Control and communication of distributed hybrid systems

ACC 2001 Tutorial

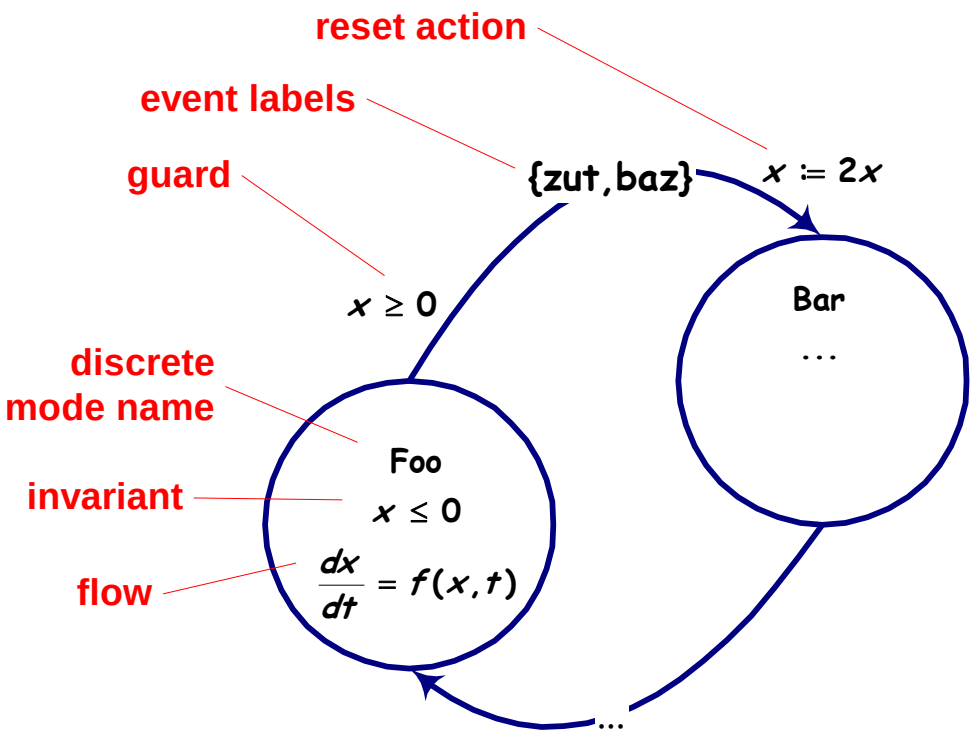
July 27th, 2001

Tunc Simsek

Joao Sousa

Pravin Varaiya

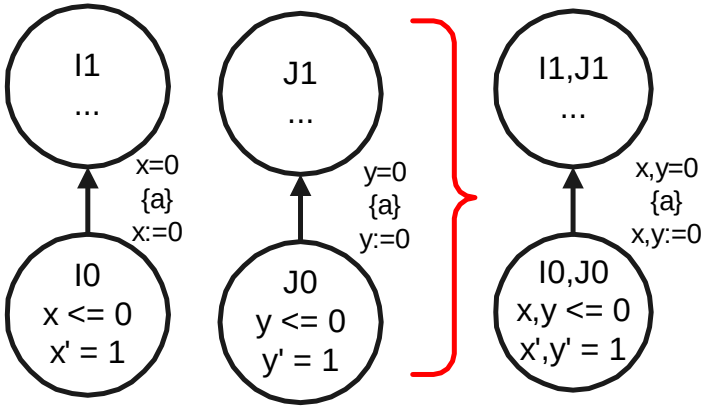
Hybrid automata



- Labeled state machine
- Hybrid state

  $(x, s) \in \Re \times \{Foo, Bar\}$
- Discrete modes model the distinct behaviors of the continuous time plant/controller
- Transitions model mode switches and discrete control actions

Hybrid Automata - composition



Synchronous composition

Input/output composition

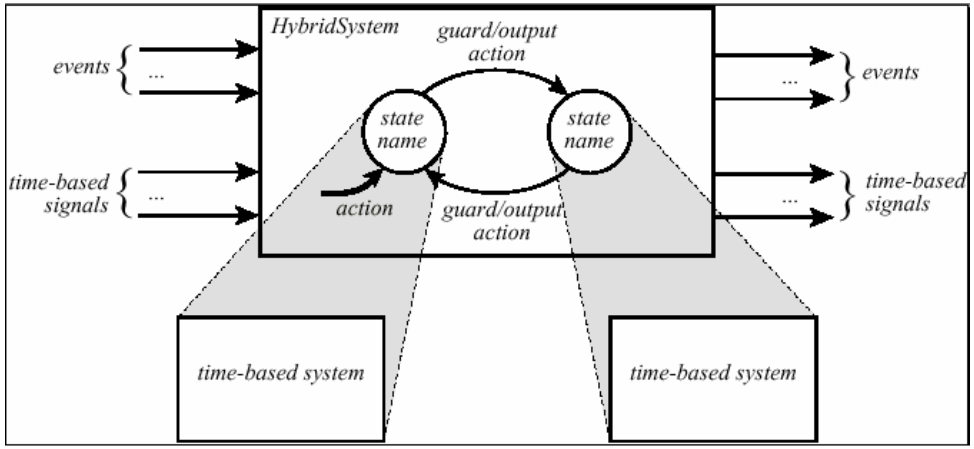
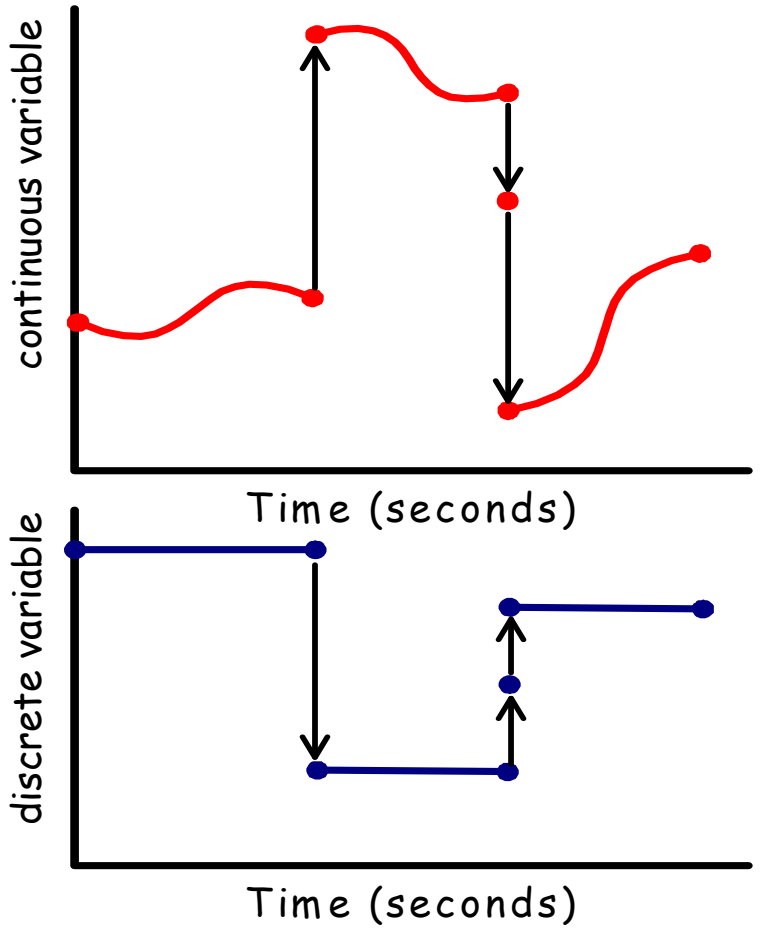


Image from E.A. Lee, P. Varaiya, Structure and Interpretation of Signals and Systems, 2000. (available from http://ptolemy.eecs.berkeley.edu/eecs20/)

Hybrid Automata - Execution traces

- **Evolution in alternating phases of discrete transitions and time passage**
- **Multiple discrete transitions**



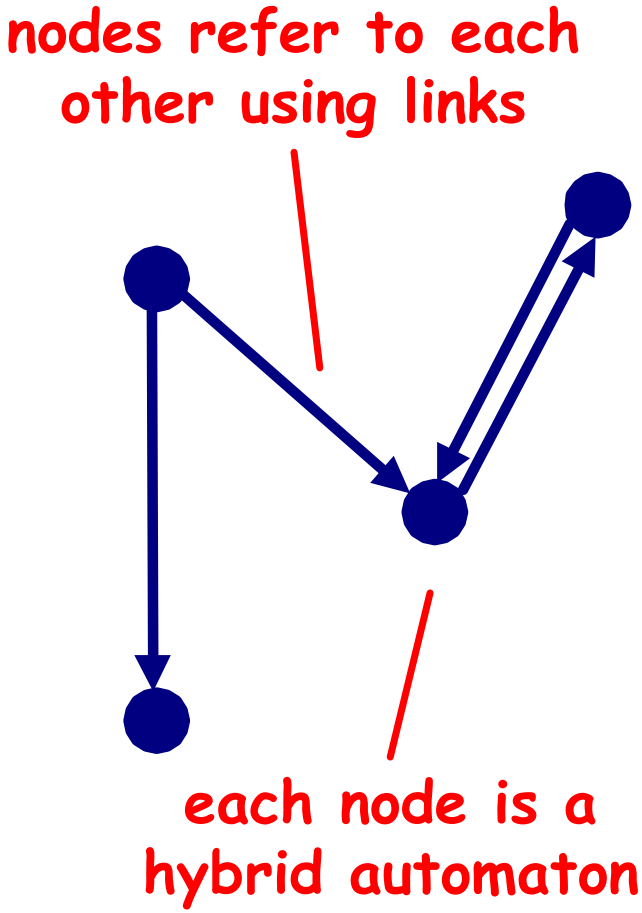
Models

A model of the operation of distributed dynamic systems should capture two essential features

- Switched mode operation
- Dynamic interactions

Click here to view Quicktime movie

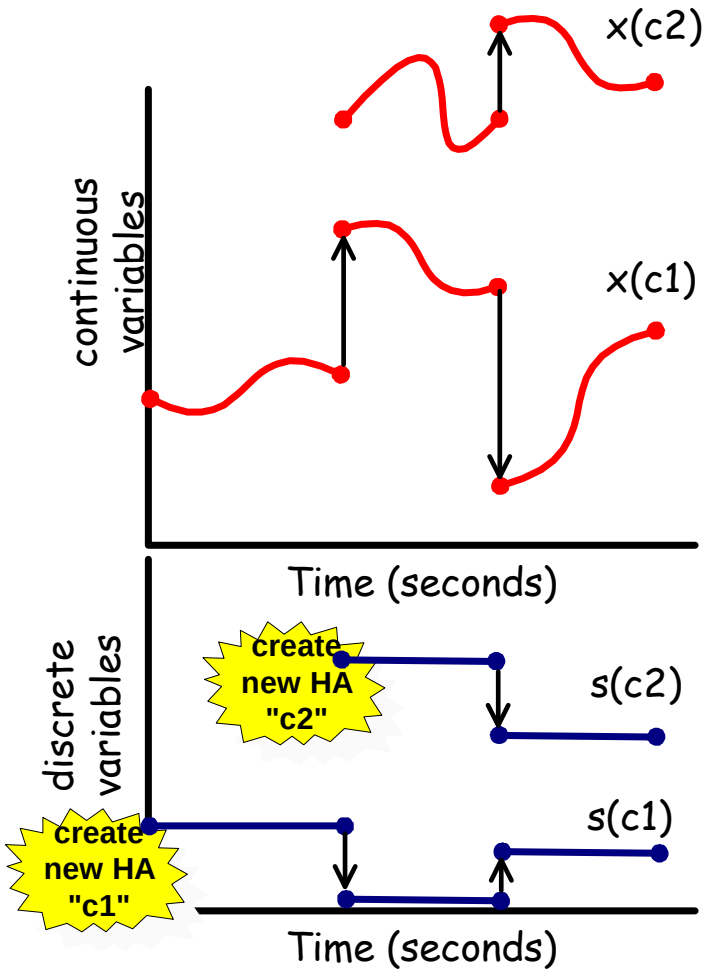
Dynamic networks of hybrid automata (DNHA)



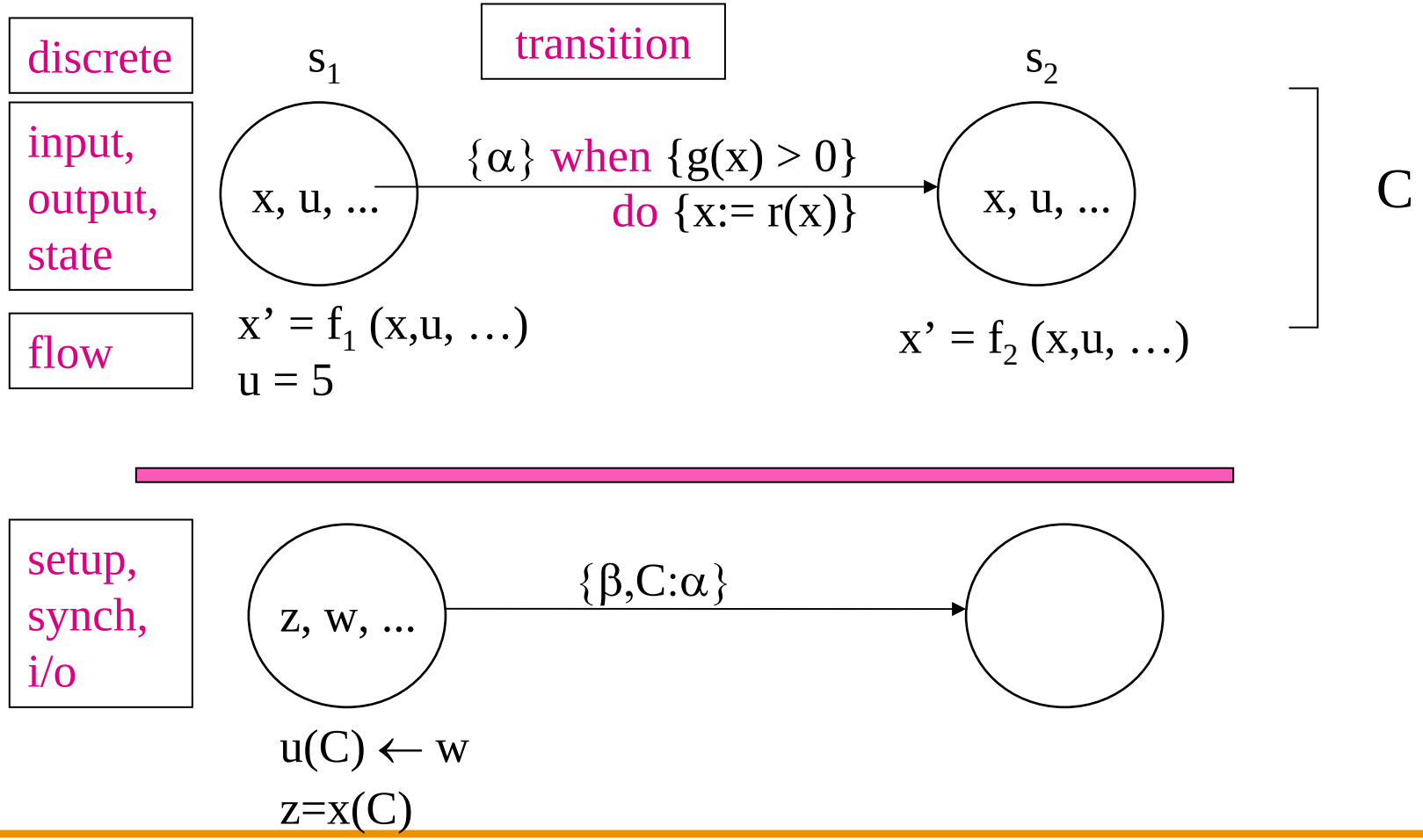
- A network of hybrid automata that evolve in parallel
- Network is dynamic because nodes may be created and destroyed as system evolves

DNHA - execution traces

- **Evolution in alternating phases of discrete transitions and time passage**
- **State vectors reconfigured on discrete transitions**



DNHA: SHIFT visual syntax



SHIFT syntax

- **Prototype-based description:**

```
type Component  {
    input …       what we feed to it
    output …      what we see on the outside
    state …       what's internal
    discrete …    discrete symbolic actions
    export …      event (transition) labels
    setup …       actions executed at create time
    flow …
    transition …
}
```

data model

continuous and discrete evolution

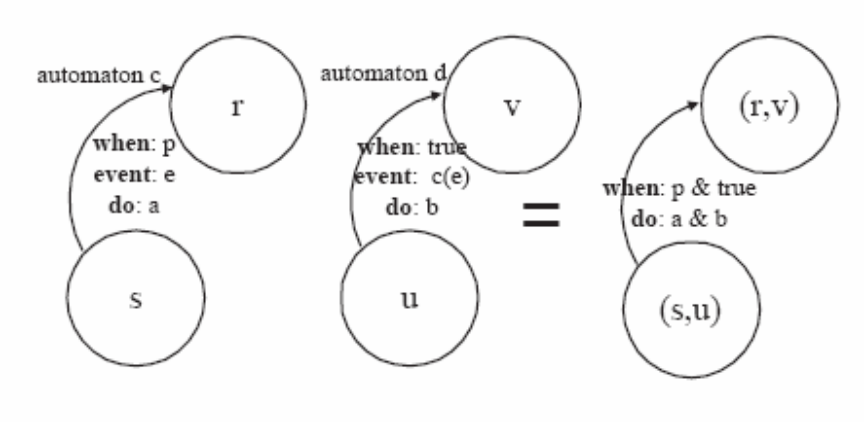
- **Prototypes are instantiated:**

```
create(Component)
```

SHIFT models

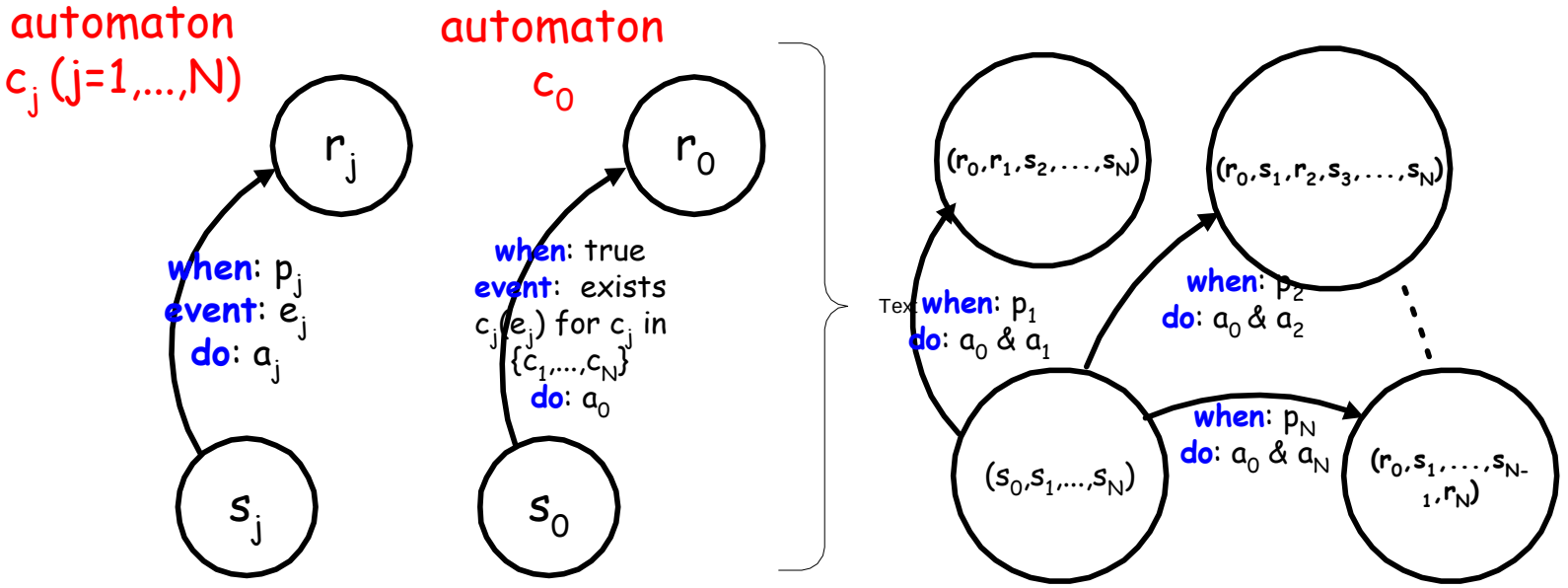
- **Primitives:**
  - **hybrid automata**
    - **encapsulated input/output/state variables**
    - **arbitrarily complex flows**
    - **structured transitions**
  - **strong typing (write syntactically correct programs)**
  - **numbers, sets, arrays (first order predicate calculus)**
- **Combination facilities:**
  - **continuous input/output connections**
  - **synchronous composition**
- **Abstraction facilities:**
  - **prototyping**
  - **inheritance**

SHIFT models: one-to-one composition



- **synchronous composition**

SHIFT models: Combination

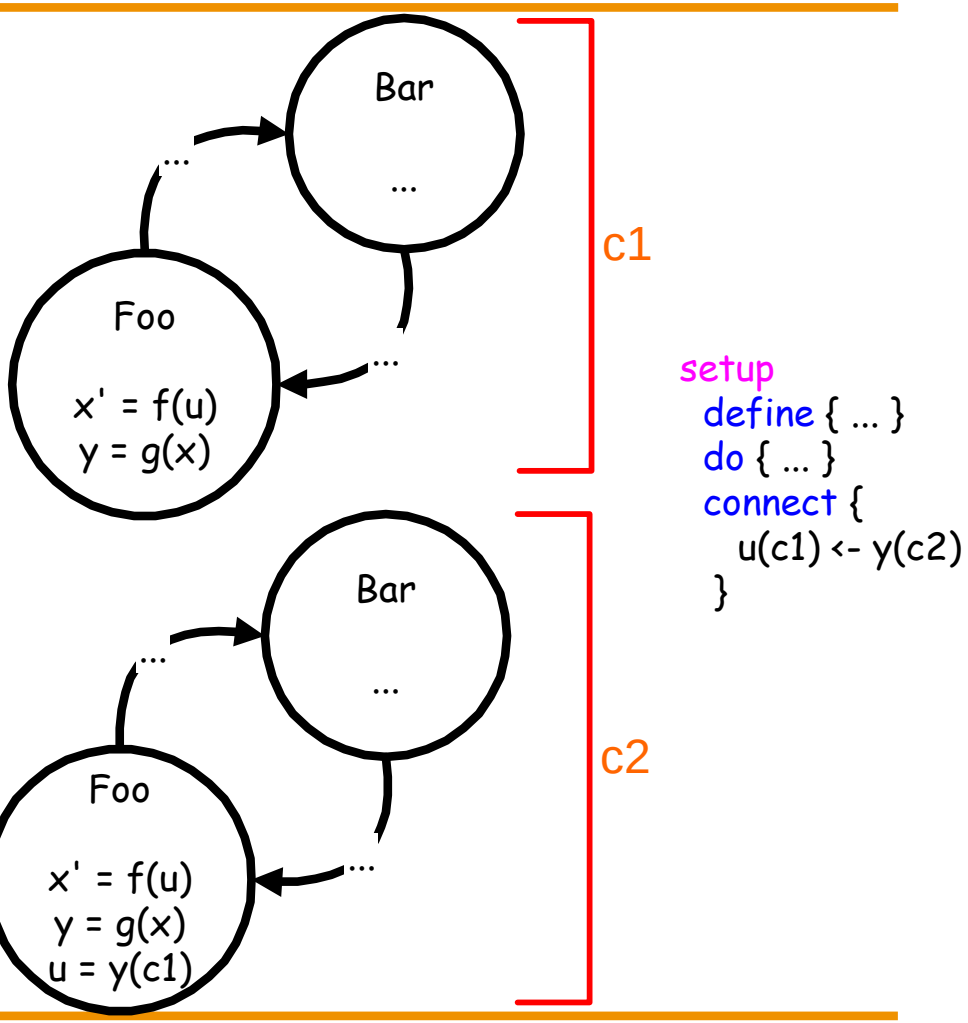


- **synchronous composition**
  - **simple**
  - **existential quantification (illustrated above)**
  - **universal quantification**
- **existentially quantified automata may be used in actions**

SHIFT models: Combination

input/output composition

- **algebraic readouts through links: u=y(c1)**
- **explicit connections**



SHIFT models: Inheritance

First define a general interface:

```
type VehicleDynamics {
        output continuous number xDot, yDot, zDot;
                continuous number wx, wy, wz;
}
```

SHIFT models: Inheritance

Specialize generic interface:

```
type k_vehicle_dynamics : VehicleDynamics {
        input      continuous number xDDot, yDotIn;
                    number xDotInit;
        output     number vehicle_length := 0;
        export     exiting, stopping, running;
        discrete   cruise, stopped, exit2;
        flow
          default {   xDot' = xDDot;   yDot = yDotIn;  };
        transition
          cruise -> stopped {stopping} do {xDot := 0;},
          stopped -> cruise {running},
          cruise -> exit2 {exiting};
}
```