[dcc]

# CONCURRENT MODELS OF COMPUTATION

Pedro Brandão, Sérgio Crisóstomo
Sistemas Embutidos 2020/21

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

1

[dcc]

## References

- Slides are from Edward A. Lee & Sanjit Seshia, UC Berkeley, EECS 149 Fall 2013
  - *Copyright © 2008-date, Edward A. Lee & Sanjit A. Seshia, All rights reserved*
- Petri networks' slides are from Reinhard von Hanxleden, Christian-Albrechts-Universität zu Kiel
- Time triggered slides are from Thomas A. Henzinger, IST Austria

U.PORTO



2

[dcc ]

# STRUCTURE OF MODELS

U.PORTO



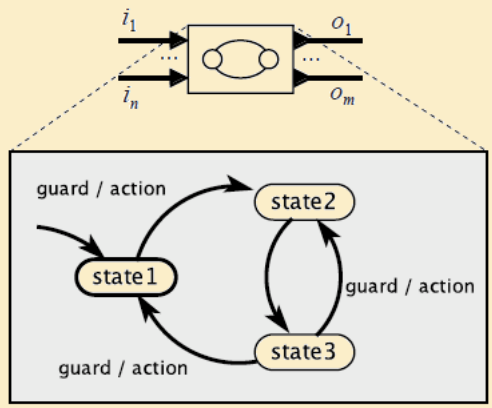
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

3

[dcc]

## Recall: Actor Model for State Machines

- Expose inputs and outputs, enabling composition:




U.PORTO

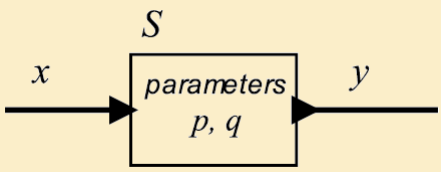
4



4

## Recall: Actor Model of Continuous-Time Systems

- A system is a function that accepts an input signal and yields an output signal.
- The domain and range of the system function are sets of signals, which themselves are functions.
- Parameters may affect the definition of the function S.



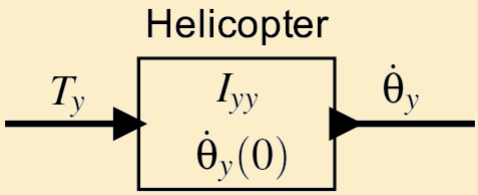

$x: \mathbb{R} \to \mathbb{R}, \qquad y: \mathbb{R} \to \mathbb{R}$

$$S: X \to Y$$
$$X = Y = \mathbb{R} \to \mathbb{R}$$



## Example: Actor model of the helicopter

- Input is the net torque of the tail rotor and the top rotor. Output is the angular velocity around the y axis.

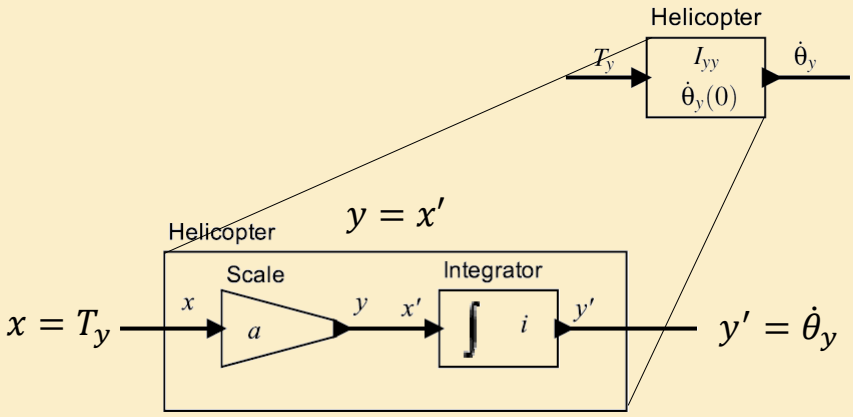



Parameters of the model are shown in the box. The input and output relation is given by the equation.

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

[dcc]

# Recall: Composition of actor models




U.PORTO

7





[dcc]

# Models of Computation (MoC)

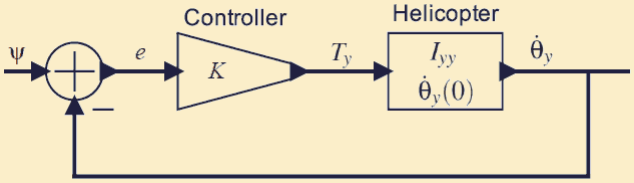
- A model of computation *assigns semantics* (meaning) to the syntax defined in the actor model.
- The MoC gives *operational rules* for executing a model. These rules determine when actors perform internal computation, update their internal state, and perform external communication.
- An MoC also defines the *nature of communication* between components (e.g., buffered I/O).

U.PORTO

8

[dcc]

# Model of Computation (MoC): Continuous Time (CT)




- Structure of a signal:

$$s: T \rightarrow R$$

- Where in our helicopter model: $T = R = \mathbb{R}$
  - *But signals can haver a rather different form*

U.PORTO

9





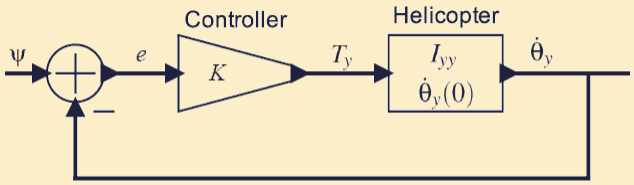
[dcc]

# Discrete-Time (DT) Actor Models

- Discrete time signals have the form:

$$x: \mathbb{Z} \rightarrow R$$

- For some data type $R$, where $\mathbb{Z}$ is the set of integers
- An index $n \in \mathbb{Z}$ is typically associated with a time value $nT$, where $T \in \mathbb{R}$ is the sampling interval
- Discrete-time helicopter model looks the same:

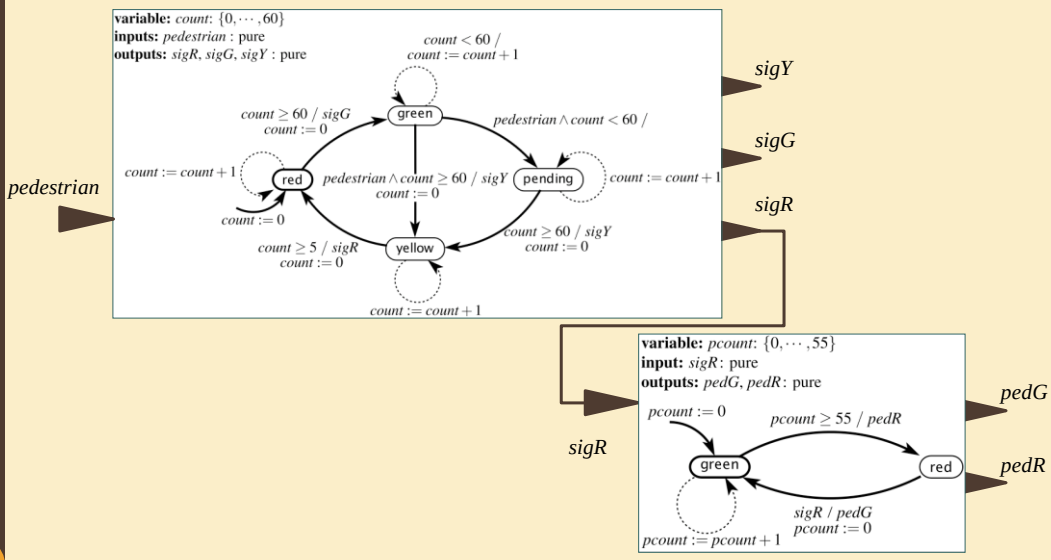



U.PORTO

10

[dcc]

# Time-Triggered Reactions of FSMs: Discrete-Time MoC




U.PORTO

11

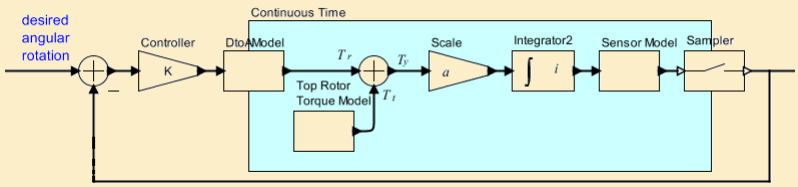


11

[dcc]

# Mixed Signal Models

- A more reasonable model of a discrete helicopter controller would be a mixed-signal model:




- Here, the signals inside the blue area are continuous-time signals, and the ones outside are discrete-time signals.

U.PORTO

12



12

[dcc]

# "absent" values

- To jointly model discrete and continuous-time signals in the same MoC, use the notion of "absent" values
- Let a continuous-time (CT) signal be a function of form

$$x: \mathbb{R} \rightarrow \mathbb{R} \cup \{\varepsilon\}$$

- Where $\varepsilon$ denotes "absent". A discrete-time signal is now modelled as a CT signal whose value is $\varepsilon$ except at times $t \in \mathbb{R}$ where $t = nT$, for some $n \in \mathbb{Z}$
- We can also model discrete-event (DE) systems, where the discrete events need not be regularly spaced.

Event-triggered composition of state machines uses a discrete-event MoC.

U.PORTO

13





[dcc]

# Example

- Consider a pure signal s that is a discrete signal given by:

$$s(t) = \begin{cases} present, & \text{if } t \text{ is a multiple of } P \\ absent, & otherwise \end{cases}$$

  - *for all $t \in \mathbb{R}$ and some $P \in \mathbb{R}$.*
- signal is a clock signal with period $P$
  - *Communication events occur every $P$ time units.*

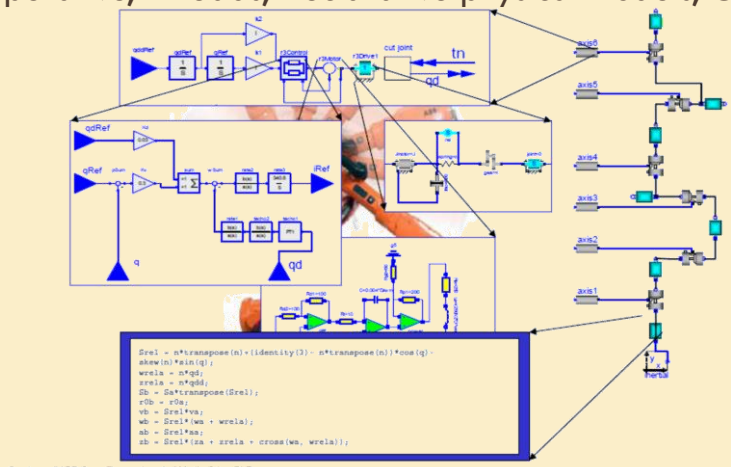
U.PORTO

14

[dcc]

# Other way to do things

- Imperative, Threads, Declarative physical models, Constraints, …




Modelica model of an industrial robot. Modelica uses Spice-like models.

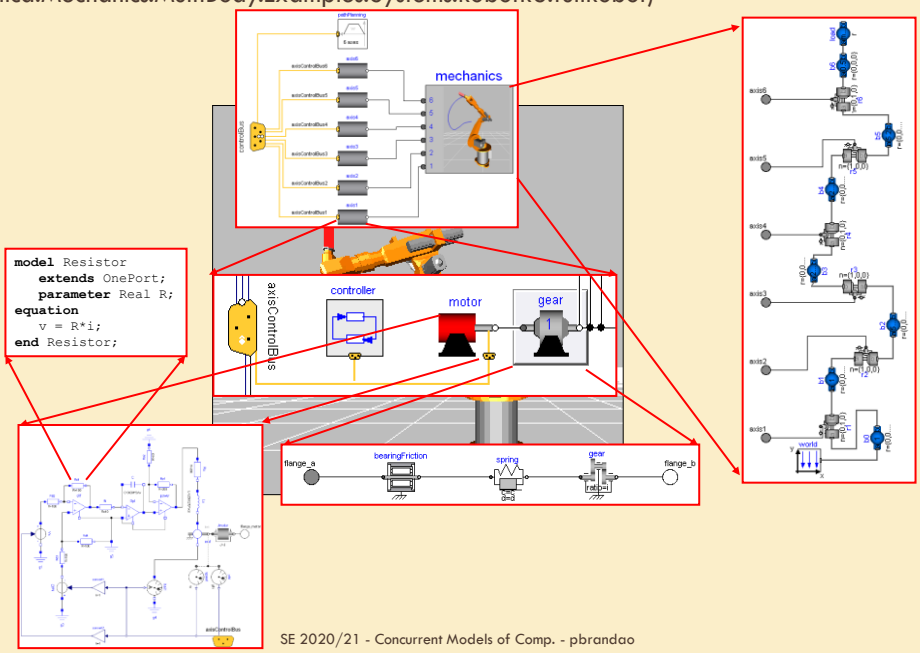
U.PORTO



[dcc]

Example of Modelica: Industrial Robots (from Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot)

From Modelica Overview




U.PORTO

[dcc ]

# SYNCHRONOUS-REACTIVE MODELS

U.PORTO



FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

17

[dcc]

## Concurrent Composition: Alternatives to Threads

- Threads yield incomprehensible behaviours.
- Composition of State Machines:
  - *Side-by-side composition*
  - *Cascade composition*
  - *Feedback composition*
- We will begin with synchronous composition.

U.PORTO

18

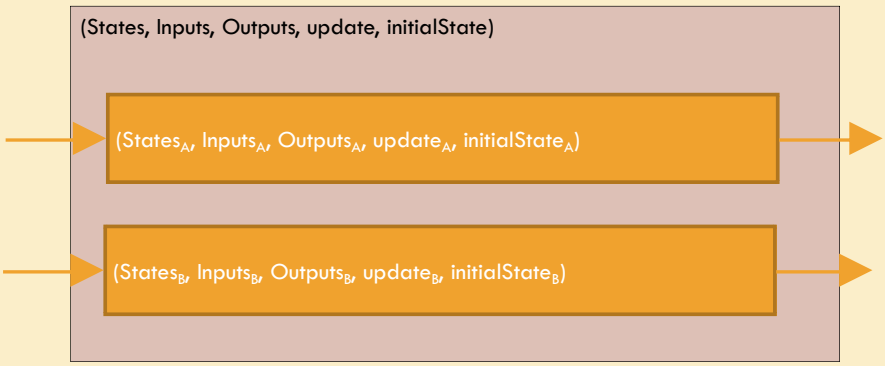


18

[dcc]

# Side-by-Side Composition

- **Synchronous composition**: the machines react simultaneously and instantaneously.



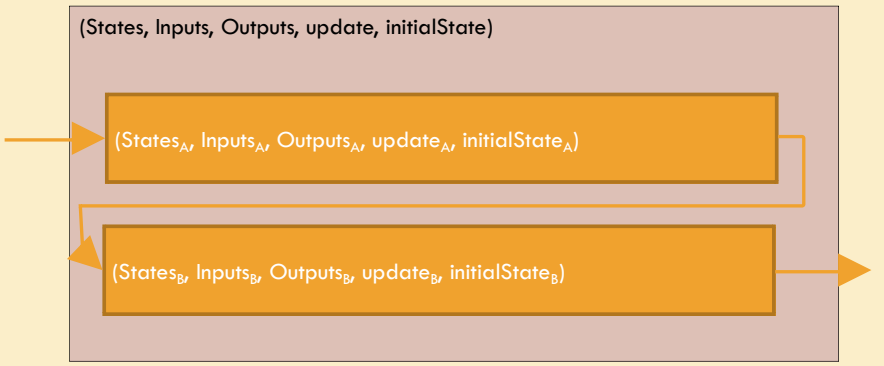

U.PORTO



[dcc]

# Cascade Composition

- **Synchronous composition**: the machines react simultaneously and instantaneously, despite the apparent causal relationship!

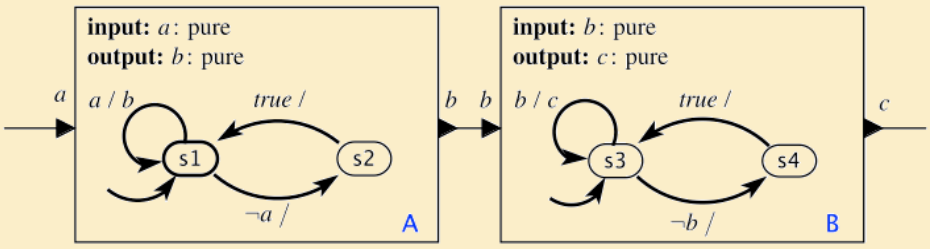



U.PORTO

[dcc]

# Synchronous Composition: Reactions are Simultaneous and Instantaneous

- Consider a cascade composition as follows:




U.PORTO

21



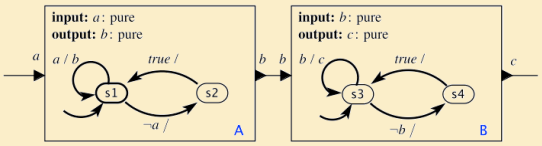


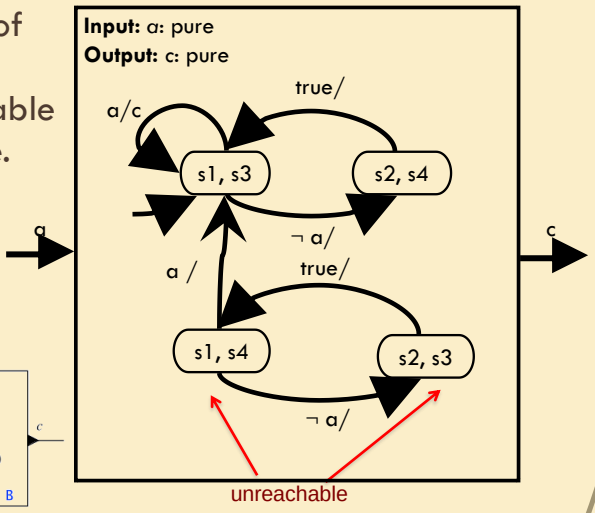
[dcc]

# Synchronous Composition: Reactions are Simultaneous and Instantaneous

- In this model, you must not think of machine A as reacting before machine B. If it did, the unreachable states would not be unreachable.

$$S_C = S_A \times S_B$$

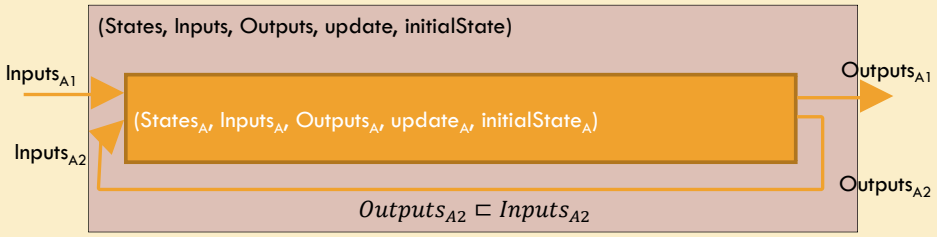



U.PORTO

22

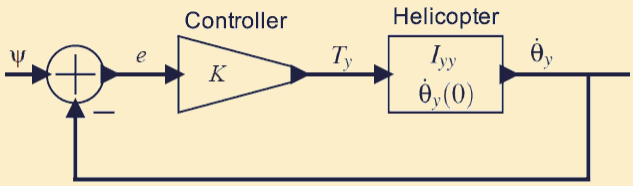
# Feedback Composition




Turns out everything can be viewed as feedback composition…



# Recall: Feedback Composition

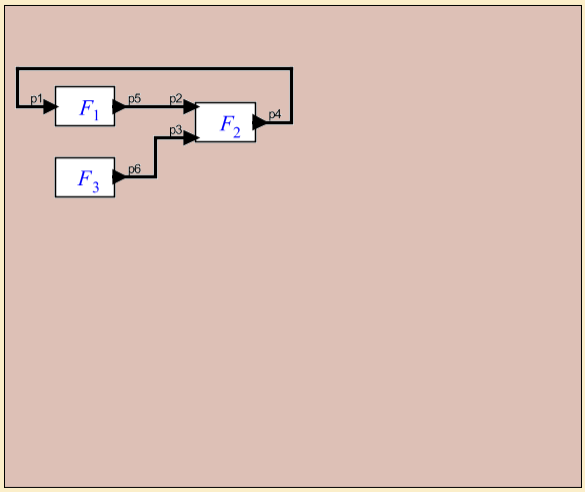



$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$
$$= \dot{\theta}_y(0) + \frac{K}{I_{yy}} \int_0^t \left( \Psi(\tau) - \dot{\theta}_y(\tau) \right) d\tau$$

Angular velocity appears on both sides. The semantics (meaning) of the model is the solution to this equation.

[dcc]

# Observation: Any Composition is a Feedback Composition




If every actor is a function, then the semantics of the overall system is the least $\mathbf{s} \in S^N$ such that $F(\mathbf{s}) = \mathbf{s}$

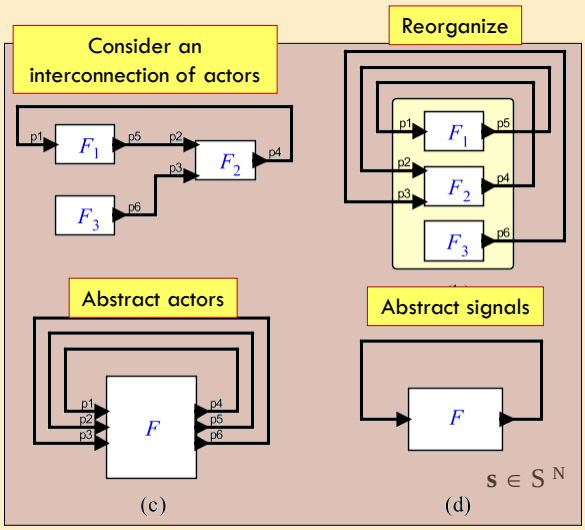
The behaviour of the system is a "fixed point."

U.PORTO



[dcc]

# Fixed Point Semantics




We seek an $\mathbf{s} \in S^N$ that satisfies $F(\mathbf{s}) = \mathbf{s}$.

Such an **s** is called a *fixed point*.

We would like the fixed point to exist and be unique. And we would like a constructive procedure to find it.

It is the *behaviour* of the system.

U.PORTO

[dcc]

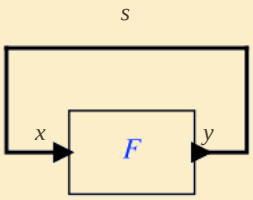
# Data Types

- As with any connection, we require compatible data types:

$$V_y \subseteq V_x$$

- Then the signal on the feedback loop is a function

$$s\colon \mathbb{N} \to V_y \cup \{absent\}$$




- Then we seek $s$ such that

$$F(s) = s$$

- Where $F$ is the actor function, which for determinate systems has form

$$F\colon (\mathbb{N} \to V_x \cup \{absent\}) \to (\mathbb{N} \to V_y \cup \{absent\})$$

U.PORTO

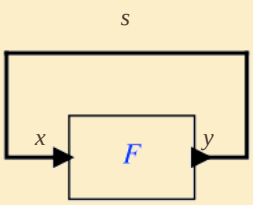


[dcc]

# Firing Functions

- With synchronous composition of determinate state machines, we can break this down by reaction. At the n-th reaction (can be a tick of a clock), there is a (state-dependent) function

$$f(n)\colon V_x \cup \{absent\} \to V_y \cup \{absent\}$$




- Such that

$$s(n) = (f(n))(s(n))$$

- This too is a fixed point

U.PORTO

[dcc] U.PORTO

# Well-Formed Feedback

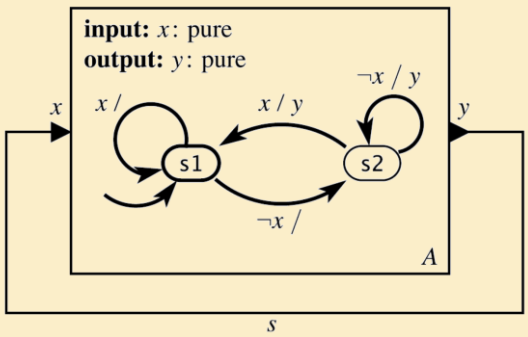
- At the n[th] reaction we seek $s(n) \in V_y \cup \{absent\}$ such that
$$s(n) = \big(f(n)\big)\big(s(n)\big)$$
- There are two potential problems:
  1. *It does not exist*
  2. *It is not unique*
- In either case, the system is ill formed.
  - *Otherwise, it is well formed*
- if a state is not reachable, it is irrelevant to determining whether ill or well formed



[dcc] U.PORTO

# Well-Formed Example

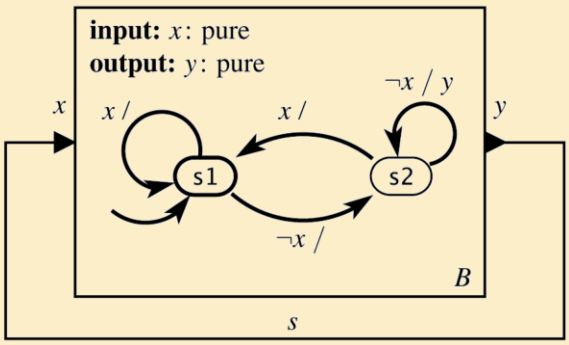
- In state s1, we get the unique fixed point $s(n) = absent$
- In state s2, we get the unique fixed point $s(n) = present$
- Therefore, s alternates between absent and present.



[dcc]

# Ill-Formed Example 1 (Existence)

- In state s1, we get the unique $s(n) = absent$
- In state s2, there is no fixed point
- Since state s2 is reachable, this composition is ill formed.




U.PORTO

31

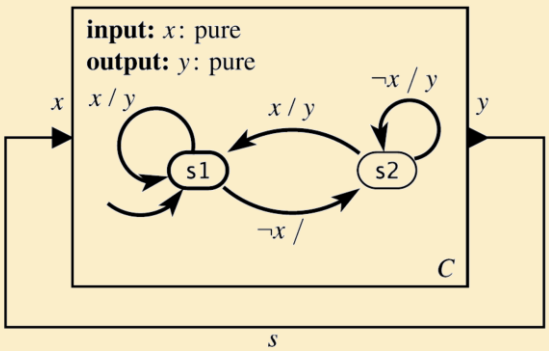




[dcc]

# Ill-Formed Example 2 (Uniqueness)

- In s1, both $s(n) = absent$ and $s(n) = present$ are fixed points
- In state s2, we get the unique $s(n) = present$
- Since state s1 is reachable, this composition is ill formed.




U.PORTO

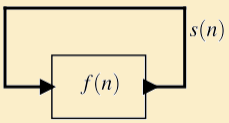
32

[dcc] U.PORTO

# Constructive Semantics: Single Signal

1. Start with *s(n)* unknown.
2. Determine as much as you can about $f_i(s(n))$, where $f_i$ is the firing function in state *i*.
3. Repeat step 2 until all values in *s(n)*:
   - *become known (whether they are present and what their values are), or*
   - *until no more progress can be made.*
4. If unknown values remain, then reject the model.

A state machine for which the procedure works in all reachable states is said to be **constructive**

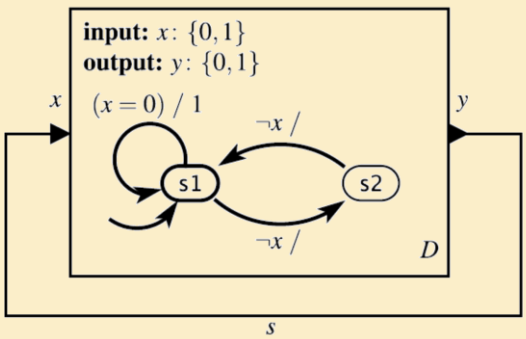





[dcc] U.PORTO

# Non-Constructive Well-Formed State Machine

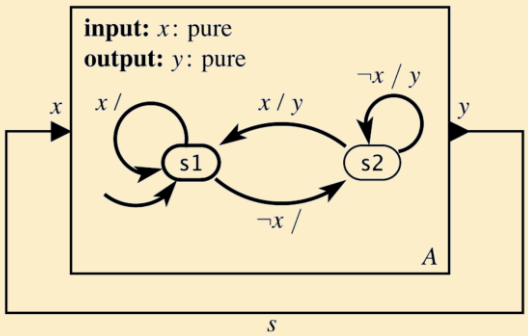
- In state s1, if the input is unknown, we cannot immediately tell what the output will be. We must try all the possible values for the input to determine that in fact $s(n) = absent$ for all n.



[dcc]

# Constructive Well-Formed State Machine

- In state s1, the machine may not produce an output. Therefore, we know the output is absent, even though the input is unknown.
- In state s2 , the machine must produce an output, so the output is present.




U.PORTO

35

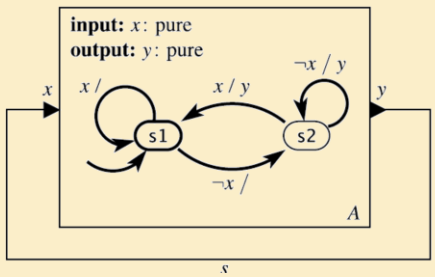
SE 2020/21 - Concurrent Models of Comp. - pbrandao

35

[dcc]

# Must / May Analysis

- In state s1, we can immediately determine that the machine may not produce an output ➔output is absent, even though the input is unknown.
  - *If the output is absent,* ➔ *the input is absent* → *procedure concludes.*
- In state s2, we can immediately determine that the machine must produce an output, ➔ output is present.




U.PORTO

36

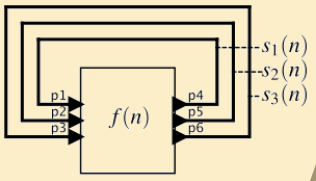
SE 2020/21 - Concurrent Models of Comp. - pbrandao

36

# Constructive Semantics: Multiple Signals

1. Start with s1(n), s2(n),…, sN(n), unknown.
2. Determine as much as you can about fi(s1(n), s2(n),…, sN(n)), where fi is the firing function in state i.
3. Repeat step 2 until all values in s1(n), s2(n),…, sN(n) become known (whether they are present and what their values are), or until no more progress can be made.
4. If unknown values remain, then reject the model.

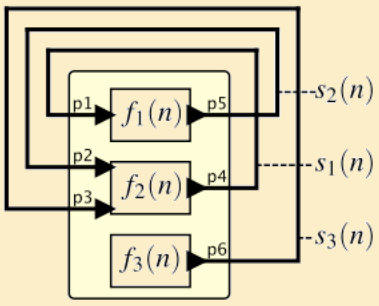
A state machine for which the procedure works in all reachable states is said to be constructive





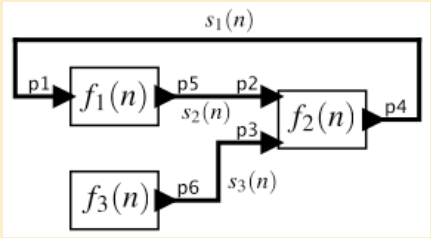

# Constructive Semantics: Multiple Actors

- Procedure is the same.



## Constructive Semantics: Arbitrary Structure

- Procedure is the same.
- A state machine language with constructive semantics will reject all compositions that in any iteration fail to make all signals known.
- Such a language rejects some well-formed compositions.






## Some conclusions:

- The emphasis of synchronous composition, in contrast with threads, is on determinate and analysable concurrency.
- Although there are subtleties with synchronous programs, all constructive synchronous programs have a unique and well-defined meaning.
- Automated tools can systematically explore all possible behaviours. This is not possible in general with threads.

[dcc]

# Summary

- Structure of models
  - *Models of computation*
- Synchronous-reactive models
  - *Fixed point semantics*

U.PORTO