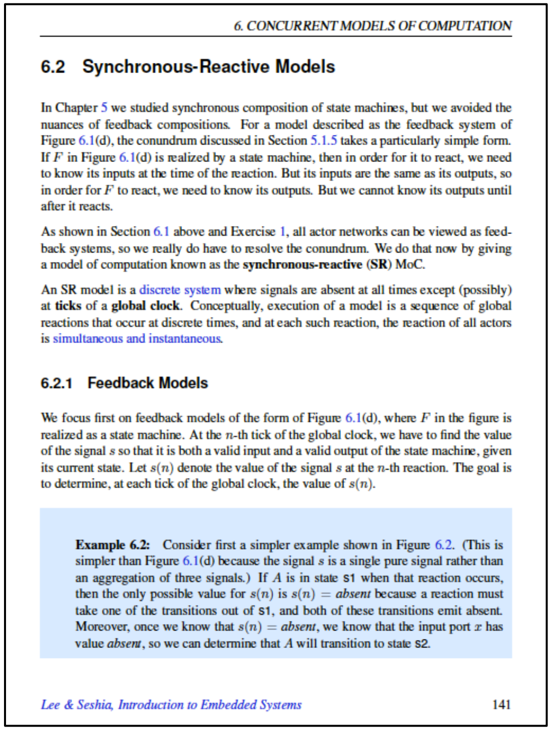
Lecture 12:
Synchronous- Reactive Models

Slides adapted from Edward A. Lee

Outline

- Feedback composition
- Well-formed vs ill-formed



6. CONCURRENT MODELS OF COMPUTATION

6.2 Synchronous-Reactive Models

In Chapter 5 we studied synchronous composition of state machines, but we avoided the nuances of feedback compositions. For a model described as the feedback system of Figure 6.1(d), the conundrum discussed in Section 5.1.5 takes a particularly simple form. If F in Figure 6.1(d) is realized by a state machine, then in order for it to react, we need to know its inputs at the time of the reaction. But its inputs are the same as its outputs, so in order for F to react, we need to know its outputs. But we cannot know its outputs until after it reacts.

As shown in Section 6.1 above and Exercise 1, all actor networks can be viewed as feedback systems, so we really do have to resolve the conundrum. We do that now by giving a model of computation known as the **synchronous-reactive** (**SR**) MoC.

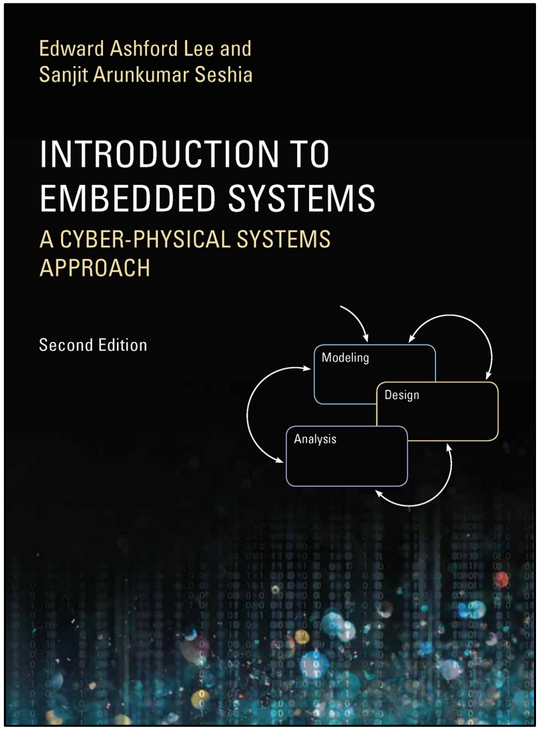
An SR model is a discrete system where signals are absent at all times except (possibly) at **ticks** of a **global clock**. Conceptually, execution of a model is a sequence of global reactions that occur at discrete times, and at each such reaction, the reaction of all actors is simultaneous and instantaneous.

6.2.1 Feedback Models

We focus first on feedback models of the form of Figure 6.1(d), where F in the figure is realized as a state machine. At the n-th tick of the global clock, we have to find the value of the signal s so that it is both a valid input and a valid output of the state machine, given its current state. Let $s(n)$ denote the value of the signal s at the n-th reaction. The goal is to determine, at each tick of the global clock, the value of $s(n)$.

Example 6.2: Consider first a simpler example shown in Figure 6.2. (This is simpler than Figure 6.1(d) because the signal s is a single pure signal rather than an aggregation of three signals.) If A is in state s1 when that reaction occurs, then the only possible value for $s(n)$ is $s(n) = absent$ because a reaction must take one of the transitions out of s1, and both of these transitions emit absent. Moreover, once we know that $s(n) = absent$, we know that the input port x has value *absent*, so we can determine that A will transition to state s2.

Lee & Seshia, Introduction to Embedded Systems 141



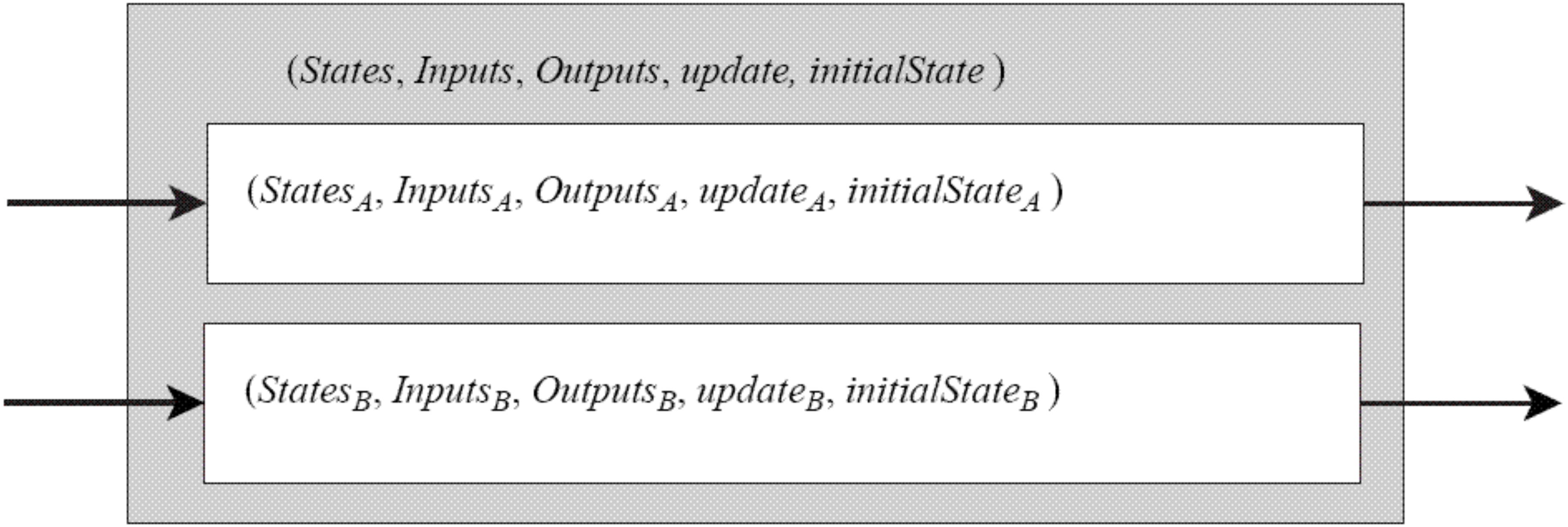
Concurrent Composition

Composition of State Machines:

- Side-by-side composition
- Cascade composition
- Hierarchical state machines (state refinement)
- Feedback composition

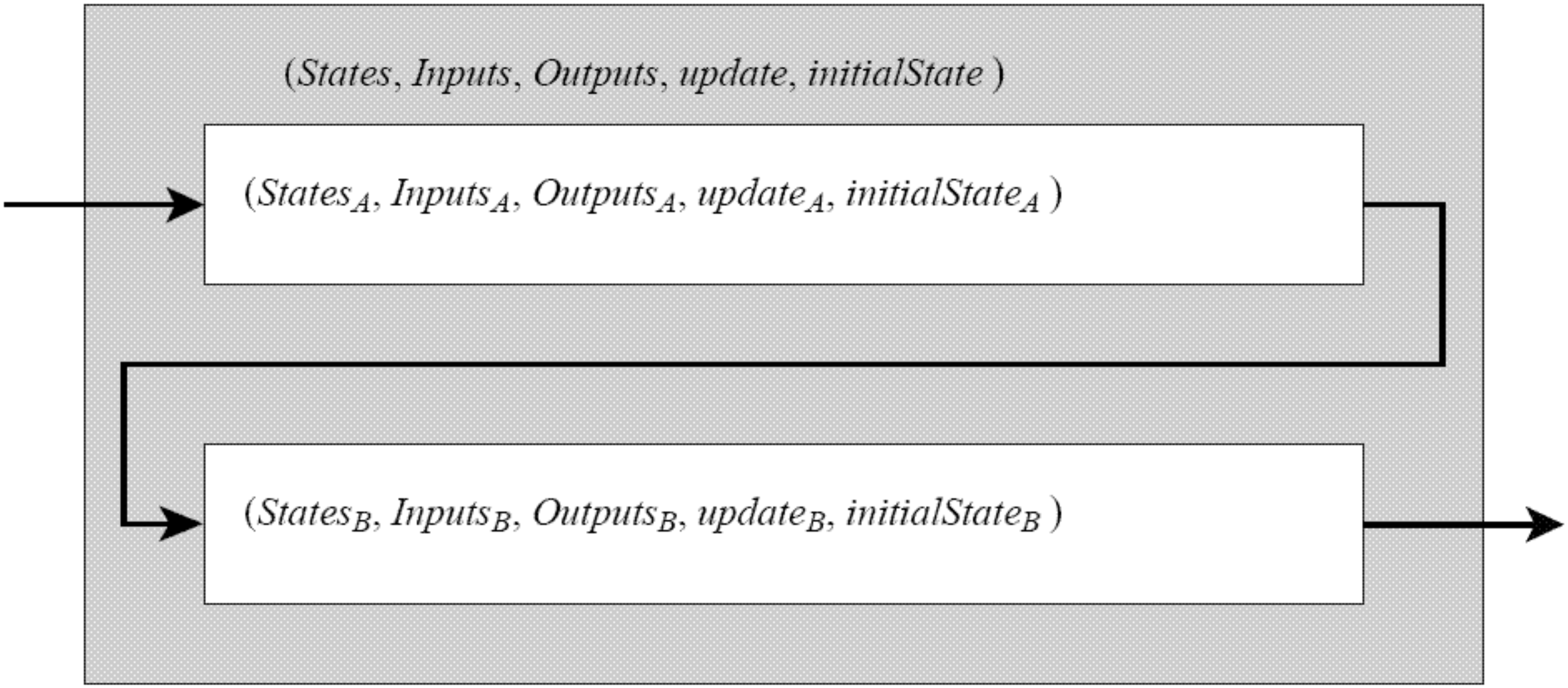
We will focus on synchronous composition, an abstraction that has been very effectively used in hardware design and is gaining popularity in software design.

Side-by-Side Composition



Synchronous composition: the machines react simultaneously and instantaneously.

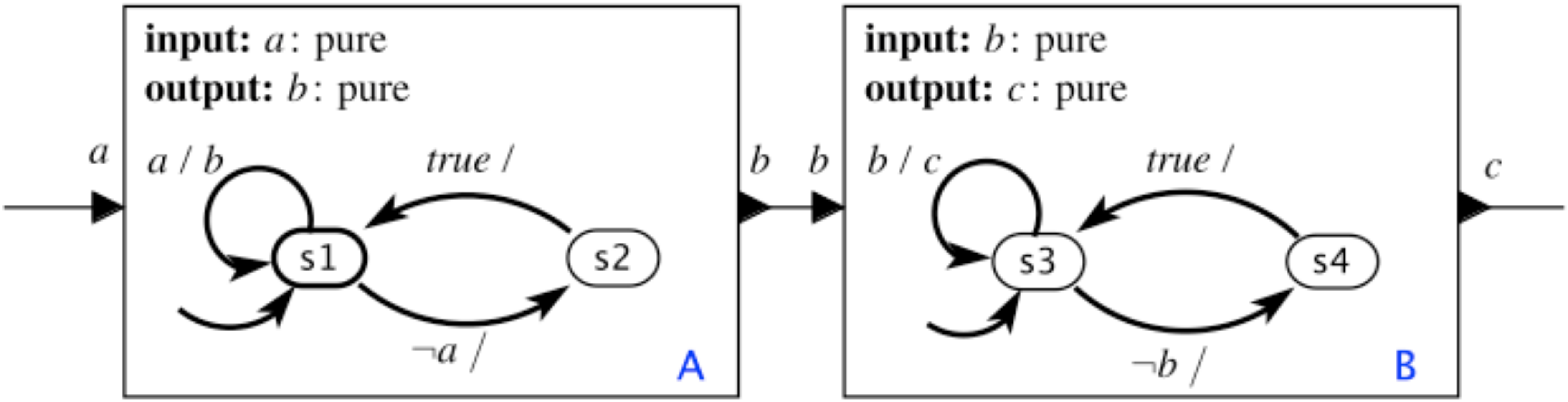
Cascade Composition



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

Synchronous Composition: Reactions are *Simultaneous* and *Instantaneous*

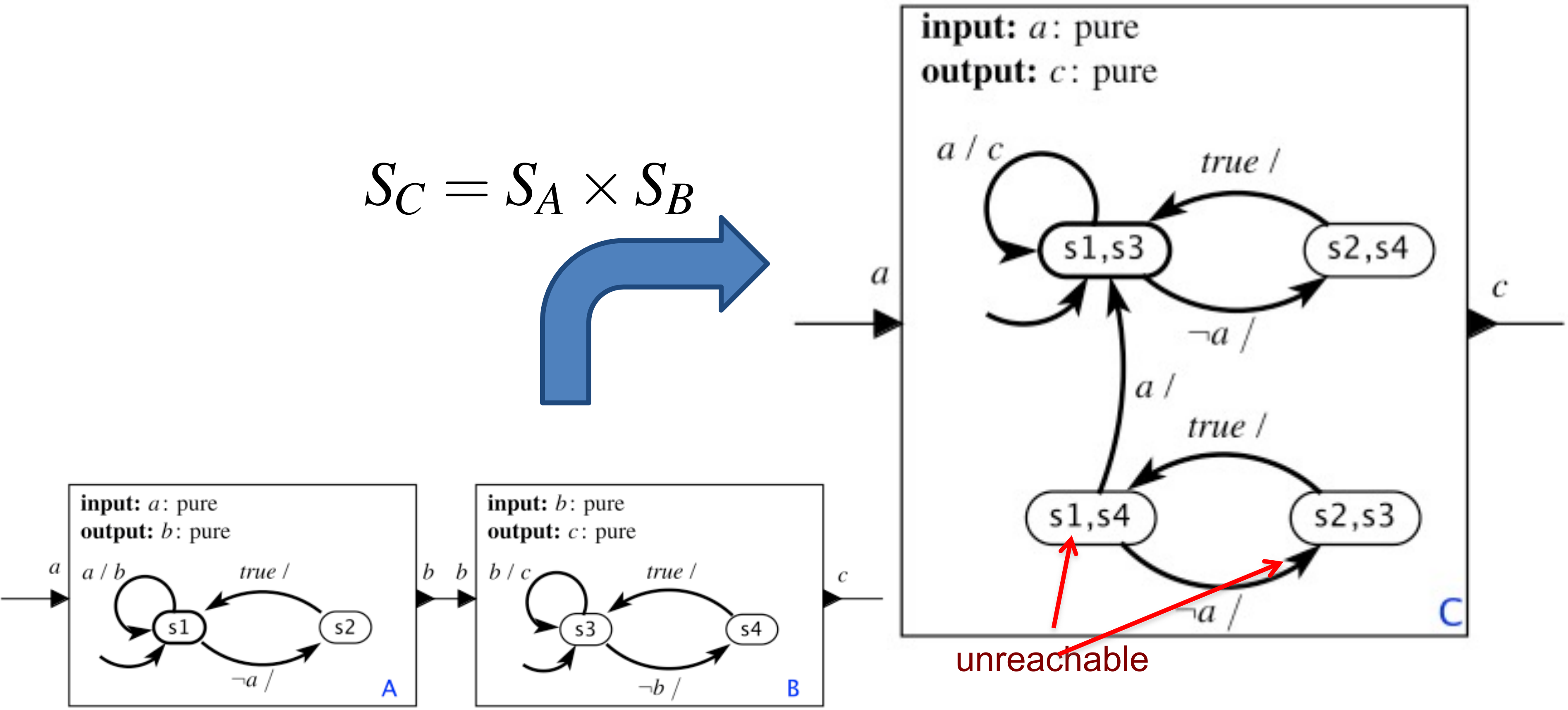
- Consider a cascade composition as follows:



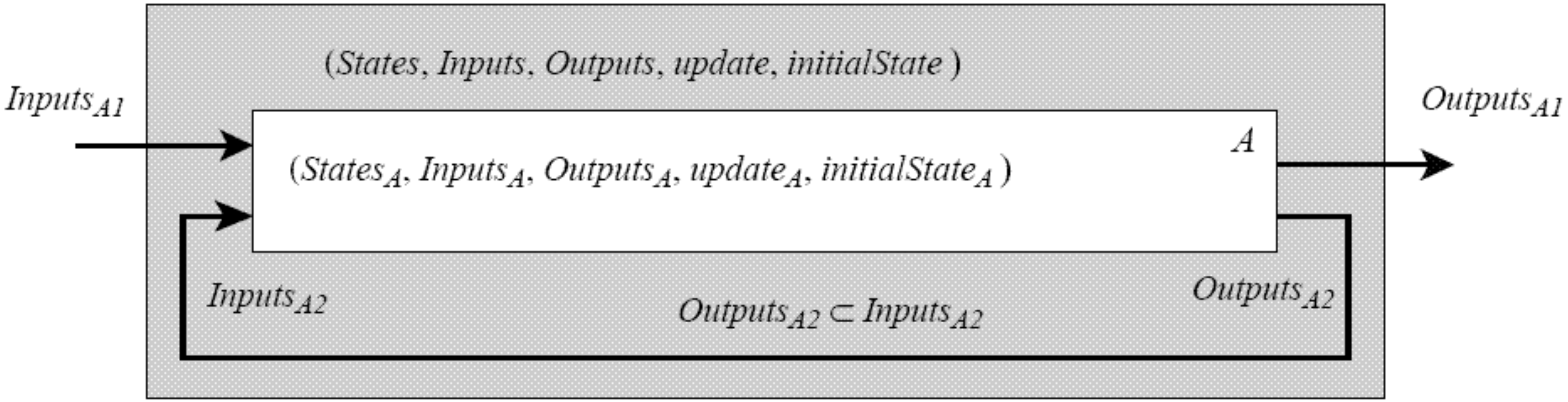
Synchronous Composition: Reactions are *Simultaneous* and *Instantaneous*

- 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.

$$S_C = S_A \times S_B$$

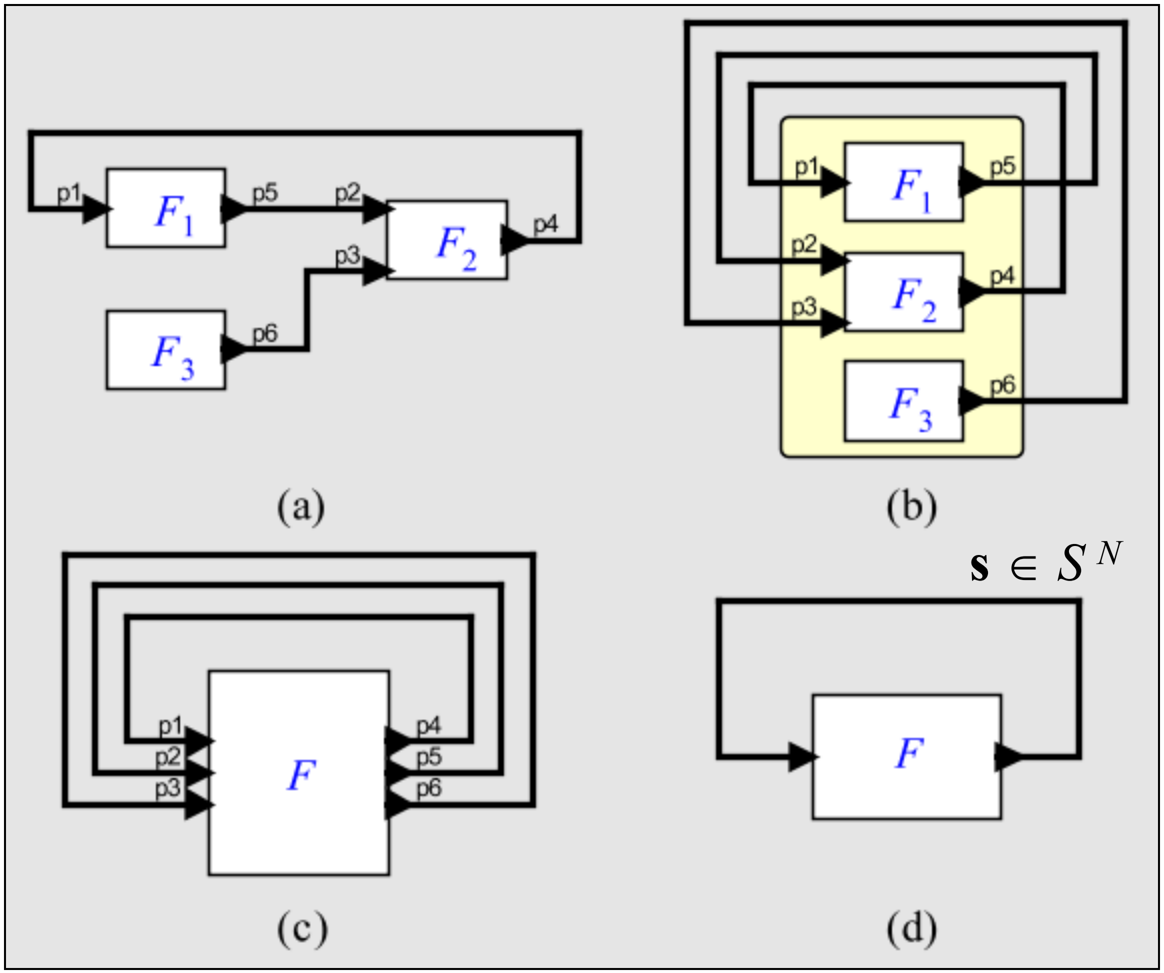


Feedback Composition



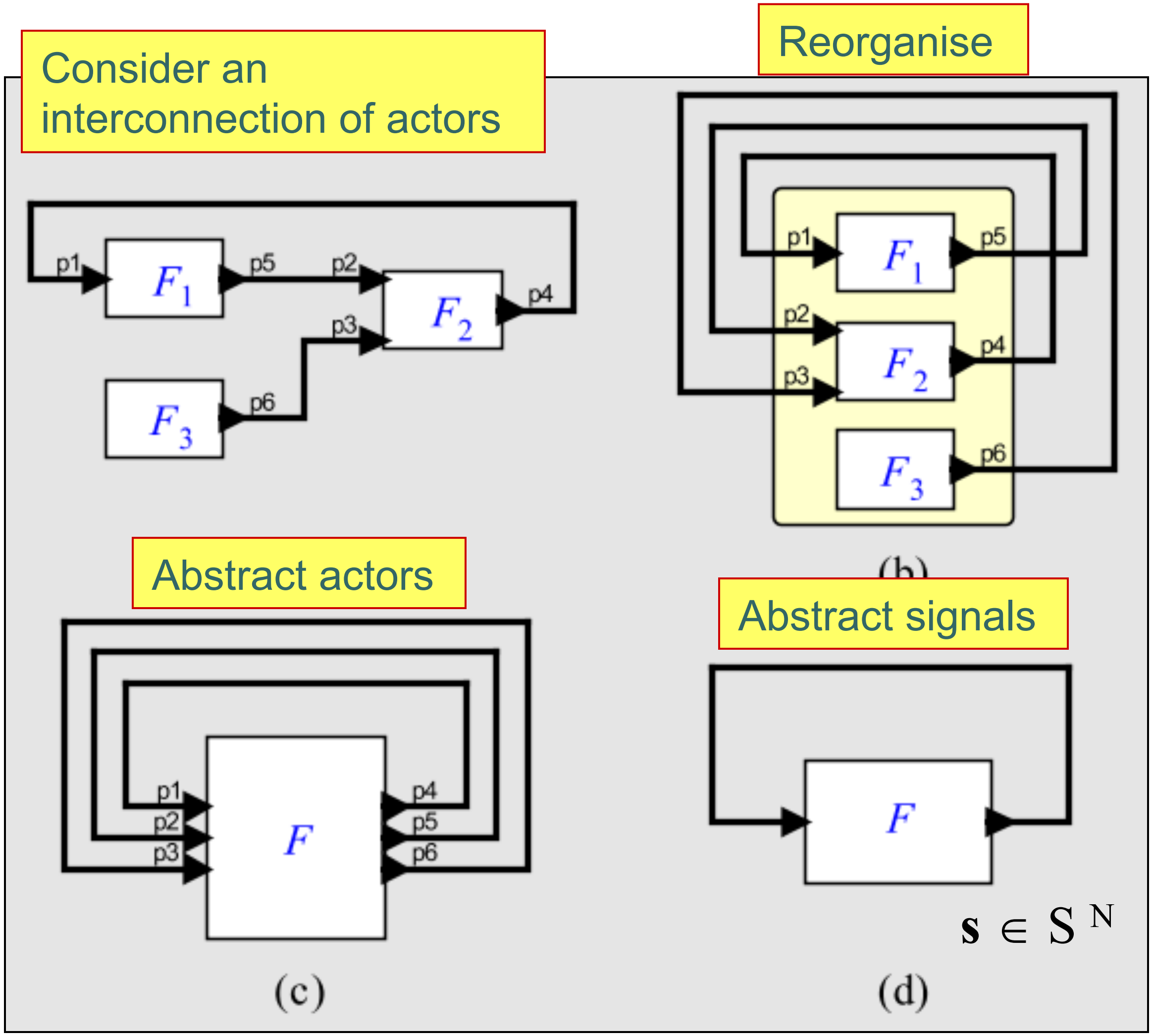
Turns out everything can be viewed as feedback composition…

Observation: Any Composition is a Feedback Composition



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

Fixed Point Semantics



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

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.

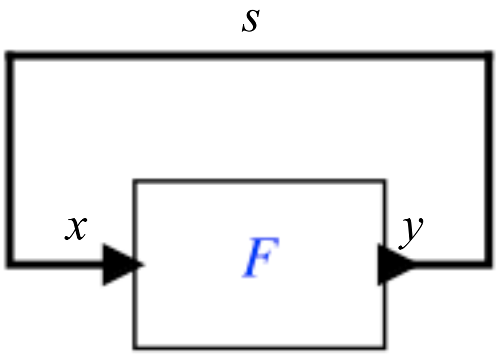
Data Types

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

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

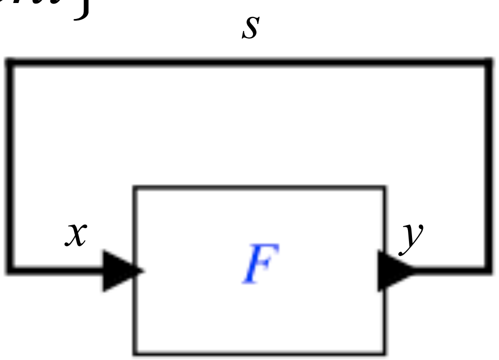
Firing Functions

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

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

such that

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



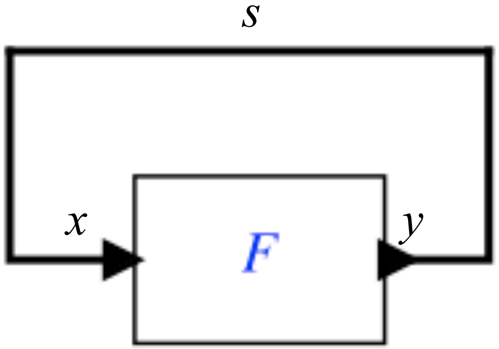
This too is a fixed point.

Well-Formed Feedback

At the n-th reaction, we seek $s(n) \in V_y \cup \{absent\}$ such that

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

There are two potential problems:

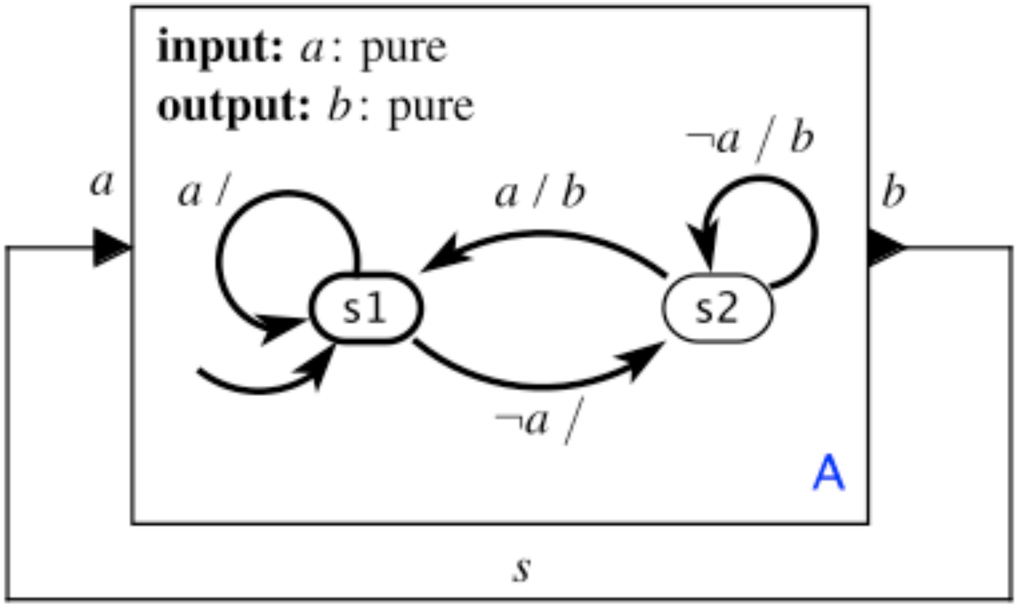


1. It does not exist.
2. It is not unique.

In either case, we call the system **ill formed**. Otherwise, it is **well formed**.

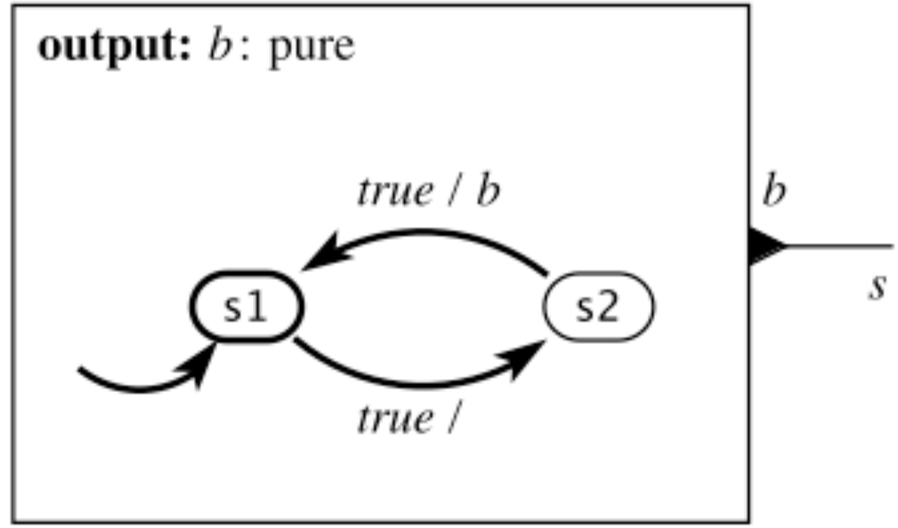
Note that if a state is not reachable, then it is irrelevant to determining whether the machine is well formed.

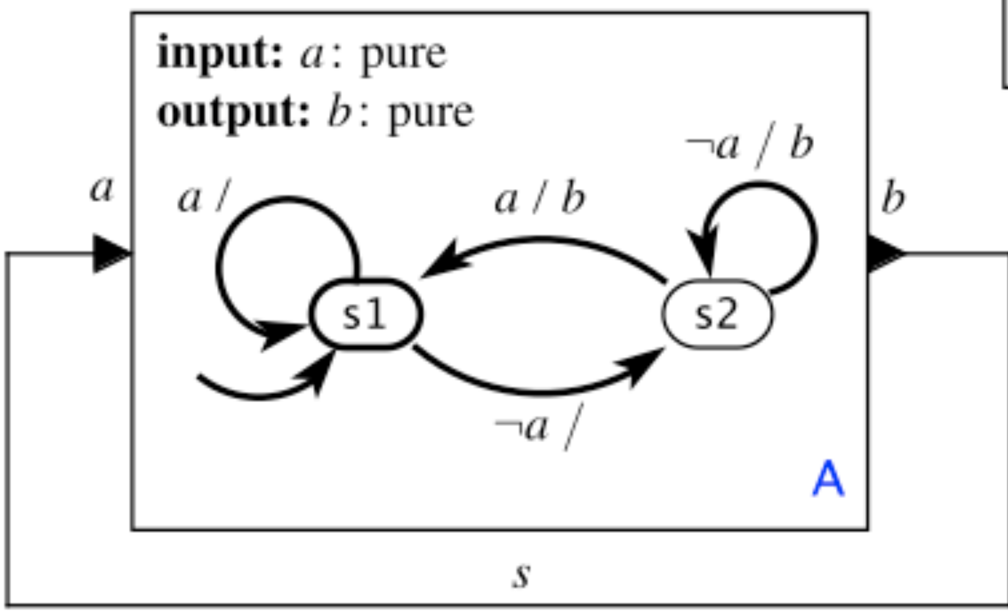
Well-Formed Example



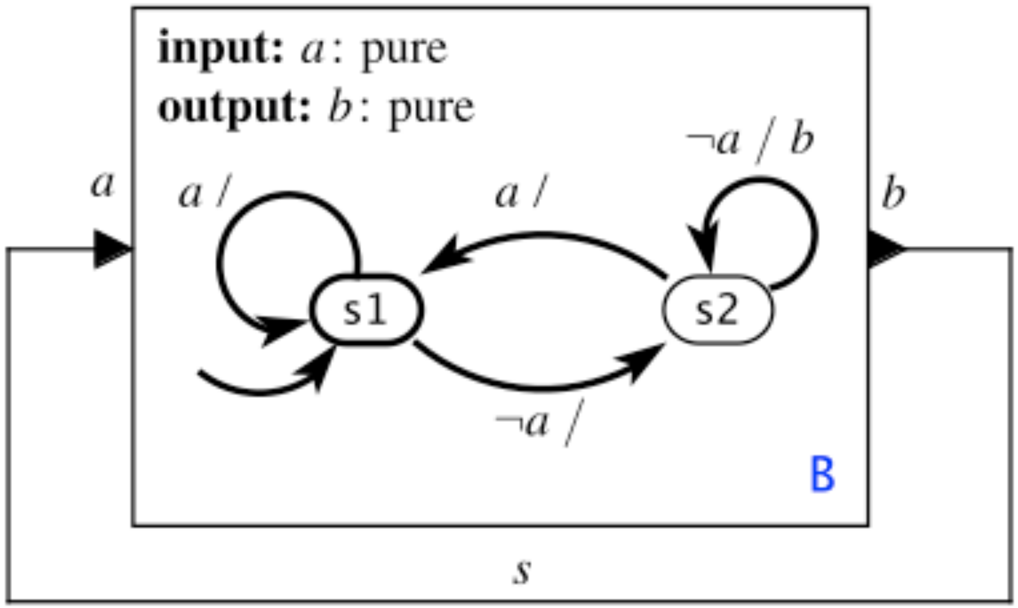
In state s1, we get the unique $s(n) = absent$.
In state s2, we get the unique $s(n) = present$.
Therefore, s alternates between *absent* and *present*.

Composite Machine



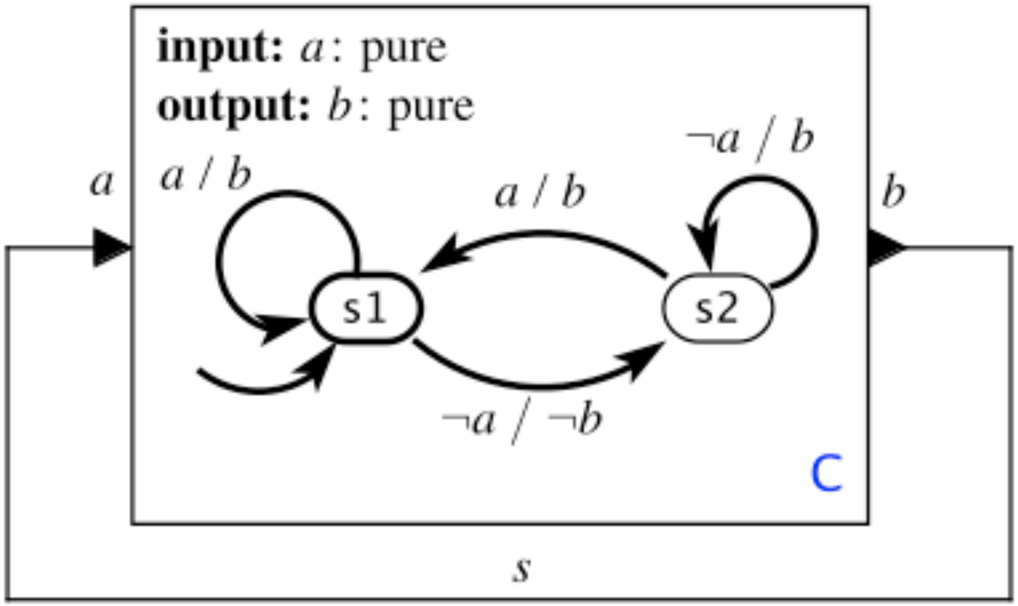


Ill-Formed Example 1 (Existence)



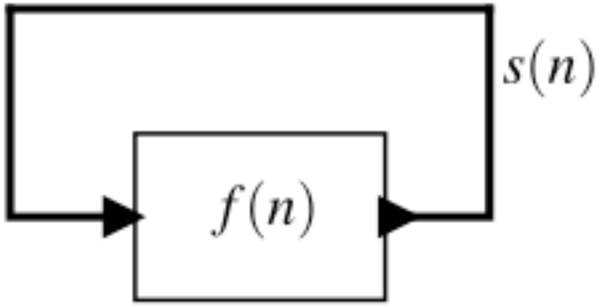
In state s1, we get the unique $s(n) = absent$.
In state s2, there is no fixed point.
Since state s2 is reachable, this composition is ill formed.

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.

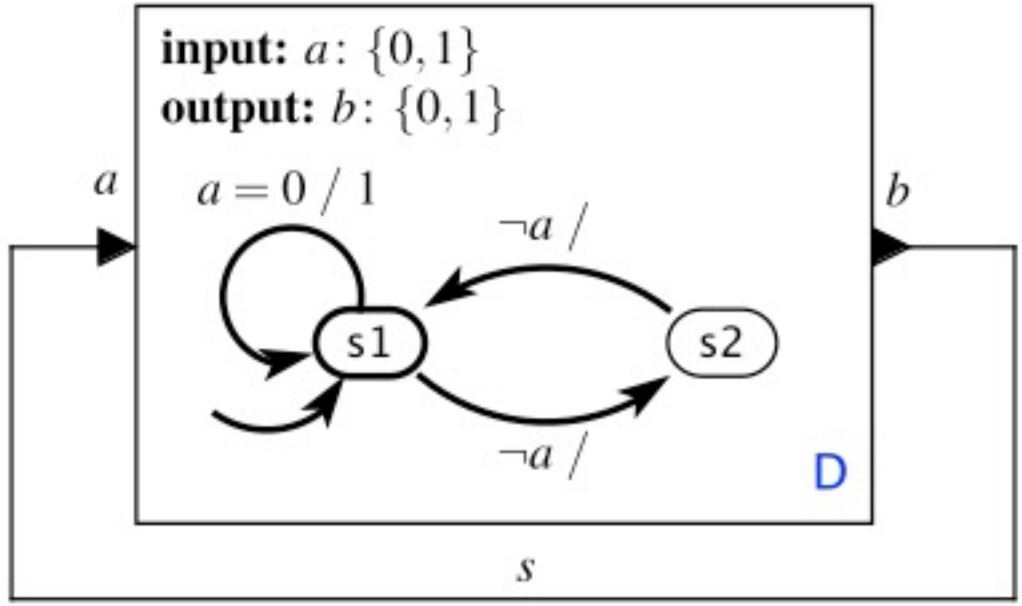
Constructive Semantics: Single Signal



1. Start with $s(n)$ *unknown*.
2. Determine as much as you can about $(f(n))(s(n))$.
3. If $s(n)$ becomes known (whether it is present, and if it is not pure, what its value is), then we have a unique fixed point.

A state machine for which this procedure works is said to be **constructive**.

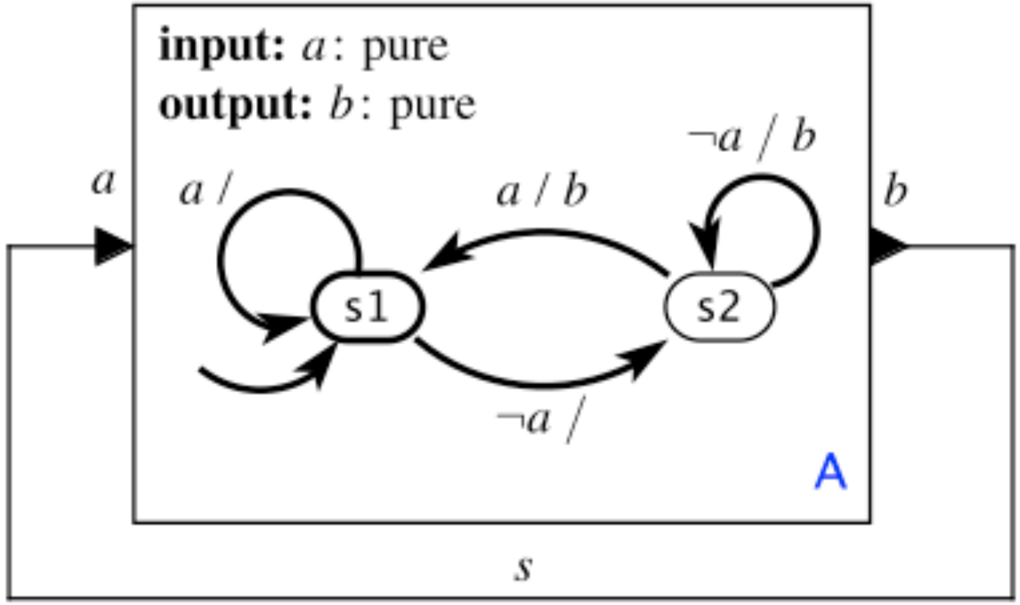
Non-Constructive Well-Formed State Machine



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

For non-constructive machines, we are forced to do **exhaustive search**. This is only possible if the data types are finite, and is only practical if the data types are small.

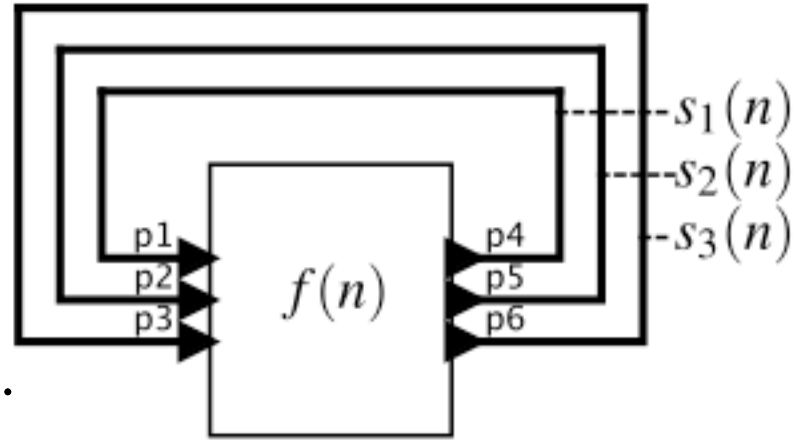
Must / May Analysis



For the above constructive machine, in state s1, we can immediately determine that the machine *may not* produce an output. Therefore, we can immediately conclude that the output is *absent*, even though the input is unknown.

In state s2, we can immediately determine that the machine *must* produce an output, so we can immediately conclude that the output is *present*.

Constructive Semantics: Multiple Signals



1. Start with $s_1(n), \cdots, s_N(n)$ *unknown*.

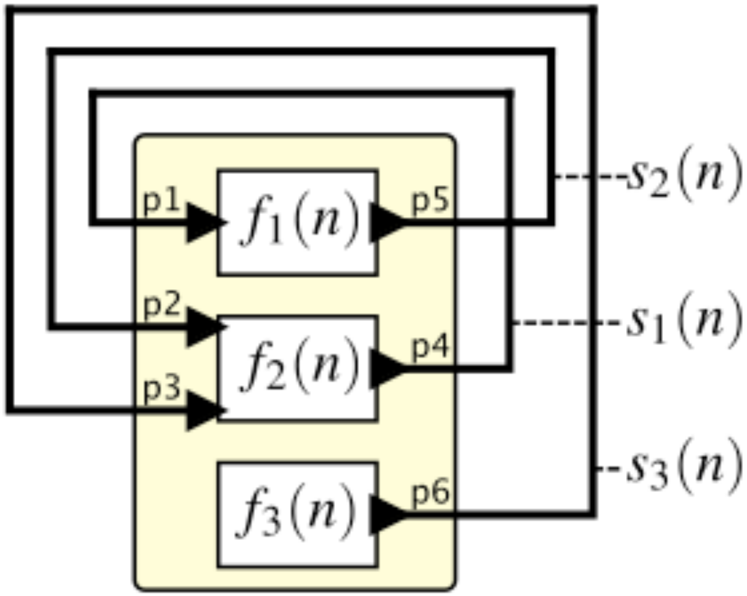
2. Determine as much as you can about $(f(n))(s_1(n), \cdots, s_N(n))$.

3. Using new information about $s_1(n), \cdots, s_N(n)$, repeat step (2) until no information is obtained.

4. If $s_1(n), \cdots, s_N(n)$ all become known, then we have a unique fixed point and a constructive machine.

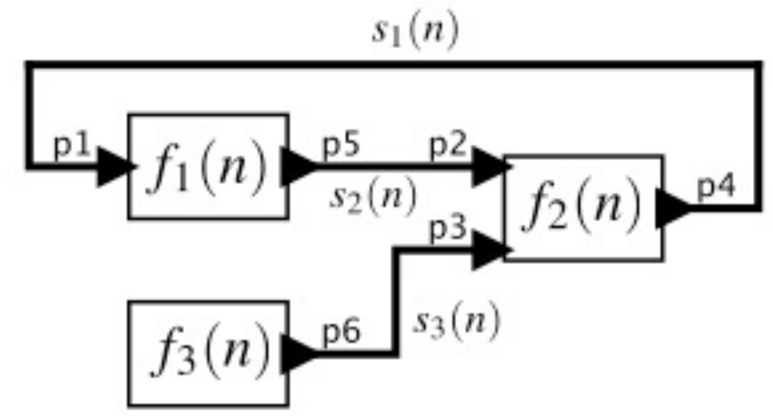
A state machine for which this procedure works is said to be **constructive**.

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.

Synchronous Reactive Models: Summary

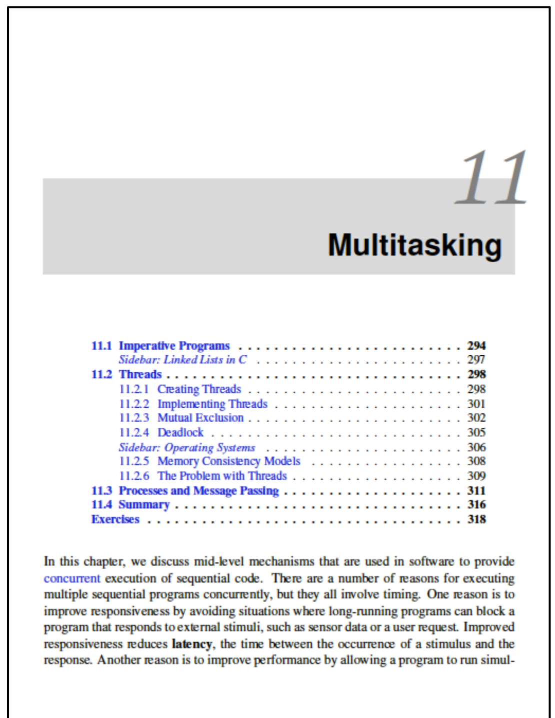
- The emphasis of synchronous composition, in contrast with threads, is on *determinate* and *analysable* concurrency.

- 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.

Things to do …

- Read Chapter 11
- Assignment 2 due Sept 16.
- Read over Workshop 6 for next week



11

Multitasking

In this chapter, we discuss mid-level mechanisms that are used in software to provide concurrent execution of sequential code. There are a number of reasons for executing multiple sequential programs concurrently, but they all involve timing. One reason is to improve responsiveness by avoiding situations where long-running programs can block a program that responds to external stimuli, such as sensor data or a user request. Improved responsiveness reduces **latency**, the time between the occurrence of a stimulus and the response. Another reason is to improve performance by allowing a program to run simul-