



Concurrent Models of Computation

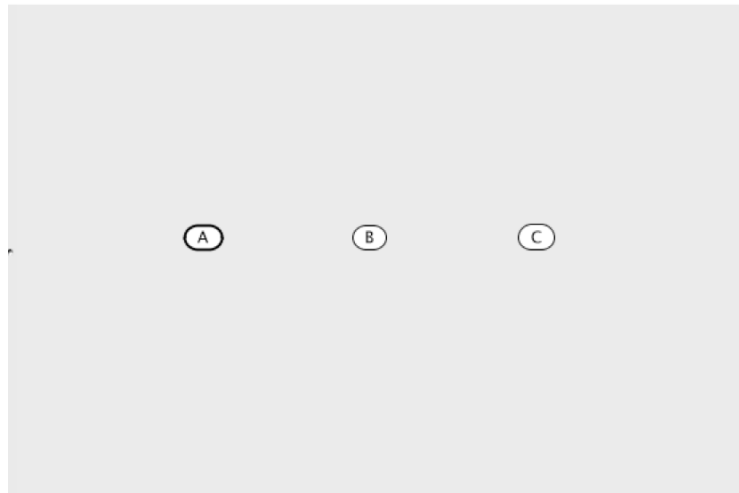
Edward A. Lee

Robert S. Pepper Distinguished Professor, UC Berkeley
EECS 290n – Advanced Topics in Systems Theory
Concurrent Models of Computation
Spring 2009

Copyright © 2009, Edward A. Lee, All rights reserved

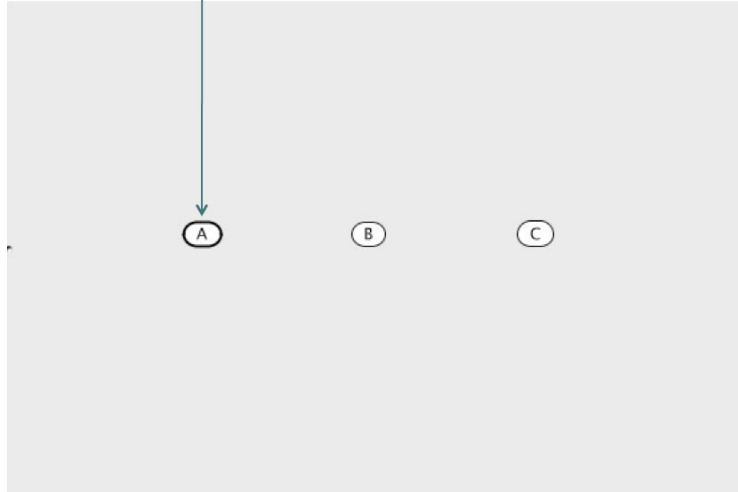
Week 7: Concurrent State Machines

A collection of *states*:



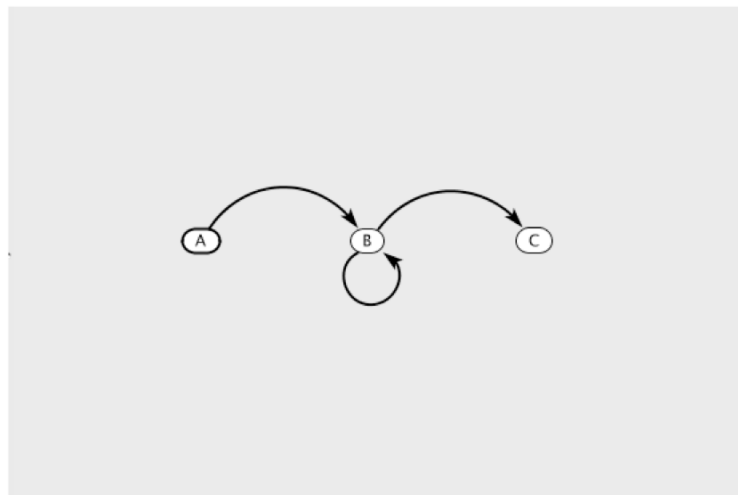
Lee 07: 2

An *initial state*:



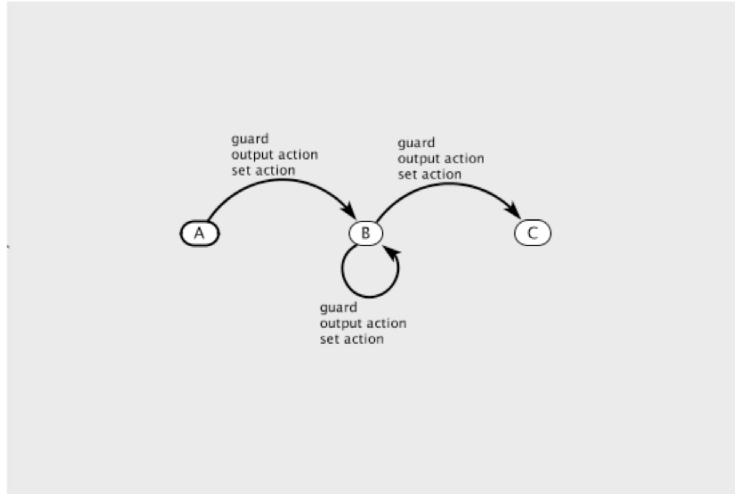
Lee 07: 3

A collection of *transitions*:



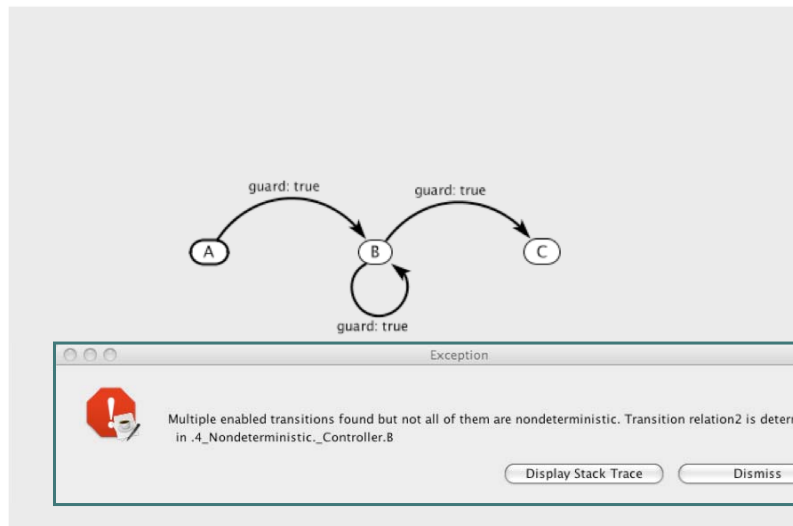
Lee 07: 4

Transitions have *labels*:



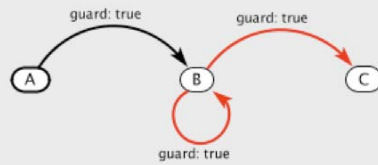
There are many variants of state machines, each giving different labels and semantics associated with those labels. Since we are interested in concurrent composition of state machines, we will give our state machines explicit inputs and outputs, and the labels will refer to these (reading and writing them). Lee 07: 5

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. Lee 07: 6

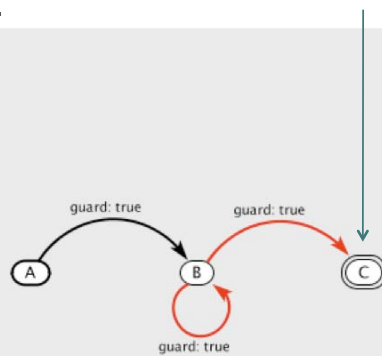
Nondeterministic State Machines



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.

Lee 07: 7

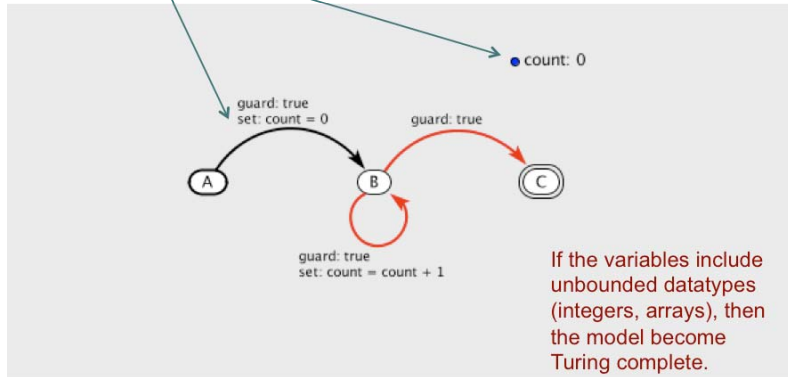
Final states:



This model stops executing when it reaches state C.

Lee 07: 8

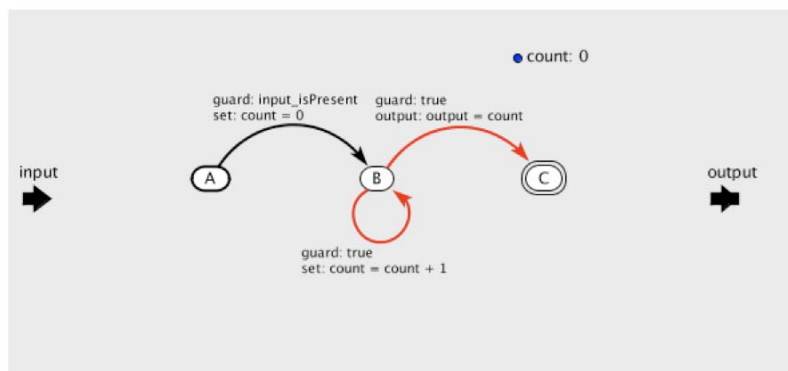
Extended State Machines can operate on variables:



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

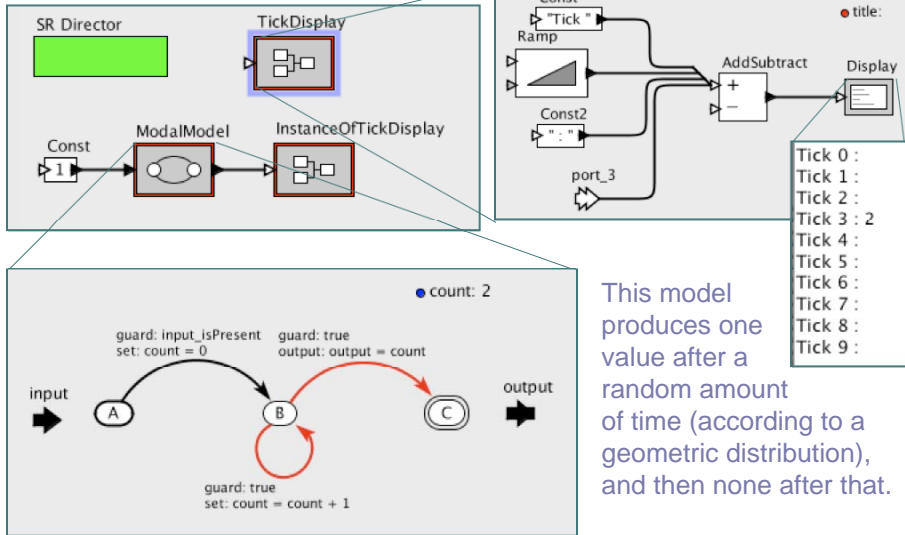
I/O Automata



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

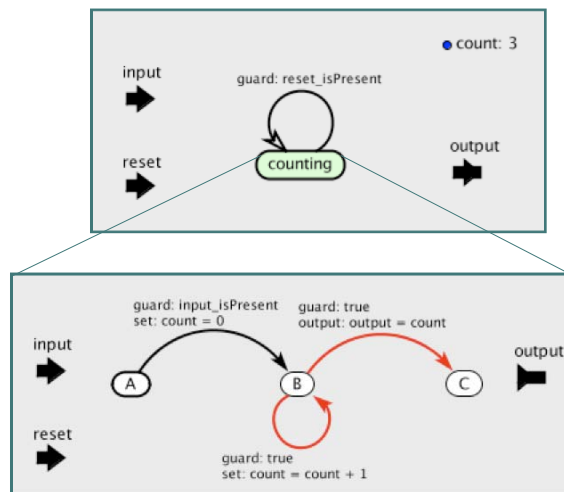


Lee 07: 11

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.



Lee 07: 12

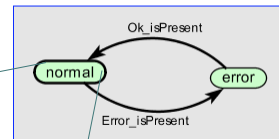
Discussion

Hierarchy is only syntactic sugar.

How much syntax does it affect?

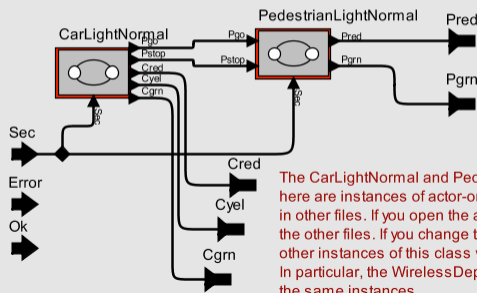
Lee 07: 13

Modal Models



SR Director

The NormalC actor generates the control signals for the car stoplights under normal operating conditions. The NormalP actor reacts to these controls to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation.



The CarLightNormal and PedestrianLightNormal actors here are instances of actor-oriented classes defined in other files. If you open the actors, you will open the other files. If you change the design, then all other instances of this class will see the change. In particular, the WirelessDeployment example uses the same instances.

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

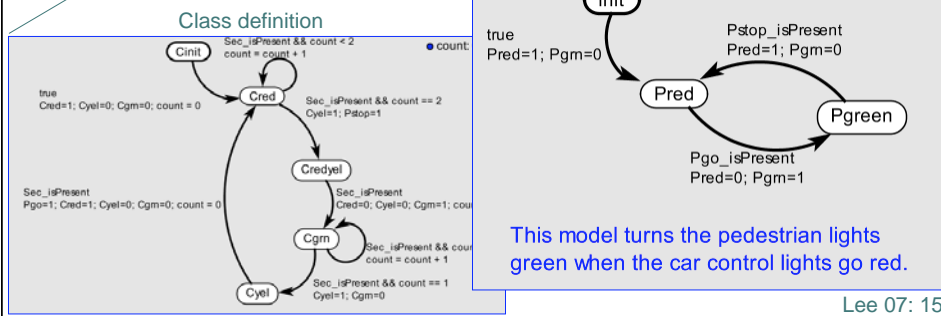
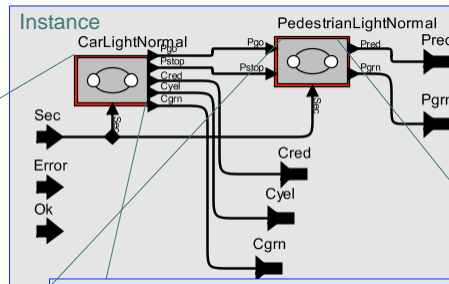
A state may refine to another state machine or to a concurrent model.

Lee 07: 14

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.



Lee 07: 15

Background on Concurrent State Machines

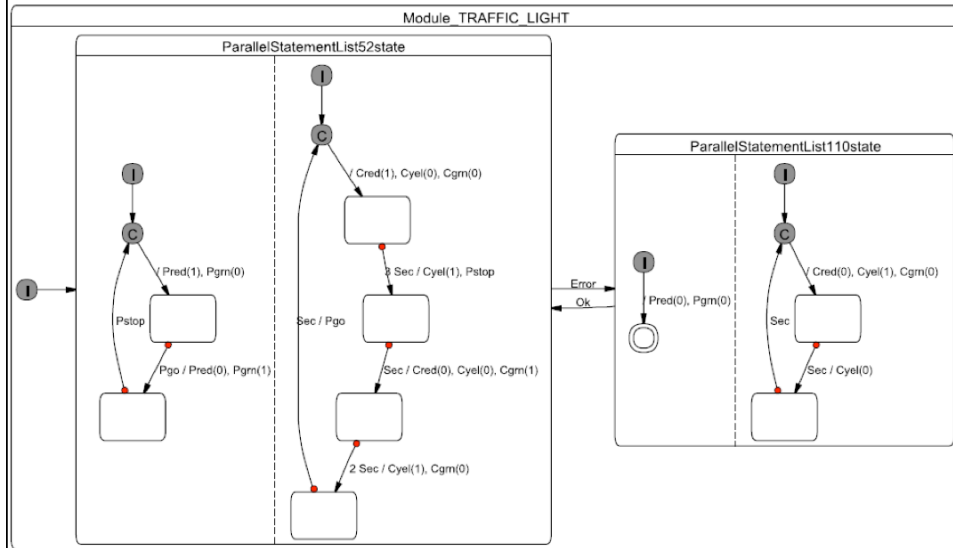
- Statecharts [Harel 87]
- I/O Automata [Lynch 87]
- Esterel [Berry 92]
- SyncCharts [André 96]
- *Charts [Girault, Lee, Lee 99]
- Safe State Machine (SSM) [André 03]
- SCADE [Berry 03]

Lee 07: 16

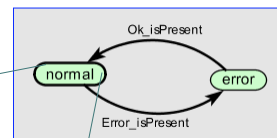
Simple Traffic Light Example in Statecharts

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

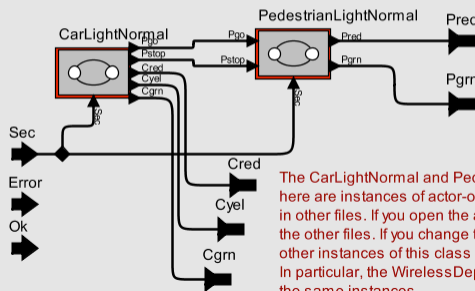


Traffic Light Example in Ptolemy II



SR Director

The NormalC actor generates the control signals for the car stoplights under normal operating conditions. The NormalP actor reacts to these controls to generate the control signals for the pedestrian lights. Look inside each actor to see its implementation.



The CarLightNormal and PedestrianLightNormal actors here are instances of actor-oriented classes defined in other files. If you open the actors, you will open the other files. If you change the design, then all other instances of this class will see the change. In particular, the WirelessDeployment example uses the same instances.

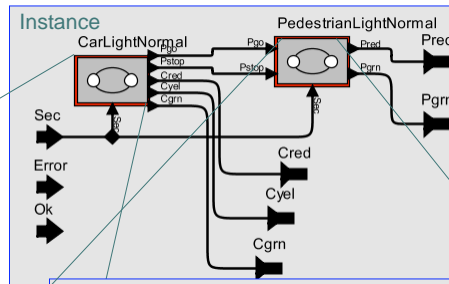
Whereas Statecharts lumps together the state machine semantics and the concurrency model, Ptolemy II separates these.

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

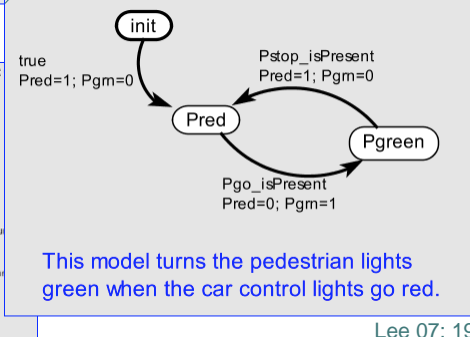
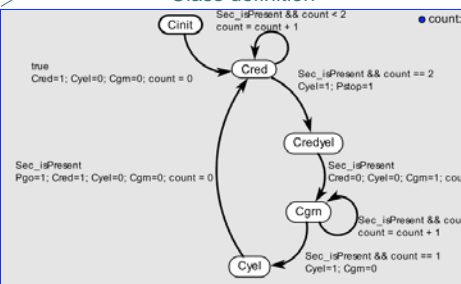
Lee 07: 18

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.



Class definition



Lee 07: 19

Syntax Comparisons between Statecharts and Ptolemy II

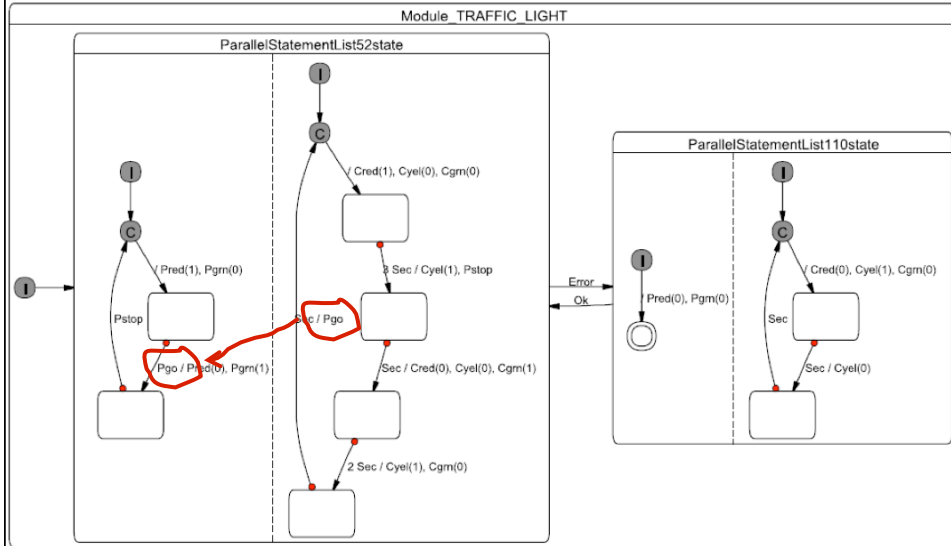
The Ptolemy II model and the Statecharts model differ in syntax. Some issues to consider when evaluating a syntax:

- 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").

Lee 07: 20

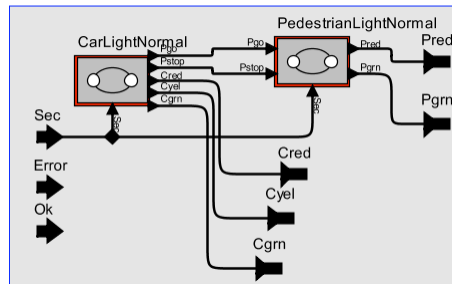
Simple Traffic Light Example in Statecharts, from Reinhard von Hanxleden, Kiel University
Case study for Ptolemy II Design

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?



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).

Lee 07: 23

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*.

Lee 07: 24