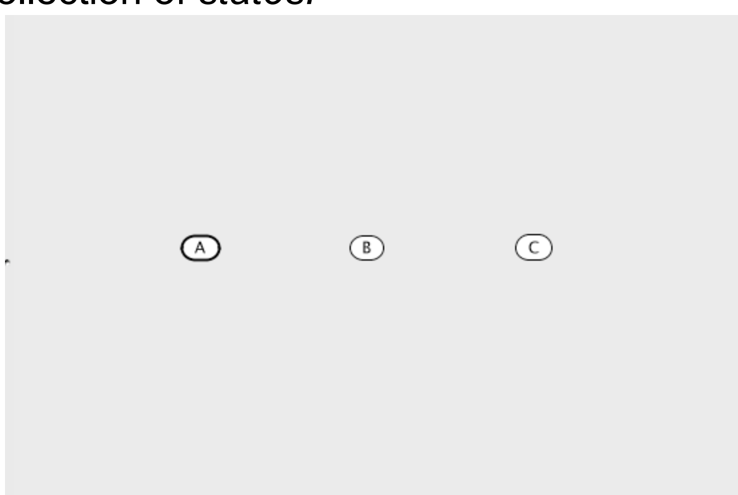
# Concurrent Models of Computation

Edward A. Lee
Robert S. Pepper Distinguished Professor, UC Berkeley
EECS 219D
*Concurrent Models of Computation*
Fall 2011

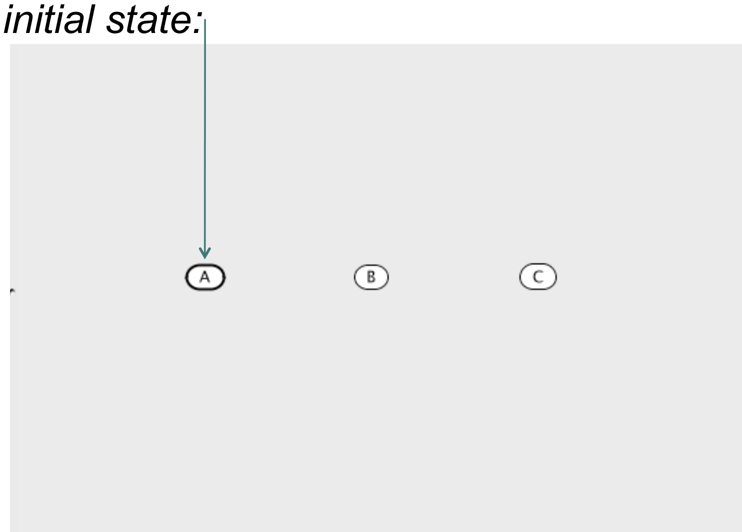


Week 7: Concurrent State Machines

A collection of *states:*

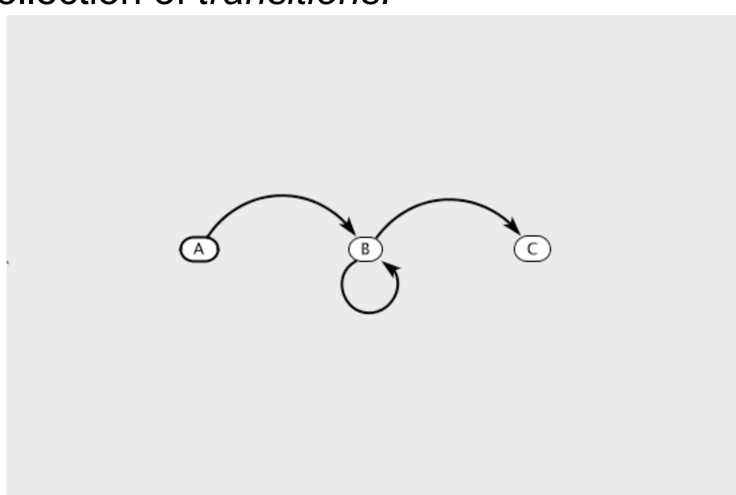


An *initial state:*

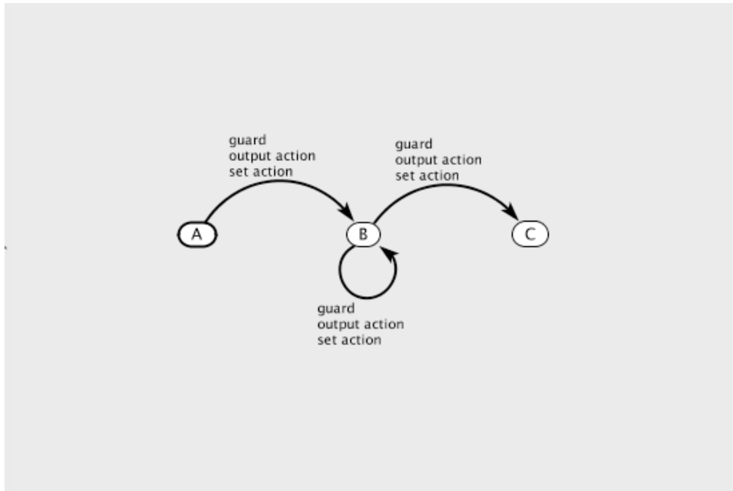





A collection of *transitions:*



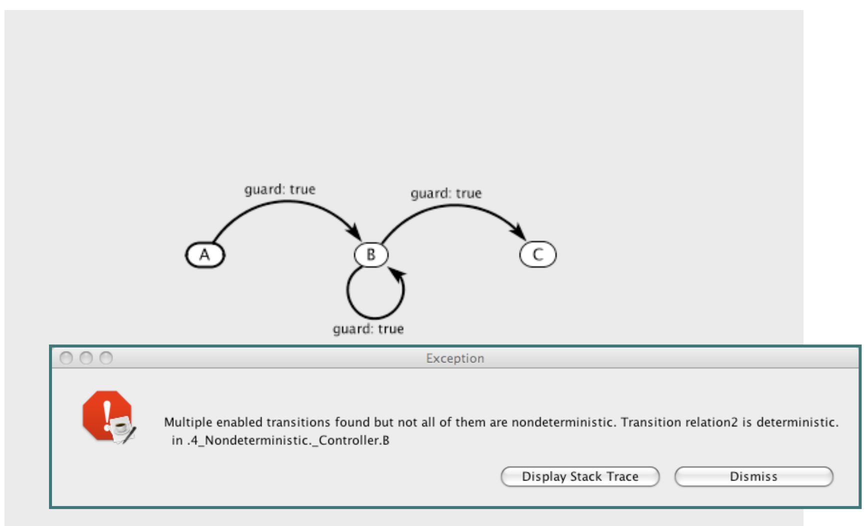
## Transitions have *labels:*




There are many variants of state machines, each giving different labels and semantics associated with those labels. Since we are interested in concurrent composition of state machines, we will give our state machines explicit inputs and outputs, and the labels will refer to these (reading and writing them).

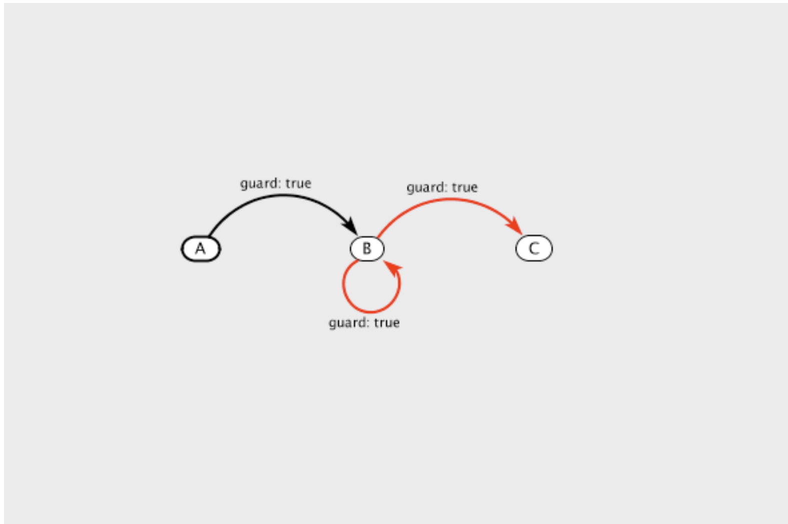


## Guards: Predicates on transitions




This state machine is nondeterminate because there are two simultaneously enabled transitions leaving state B. Ptolemy II by default rejects such state machines.

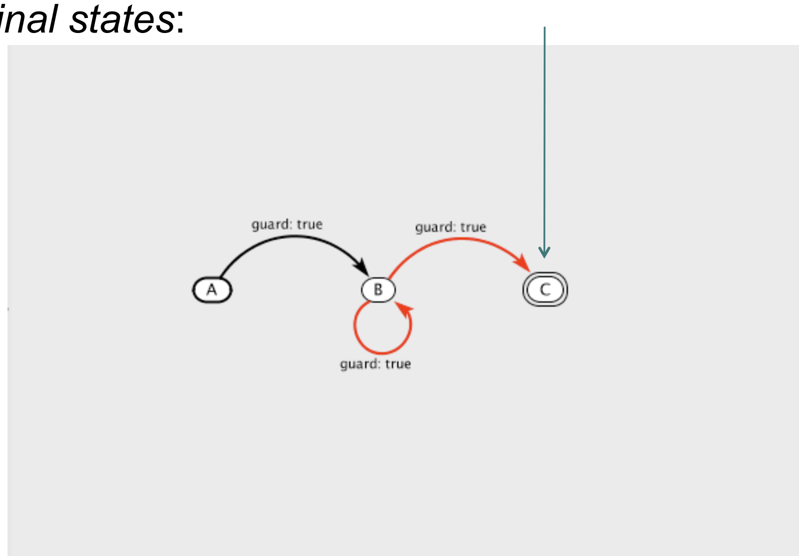
## Nondeterministic State Machines




Transitions can be marked *nondeterministic* and the model executes. This state machine will remain in state B for a random number of ticks then go to C and stay there.

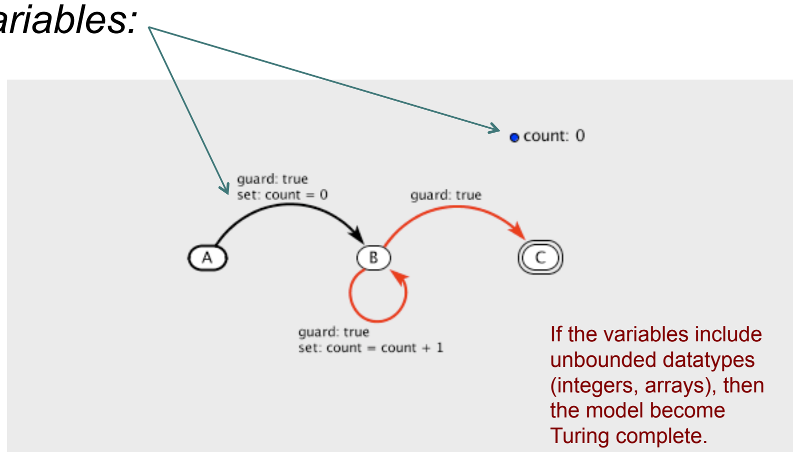


## *Final states*:




This model stops executing when it reaches state C.

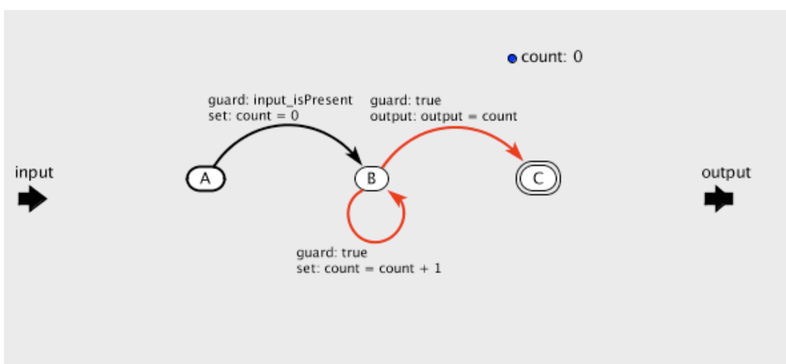
## Extended State Machines can operate on *variables:*




This model produces a random number and then stops.
The *set actions* perform the operations on the local variable *count*.
If the selection among transitions has fixed probability, then the random number generated will have a geometric distribution.

Lee 07: 9

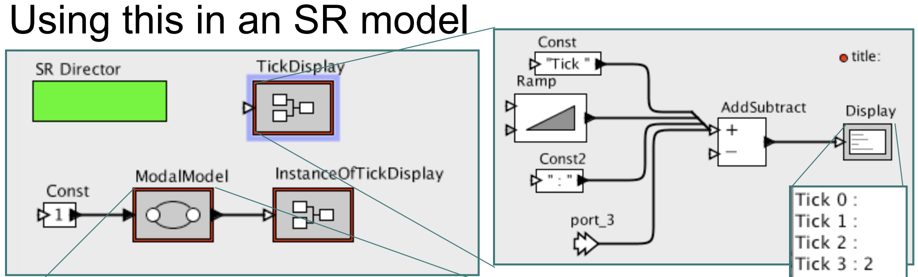
## I/O Automata

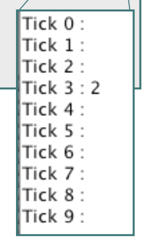


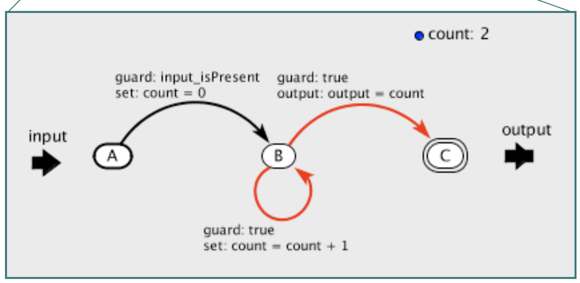

This model has an input port named "input" and an output port named "output". Given an input with any (non-absent) value, it starts counting. It counts a random number of ticks according to a geometric distribution, and then produces an output.

Lee 07: 10

## Using this in an SR model



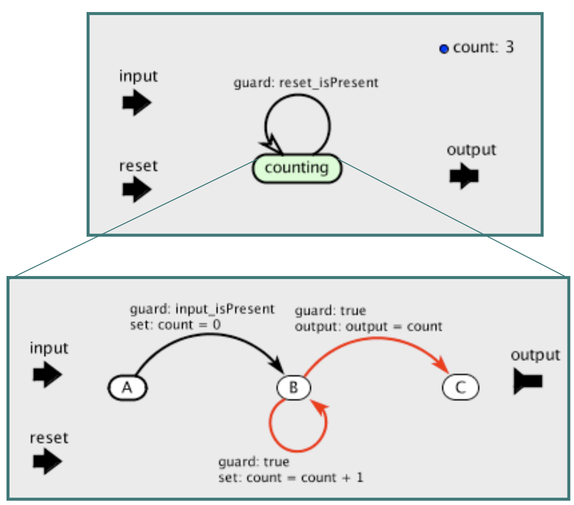

This model produces one value after a random amount of time (according to a geometric distribution), and then none after that.



## Hierarchical State Machines & Preemption

Here, the count can be preempted by a reset signal.

Here, the self transition is a *reset* transition, which means that when entering the destination state, it gets reset to its initial state.



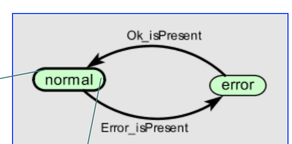
## Discussion

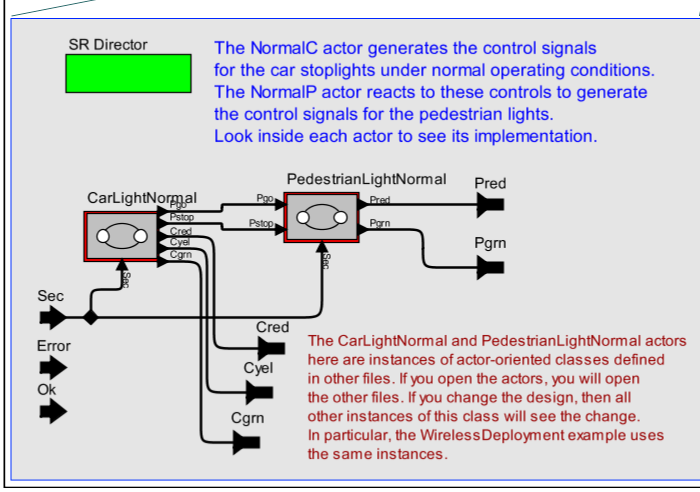
Hierarchy is only syntactic sugar.

How much syntax does it affect?



## Modal Models



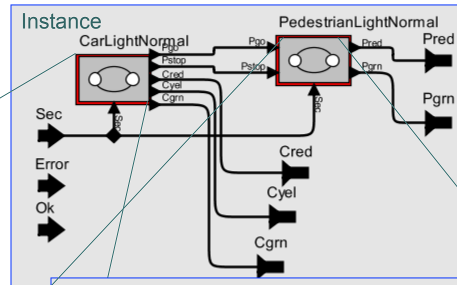

Whereas Statecharts lumps together the state machine semantics and the concurrency model, Ptolemy II separates these.

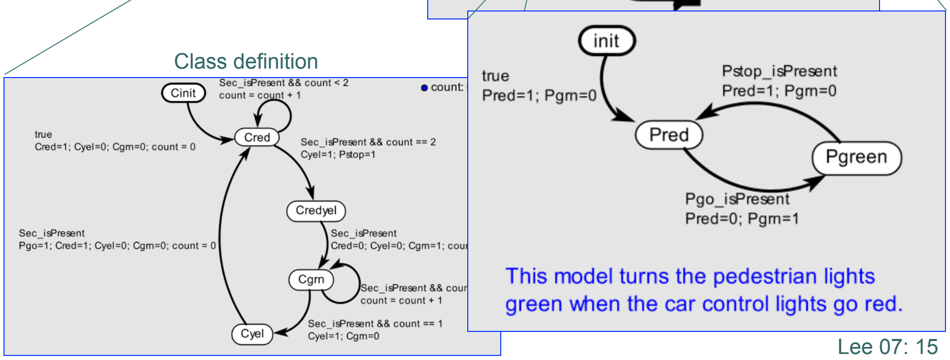
A state may refine to another state machine or to a concurrent model.

## Concurrent State Machines in Ptolemy II

The hierarchy can be further extended, where the concurrent model can include components that refine to state machines or other concurrent models.

This gives us concurrent state machines.





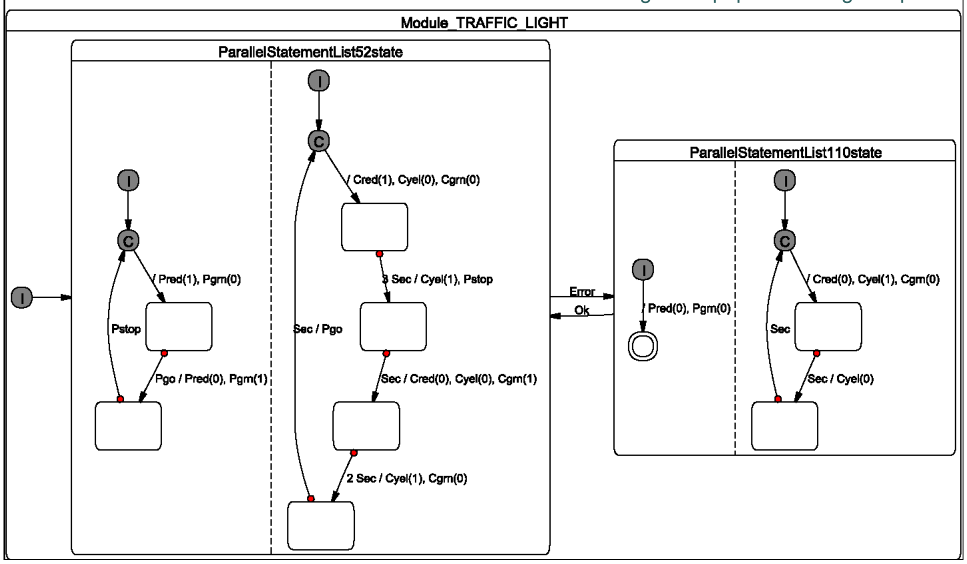

## Background on Concurrent State Machines

Statecharts [Harel 87]
I/O Automata [Lynch 87]
Esterel [Berry 92]
SyncCharts [André 96]
*Charts [Girault, Lee, Lee 99]
Safe State Machine (SSM) [André 03]
SCADE [Berry 03]

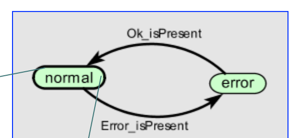
Simple Traffic Light Example in Statecharts

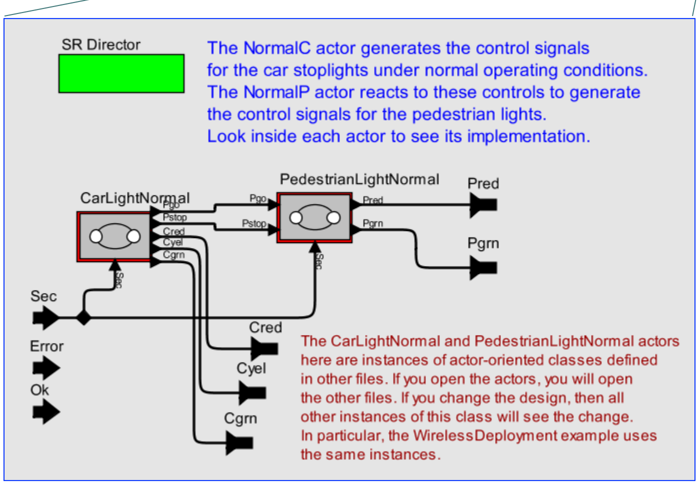
**Case study**

- Pred: pedestrian red signal
- Pgrn(0): turn pedestrian green off
- Cgrn: car green
- Sec: one second time
- 2 Sec: two seconds time
- Pgo/Pstop: pedestrian go/stop




# Traffic Light Example in Ptolemy II



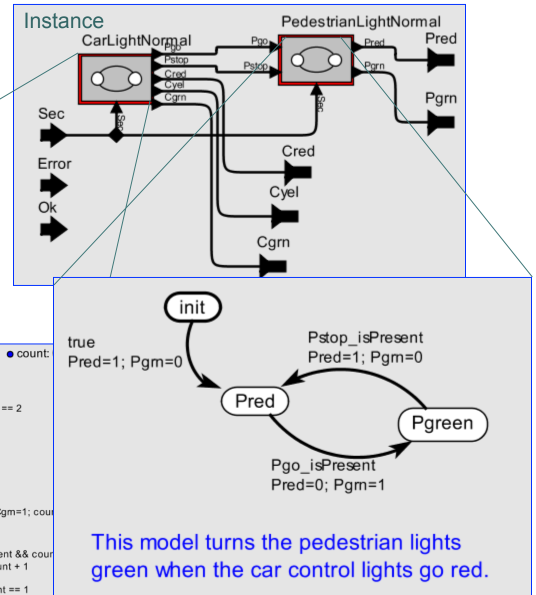



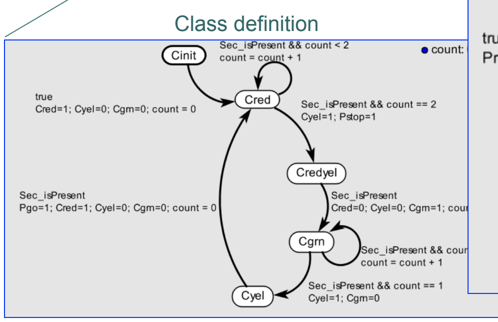

Whereas Statecharts lumps together the state machine semantics and the concurrency model, Ptolemy II separates these.

Here we have chosen the SR Director, which realizes a true synchronous fixed point semantics.

## Concurrent State Machines in Ptolemy II

In Ptolemy II, we have implemented an SR Director (for synchronous concurrent models) and an FSM Director (for sequential decision logic). Rather than combining them into one language (like Statecharts), Ptolemy II supports hierarchical combinations of MoCs.






## Syntax Comparisons between Statecharts and Ptolemy II

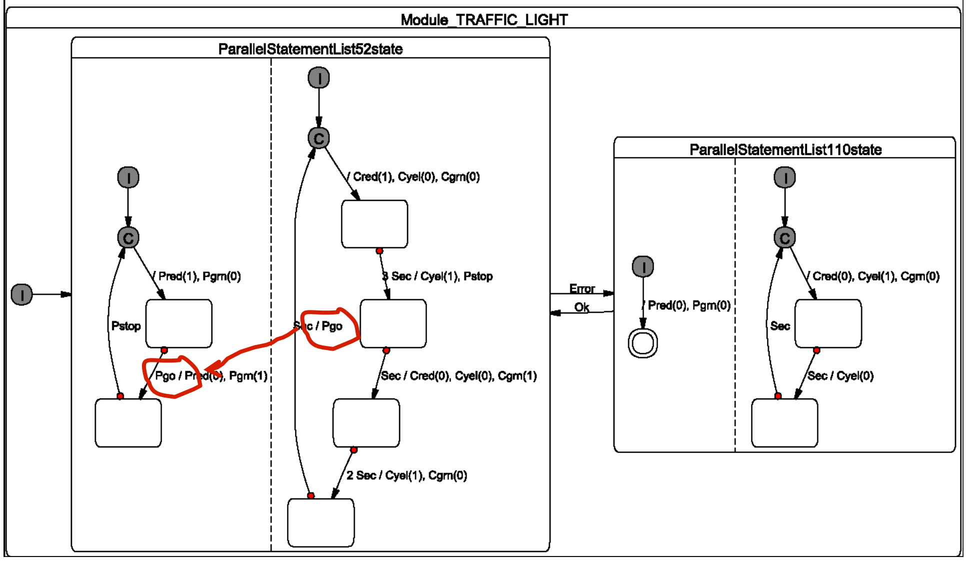
The Ptolemy II model and the Statecharts model differ in syntax. Some issues to consider when evaluating a syntax:

- Rendering on a page
- Showing dependencies in concurrent models
- Scalability to complex models
- Reusability (e.g. with other concurrency models)
- Special notations (e.g. “3 Sec”).

Simple Traffic Light Example in Statecharts, from Reinhard von Hanxleden, Kiel University

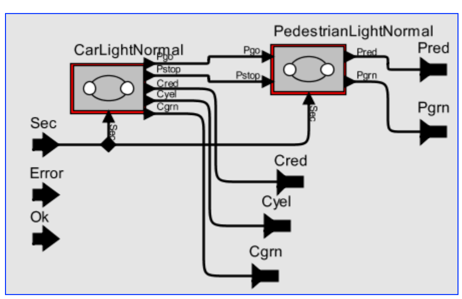
**Case study for Ptolemy II Design**

In StateCharts, the communication between concurrent components is not represented graphically, but is rather represented by name matching. Can you tell whether there is feedback?




# Syntax comparisons

Now can you tell whether there is feedback?




Lee 07: 22

## Semantics Comparisons

The Ptolemy II model and the Statecharts model have similar semantics, but combined in different ways. Some issues to consider:

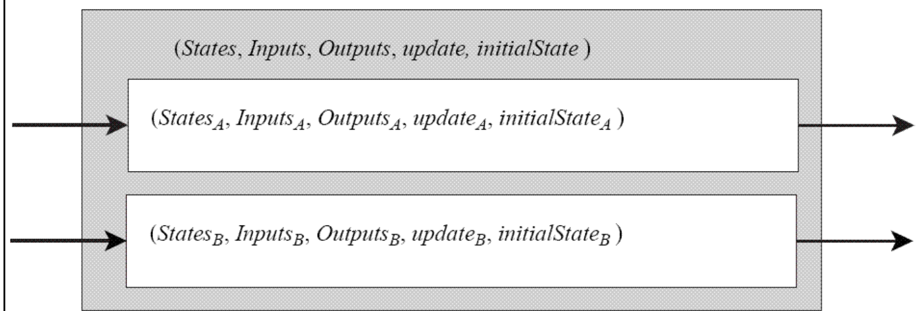
- Separation of concurrency from state machines
- Nesting of distinct models of computation
- Expanding beyond synchronous + FSM to model the (stochastic) environment and deployment to hardware.
- Styles of synchronous semantics (Ptolemy II realizes a true fixed-point constructive semantics).



## Constructive Semantics (Part 1)

When using state machines with SR providing the concurrency model, then semantics is given by the least fixed point, obtained constructively via the Kleene fixed-point theorem.

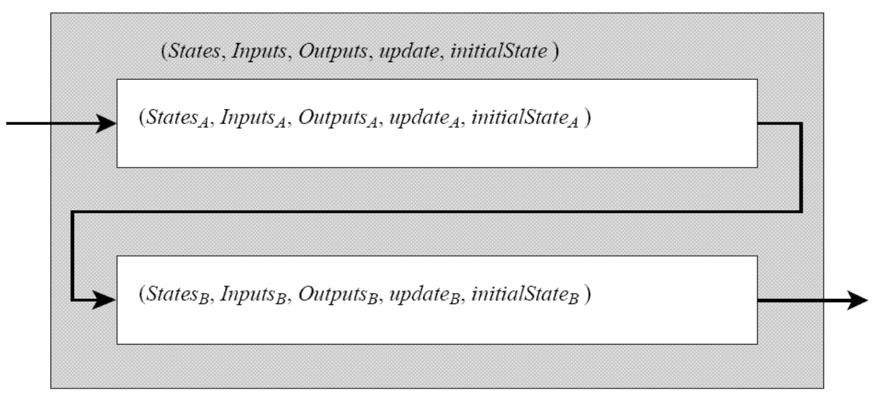
## Side-by-Side Composition




Synchronous composition: the machines react simultaneously and instantaneously.



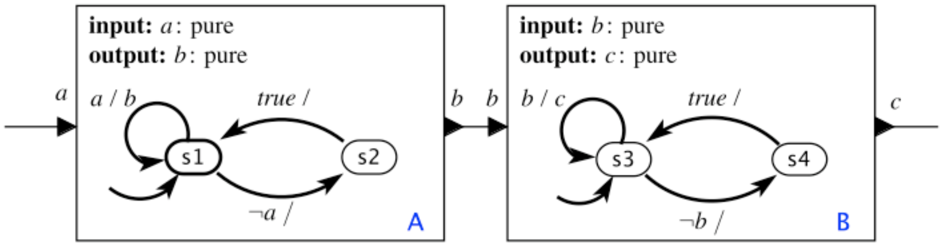
## Cascade Composition




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:

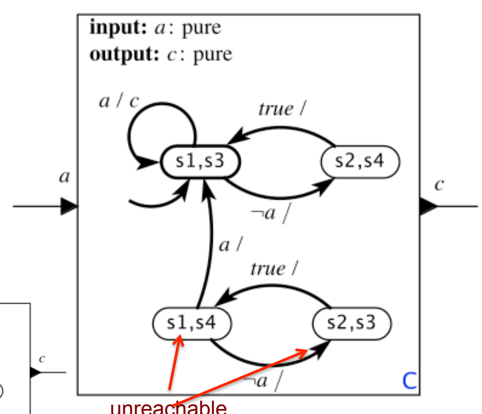


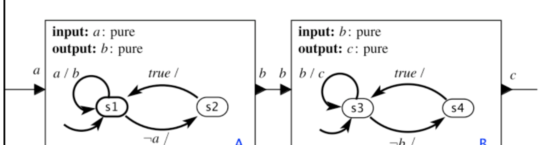



## Synchronous Composition:
## Reactions are *Simultaneous* and *Instantaneous*

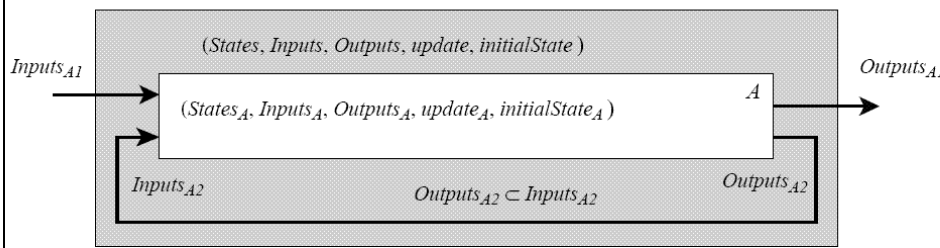
In this model, you must not think of machine A as reacting before machine B. If it did, the unreachable states would not be unreachable.

$$S_C = S_A \times S_B$$



## Feedback Composition




Recall that everything can be viewed as feedback composition.

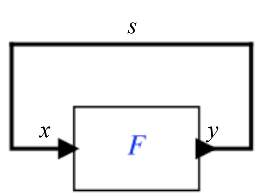
Lee 07: 29

## 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:



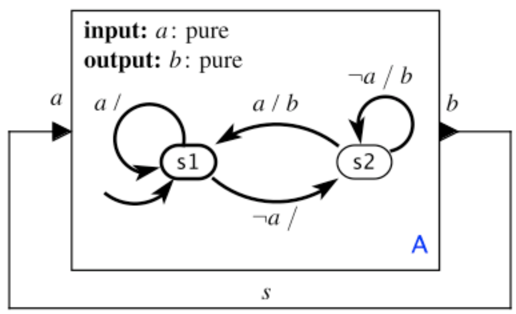

1. It does not exist.
2. It is not unique.

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.

Lee 07: 30

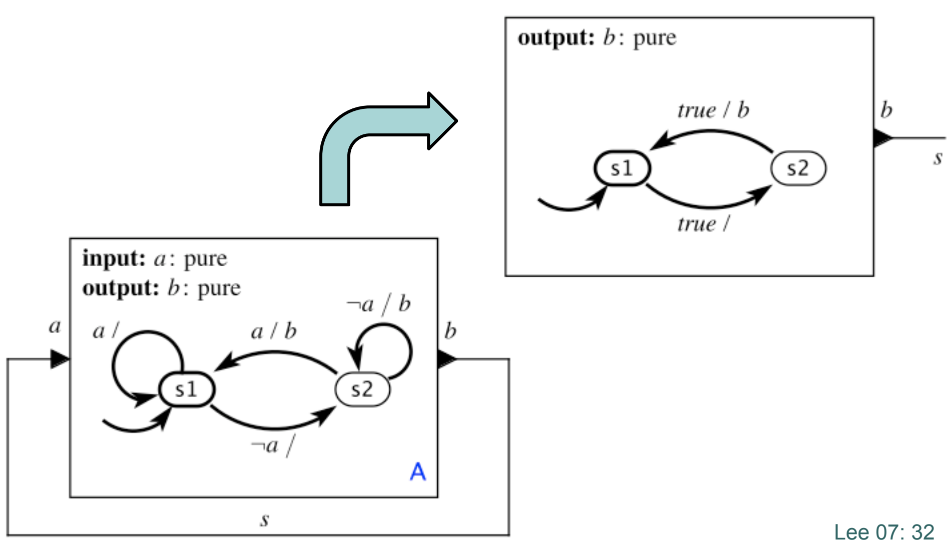
## Well-Formed Example



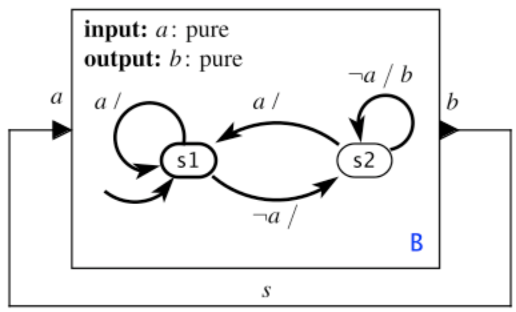

In state s1, we get the unique $s(n) = absent$.
In state s2, we get the unique $s(n) = present$.
Therefore, $s$ alternates between *absent* and *present*.



## Composite Machine



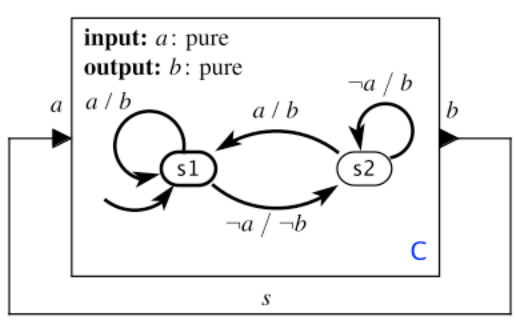
## Ill-Formed Example 1 (Existence)




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.

Lee 07: 33

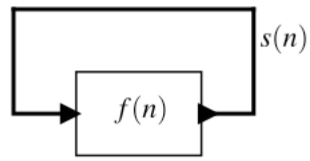
## Ill-Formed Example 2 (Uniqueness)




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.

Lee 07: 34

## Constructive Semantics: Single Signal

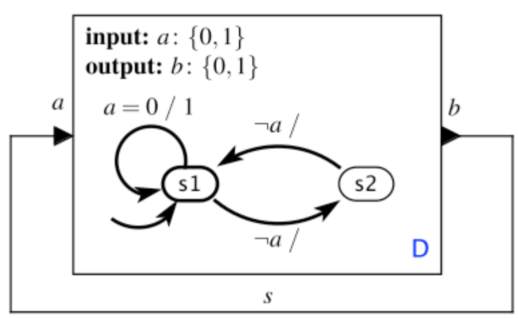



1. Start with $s(n)$ *unknown*.
2. Determine as much as you can about $(f(n))(s(n))$.
3. If $s(n)$ becomes known (whether it is present, and if it is not pure, what its value is), then we have a unique fixed point.

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



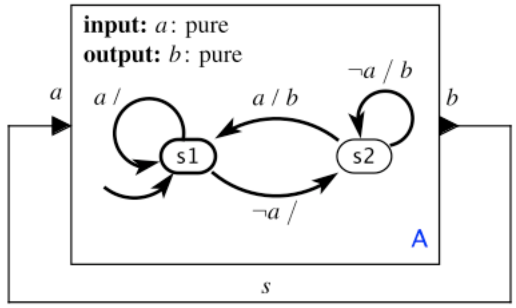
## Non-Constructive Well-Formed State Machine




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*.

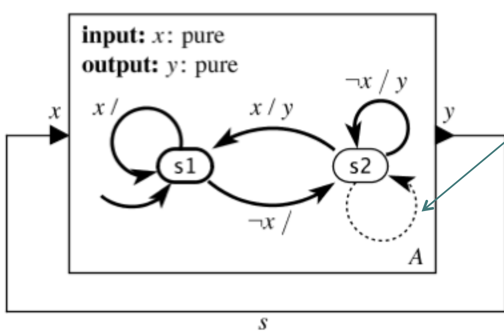
Lee 07: 37

## Subtlety: Constructive Semantics (Part 2)

The constructive semantics is based on two things:

- Iteration to a least fixed point.
- Construction of the functions $f(n)$ from an FSM

The second of these is subtle.




Implicit default transition does not produce an output. Hence, to know that the output is present, must analyze guards to know that at least one of them is true! In general, this analysis is undecidable. In practice, it is far from trivial.

Lee 07: 38

## Constructive Semantics, Esterel Style (Berry, 2003)

- Iteration to a least fixed point.
- Construction of the functions $f(n)$ from an FSM

Where the latter asserts:

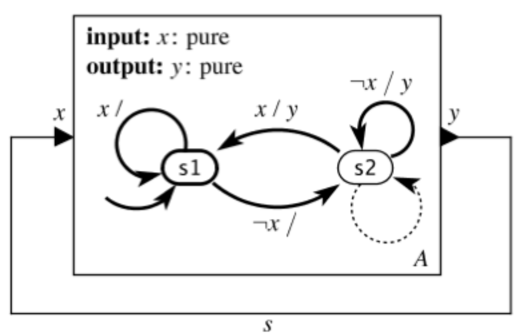
- An output is absent if no transition that might become enabled asserts it is present.
- An output is present if there exists a transition that is enabled and asserts the output.

(Notice the asymmetry).



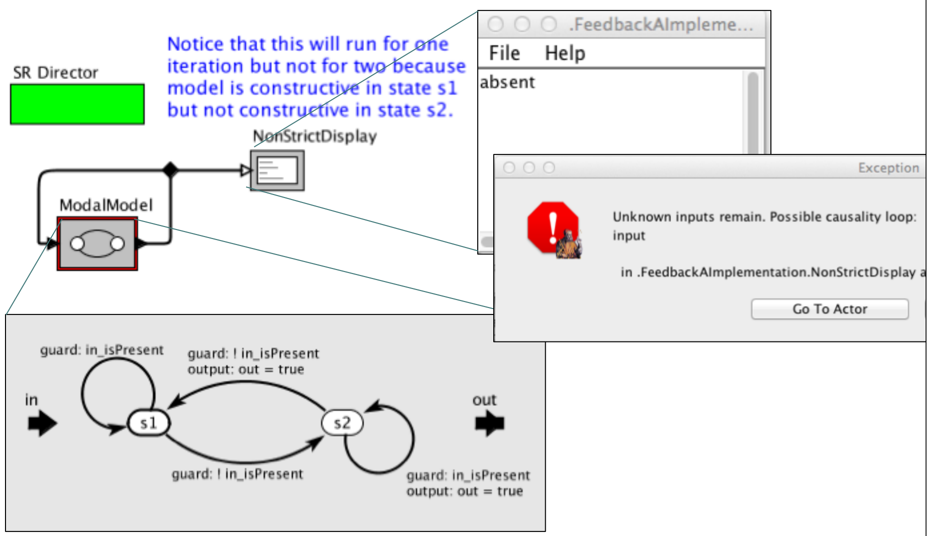
## Constructive Semantics, Esterel Style (Berry, 2003)

Rejects this model:



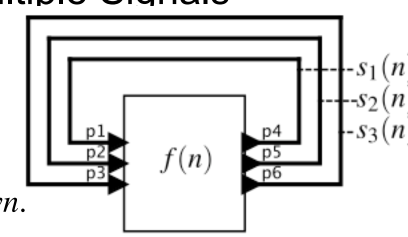

because when the input is unknown, there is no single transition enabled that asserts the transition.

## Ptolemy II Implements the Esterel-Style Constructive Semantics




Lee 07: 41

## Constructive Semantics: Multiple Signals

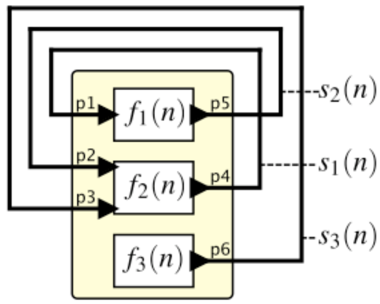



1. Start with $s_1(n), \cdots, s_N(n)$ *unknown*.
2. Determine as much as you can about $(f(n))(s_1(n), \cdots, s_N(n))$.
3. Using new information about $s_1(n), \cdots, s_N(n)$, repeat step (2) until no information is obtained.
4. If $s_1(n), \cdots, s_N(n)$ all become known, then we have a unique fixed point and a constructive machine.

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

Lee 07: 42

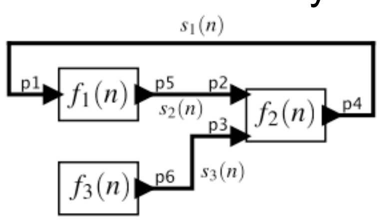
## Constructive Semantics: Multiple Actors




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.

## Conclusions

- State machines, extended state machines, and I/O automata provide expressive sequential decision logic.
- Variants support hierarchy (in different ways), nondeterminism, etc.
- Statecharts is a composition of a single-clock synchronous-reactive concurrent MoC with finite state machines.
- Ptolemy II separates these two semantic models using the idea of *modal models*.