronous composition: example



# Composition using shared variable

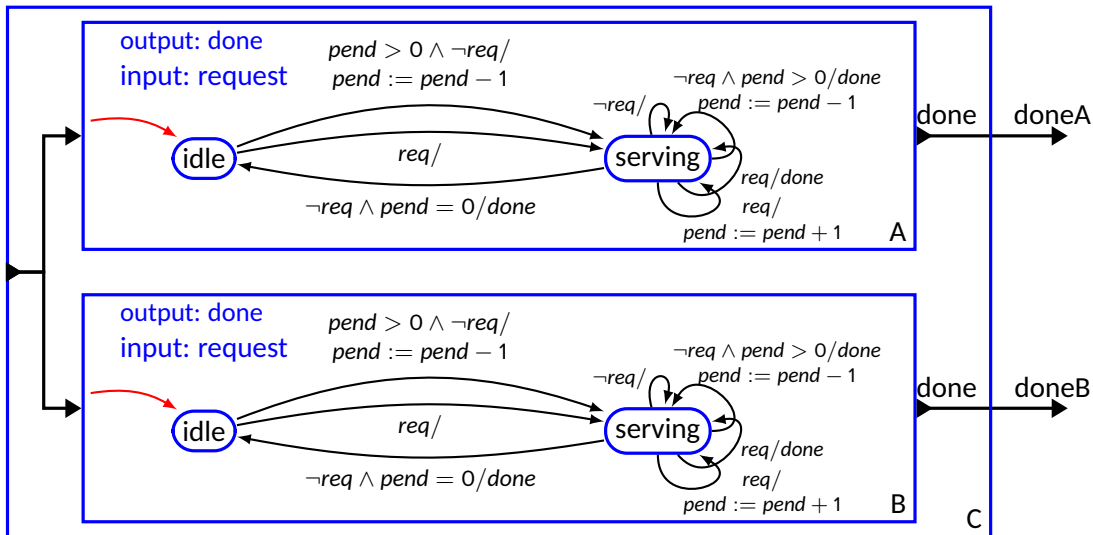
- Extended state machine has local variables
- Sometime it is useful when composing state machines allow these variables shared
  - Useful for modeling interrupts, threads, etc.
- Atomic operation

# Composition using shared variable: example

- Two servers receive request
- Each request requires unknown amount of time of service
- Server shares queue of request
- If one server is busy, the other server can respond to a request, even if the request arrives at the network interface of the first server

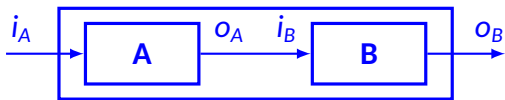


# Server queue



# Cascade composition

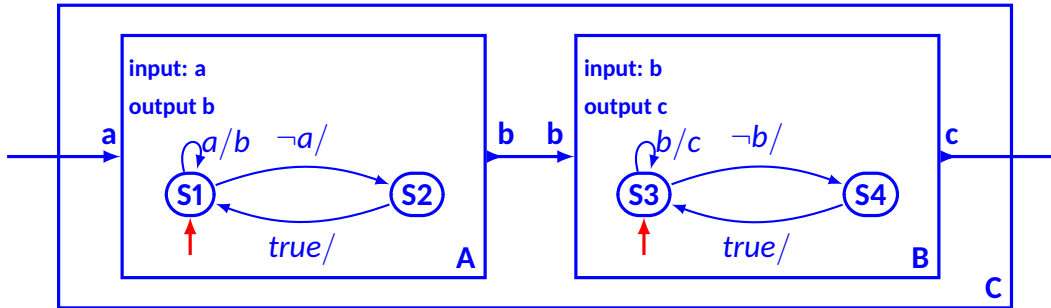
- Output of one of the machines feeds into input of the other



- Data type must conform
- Synchronous composition

# Cascade composition: Example

- Output of one of the machines feeds into input of the other



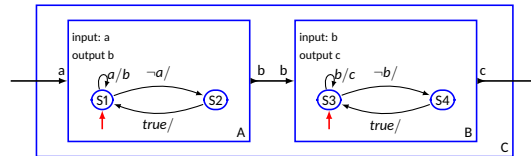
# Cascade composition:synchronous

S1,S3

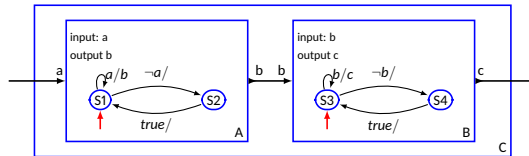
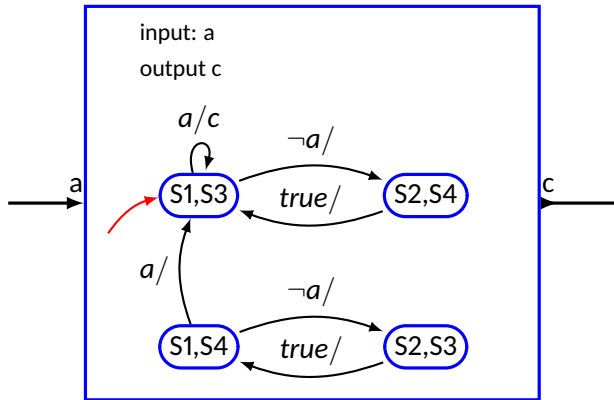
S2,S4

S1,S4

S2,S3



# Cascade composition:synchronous

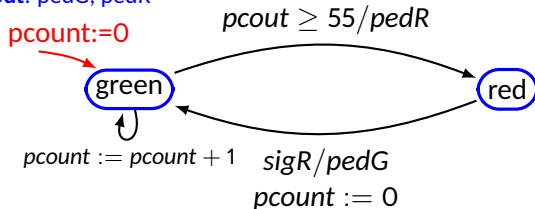


# Cascade composition: Traffic light

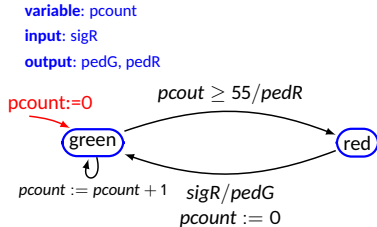
variable: pcount

input: sigR

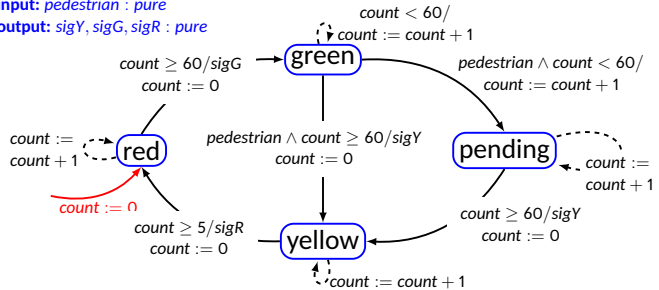
output: pedG, pedR



# Cascade composition: Traffic light



variable: count : {0, 1, ..., M}  
input: pedestrian : pure  
output: sigY, sigG, sigR : pure

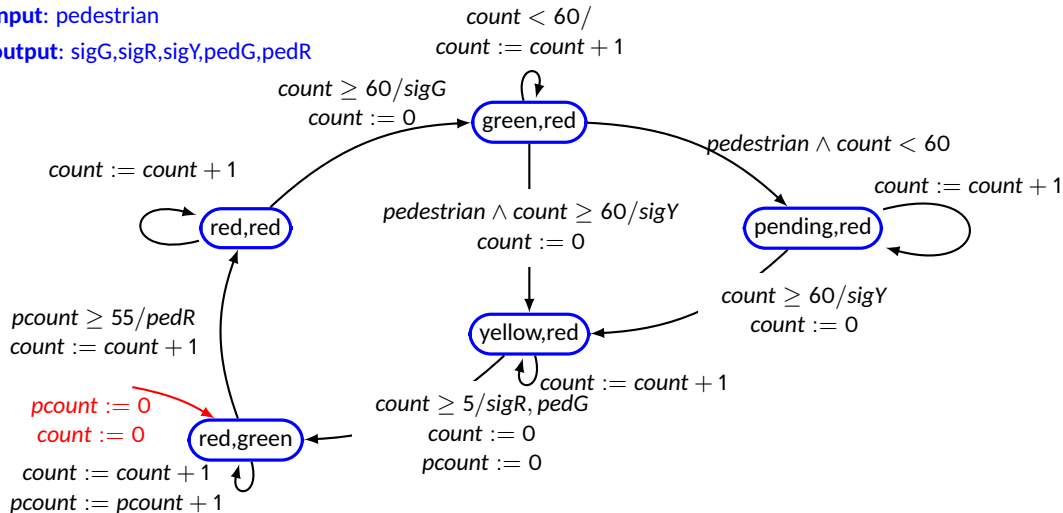


# Cascade composition: Traffic light

variable: pcount, count

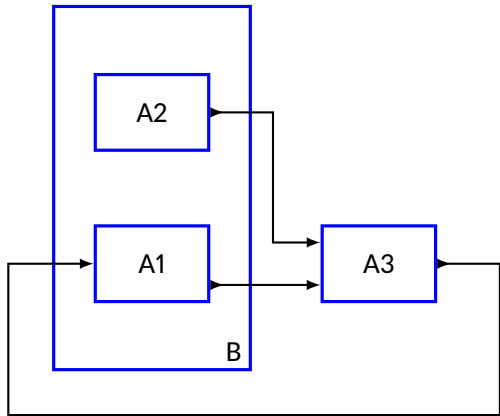
input: pedestrian

output: sigG, sigR, sigY, pedG, pedR

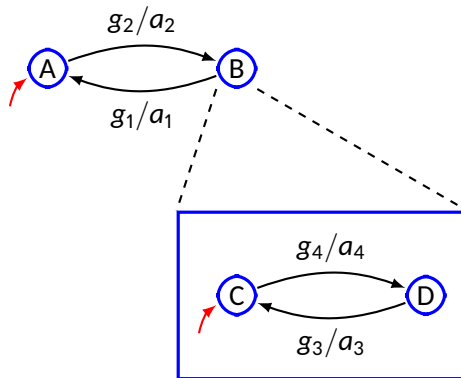




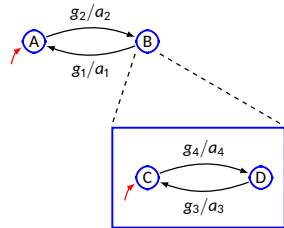
# General composition



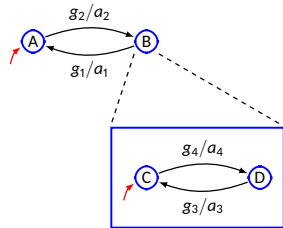
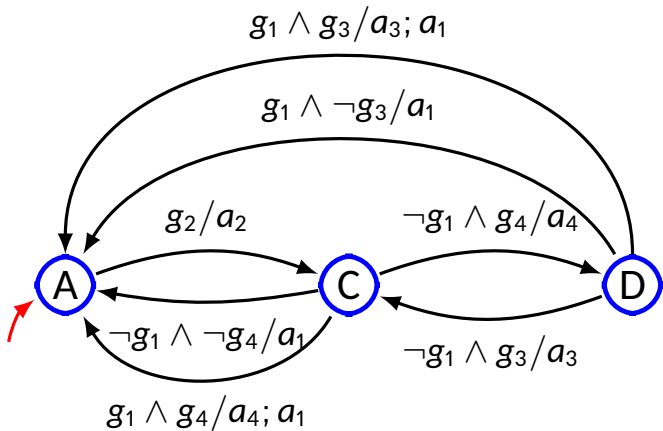
# Hierarchical SM



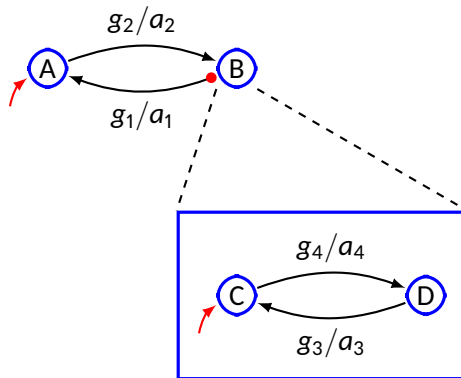
# Hierarchical SM



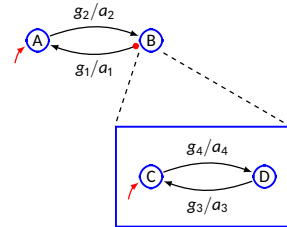
# Hierarchical SM



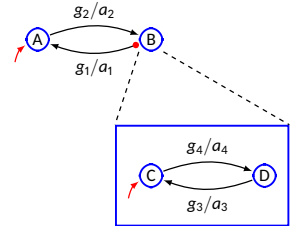
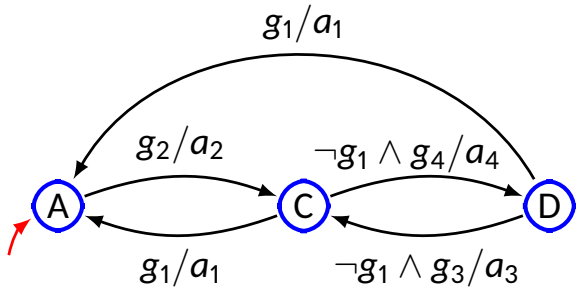
# Hierarchical SM: preemptive transition



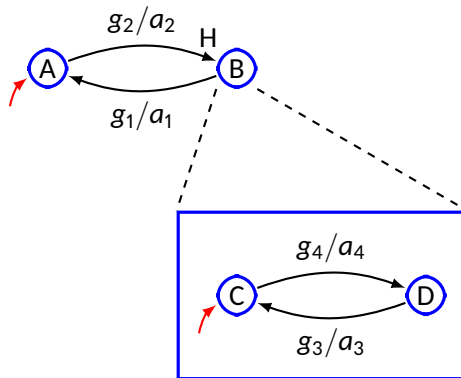
# Hierarchical SM: preemptive transition



# Hierarchical SM: preemptive transition

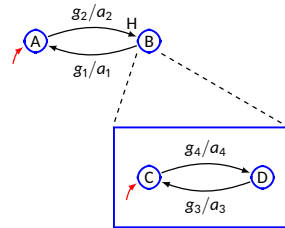
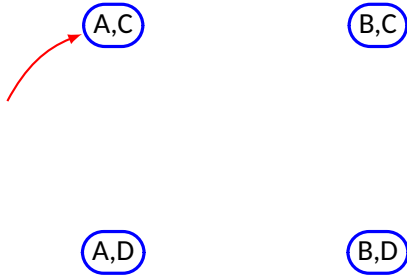


# Hierarchical SM: history transition

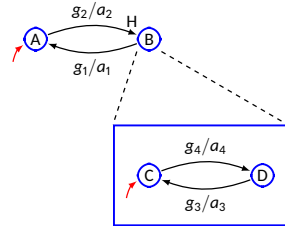
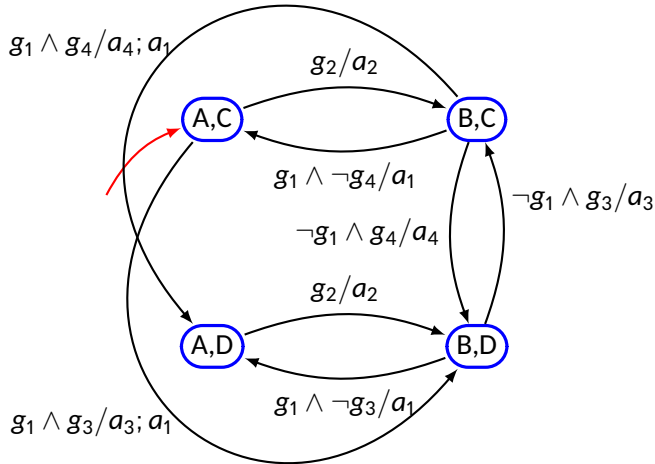




# Hierarchical SM: history transition



# Hierarchical SM: history transition



# Summary

- A big system may be developed by composing simple / smaller components
- We have seen two forms of composition - concurrent and hierarchical
- Concurrent composition - asynchronous and synchronous semantics
- Hierarchical composition - refinement semantics