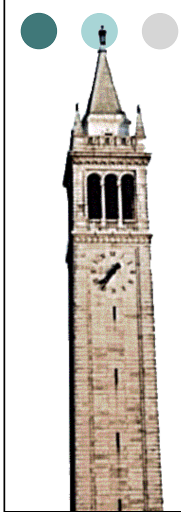


Concurrent Models of Computation

Edward A. Lee

Robert S. Pepper Distinguished Professor, UC Berkeley

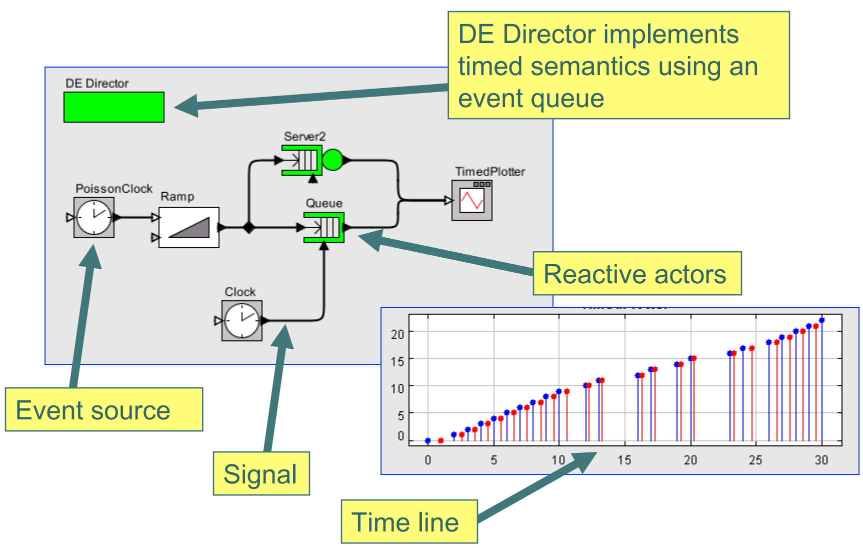
EECS 219D

Concurrent Models of Computation

Fall 2011

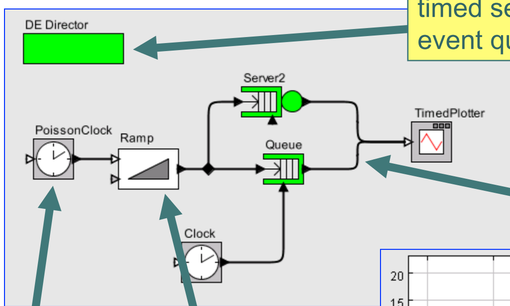
Week 12: Discrete-Event Systems

Discrete Event Models

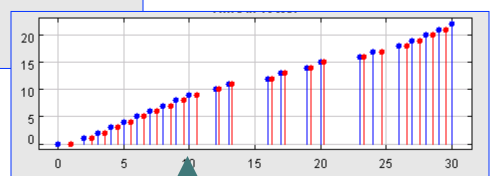


Discrete Events (DE): A Timed Concurrent Model of Computation

DE Director implements timed semantics using an event queue



Actors communicate via "signals" that are marked point processes (discrete, valued, events in time).



Actor (in this case, source of events)

Reactive actors produce output events in response to input events

Plot of the value (marking) of a signal as function of time.

Our Applications of DE

Modeling and simulation of

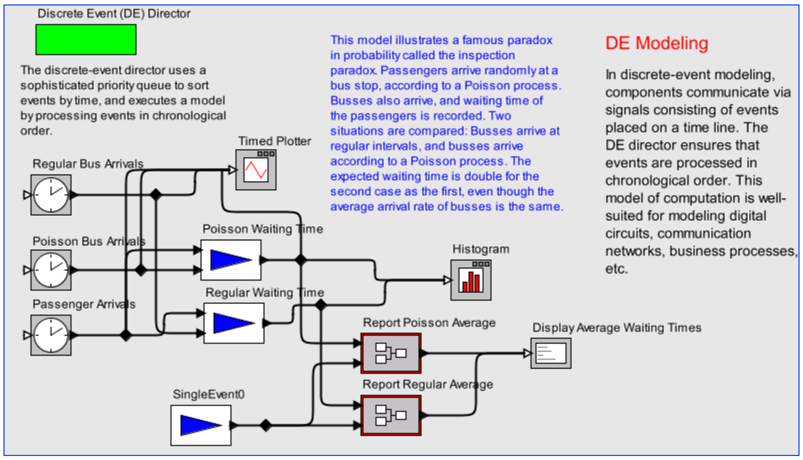
- Communication networks (mostly wireless)
- Hardware architectures
- Systems of systems

Design and software synthesis for

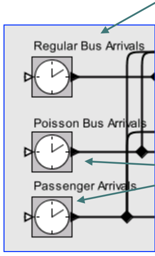
- Sensor networks
- Distributed real-time software
- Hardware/software systems

Design of Discrete-Event Models

Example: Model of a transportation system:

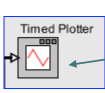


Event Sources and Sinks



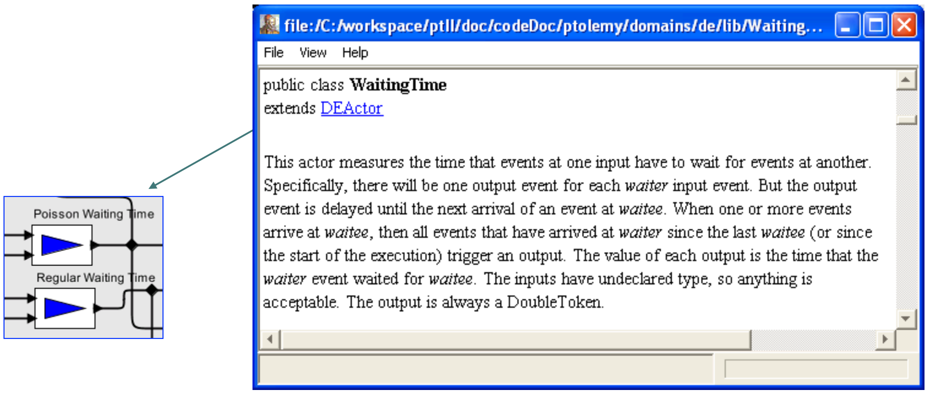
The Clock actor produces events at regular intervals. It can repeat any finite pattern of event values and times.

The PoissonClock actor produces events at random intervals. The time between events is given by an exponential random variable. The resulting output random process is called a Poisson process. It has the property that at any time, the expected time until the next event is constant (this is called the *memoryless* property because it makes no difference what events have occurred before that time).



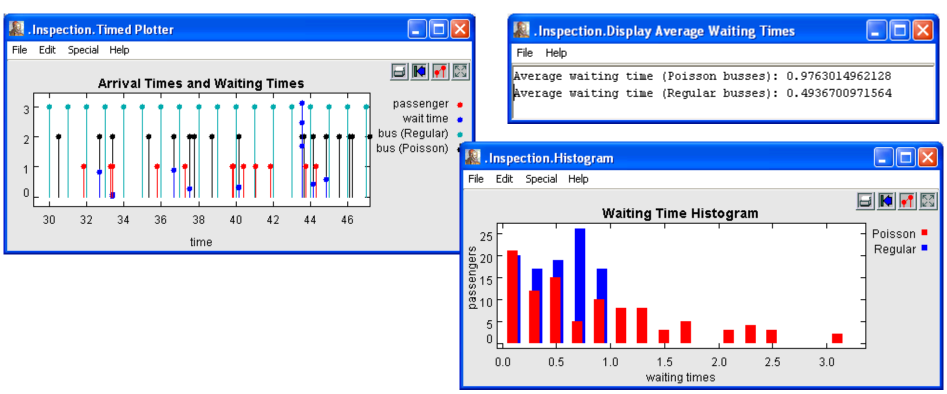
The TimedPlotter actor plots double-valued events as a function of time.

Actors that Use Time



Execution of the Transportation System Model

These displays show that the average time that passengers wait for a bus is smaller if the busses arrive at regular intervals than if they arrive random intervals, even when the average arrival rate is the same. This is called the *inspection paradox*.



Uses for Discrete-Event Modeling

- Modeling timed systems
  - transportation, commerce, business, finance, social, communication networks, operating systems, wireless networks, …
- Designing digital circuits
  - VHDL, Verilog
- Designing real-time software
  - Music systems (Max, …)

Using DE to Model Real-Time Software

Consider a real-time program on an embedded computer that is connected to two sensors *A* and *B*, each providing a stream of data at a normalized rate of one sample per time unit (exactly). The data from the two sensors is deposited by an interrupt service routine into a register.

Assume a program that looks like this:

```
while(true) {
    wait for new data from A;
    wait a fixed amount of time T;
    observe registered data from B;
    average data from A and B;
}
```

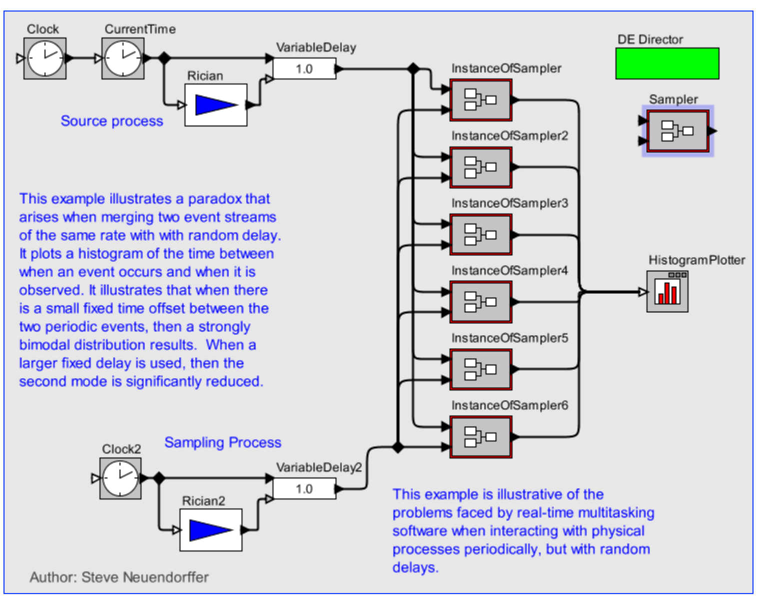
The Design Question

Assume that there are random delays in the software (due to multitasking, interrupt handling, cache management, etc.) for both the above program and the interrupt service routines.

What is the best choice for the value for T?

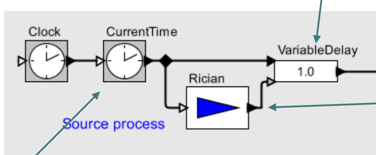
One way to frame the question: How old is the data from *B* that will be averaged with the data from *A*?

A Model that Measures for Various Values of T



Modeling Random Delay in Sensor Data

Given an event with time stamp t on the upper input, the VariableDelay actor produces an output with the same value but time stamp $t + t'$, where t' is the value of the most recently seen event on the lower input.

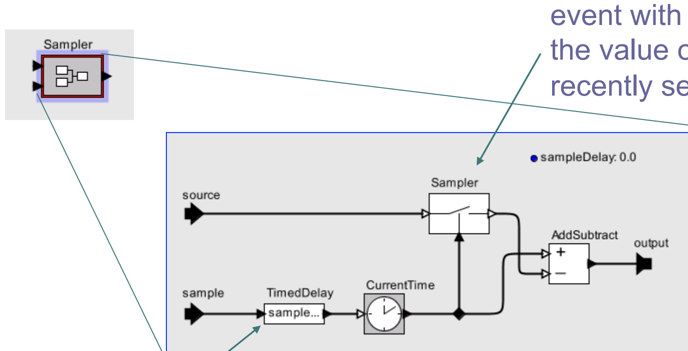


The Rician actor, when triggered, produces an output event with a non-negative random value and with time stamp equal to that of the trigger event.

Given an input event at time t with any value, the CurrentTime actor outputs the double t with time stamp t.

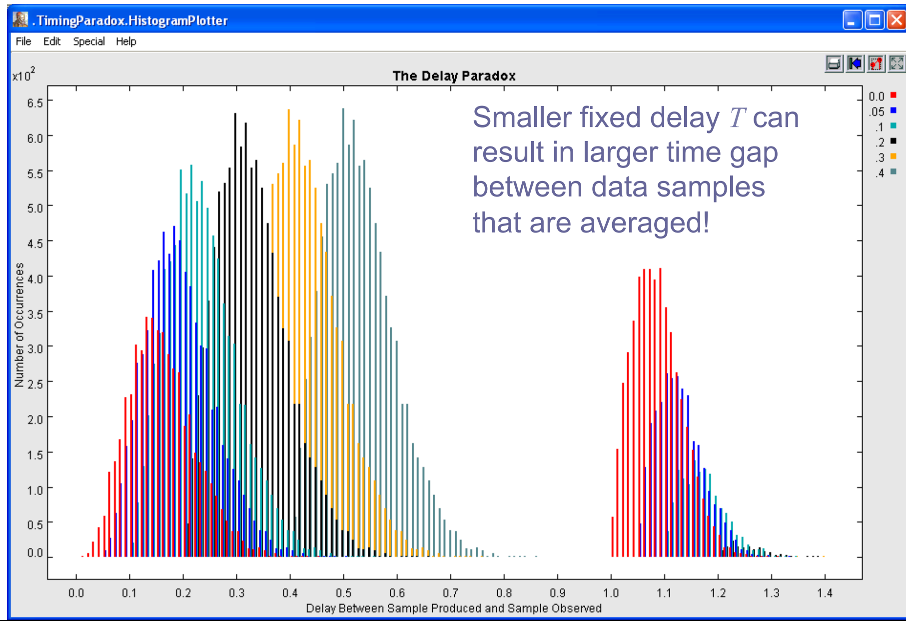
Actor-Oriented Sampler Class

Given a trigger event with time stamp t the Sampler actor produces an output event with value equal to the value of the most recently seen input event.



The TimedDelay actor transfers every input event to the output with a fixed increment in the time stamp. Here, the value is sampleDelay, a parameter of the composite actor.

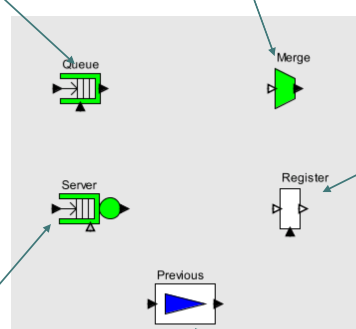
Result of Executing this Model



Design in DE: Some Useful Actors

When a token is received on the input port, it is stored in the queue. When the trigger port receives a token, the oldest element in the queue is output. If there is no element in the queue when a token is received on the trigger port, then no output is produced.

Merge is deterministic in DE.



Like a register in digital circuits.

Like the Queue, except that a *serviceTime* parameter provides a lower bound on the time between outputs.

When triggered by an input, output the previous input. Is this useful in feedback loops?

Signals in DE

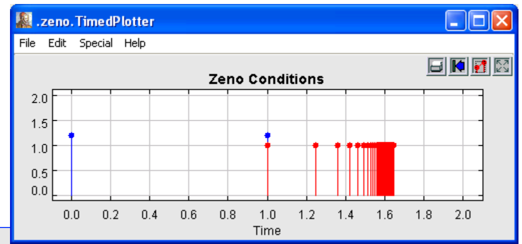
A signal in DE is a partial function $a : T \to A$, where A is a set of possible event values (a data type and an element indicating "absent"), and T is a totally ordered set of *tags* that represent *time stamps* and ordering of events at the same time stamp.

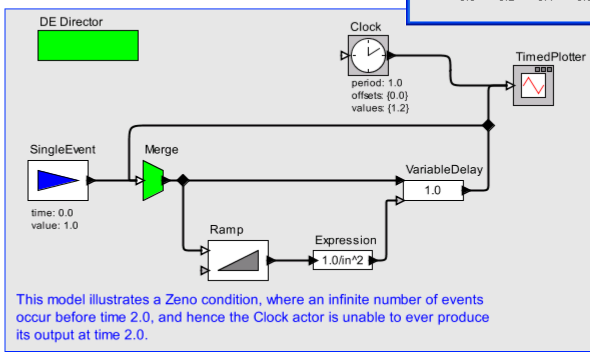
In a DE model, all signals share the same domain T, but they may have different ranges A.

Executing Discrete Event Systems

- Maintain an *event queue*, which is an ordered set of events.
- Process the least event in the event queue by sending it to its destination port and firing the actor containing that port.
- Questions:
  - How to get fast execution when there are many events in the event queue…
  - What to do when there are multiple simultaneous events in the event queue…

Zeno Signals





Eventually, execution stops advancing time. Why?

Note that if the Ramp is set to produce integer outputs, then eventually the output will overflow and become negative, which will cause an exception.

Taking Stock

- The discrete-event model of computation is useful for modeling and design of time-based systems.
- In DE models, signals are time-stamped events, and events are processed in chronological order.
- Simultaneous events and Zeno conditions create subtleties that the semantics will have to deal with.

First Attempt at a Model for Signals

Let $\mathbb{R}_+$ be the non-negative real numbers. Let V be an arbitrary family of values (a data type, or alphabet). Let

$$V_\varepsilon = V \cup \{\varepsilon\}$$

be the set of values plus "absent." Let s be a signal, given as a partial function:

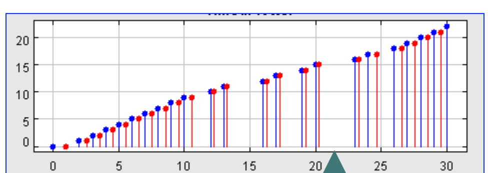
$$s\colon \mathbb{R}_+ \rightharpoonup V_\varepsilon$$

defined on an initial segment of $\mathbb{R}_+$

First Attempt at a Model for Signals

$$s\colon \mathbb{R}_+ \rightharpoonup V_\varepsilon$$

Values V

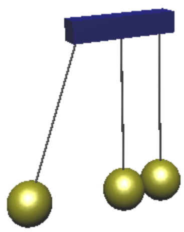


Initial segment $I \subseteq \mathbb{R}_+$ where the signal is defined.

Absent: $s(\tau) = \varepsilon$ for almost all $\tau \in I$.

This model is not rich enough because it does not allow a signal to have multiple events at the same time.

Example Motivating the Need for Simultaneous Events Within a Signal



Newton's Cradle:

Steel balls on strings

Collisions are events

Momentum of the middle ball has three values at the time of collision.

This example has continuous dynamics as well

(I will return to this)

Other examples:

- Batch arrivals at a queue.
- Software sequences abstracted as instantaneous.
- Transient states.

A Better Model for Signals: *Super-Dense Time*

Let $T = \mathbb{R}_+ \times \mathbb{N}$ be a set of "tags" where $\mathbb{N}$ is the natural numbers, and give a signal s as a partial function:

$$s : T \rightharpoonup V_\varepsilon$$

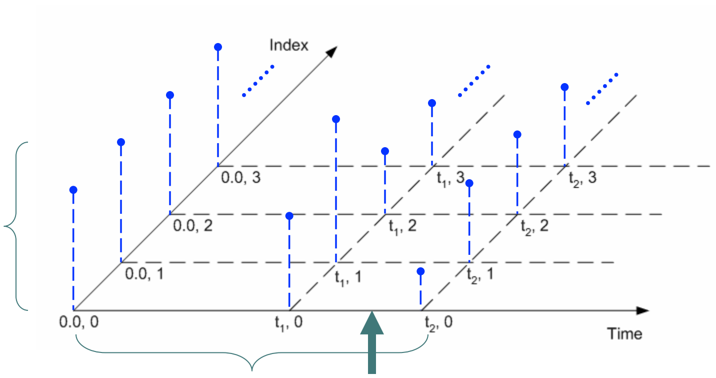
defined on an initial segment of T, assuming a lexical ordering on T:

$$(t_1, n_1) \le (t_2, n_2) \iff t_1 < t_2, \text{ or } t_1 = t_2 \text{ and } n_1 \le n_2 \, .$$

This allows signals to have a sequence of values at any real time t.

Super Dense Time

Values V



Initial segment $I \subseteq \mathbb{R}_+ \times \mathbb{N}$ where the signal is defined

Absent: $s(\tau) = \varepsilon$ for almost all $\tau \in I$.

Events and Firings

$$s : T \rightharpoonup V_\varepsilon$$

- A *tag* is a time-index pair, $\tau = (t, n) \in T = \mathbb{R}_+ \times \mathbb{N}$.
- An *event* is a tag-value pair, $e = (\tau, v) \in T \times V$.
- $s(\tau)$ is an event if $s(\tau) \neq \varepsilon$.

Operationally, events are processed by presenting all input events at a tag to an actor and then *firing* it.

However, this is not always possible!

Discrete Signals

A signal s is *discrete* if there is an *order embedding* from its tag set $\pi(s)$ (the tags for which it is defined and not abent) to the integers (under their usual order).

A system S (a set of signals) is *discrete* if there is an *order embedding* from its tag set $\pi(s)$ to the integers (under their usual order).

Terminology: Order Embedding

Given two posets A and B, an *order embedding* is a function $f : A \to B$ such that for all $a, a' \in A$,

$$a \le a' \iff f(a) \le f(a')$$

Exercise: Show that if A and B are two posets, and $f : A \to B$ is an order embedding, then f is *one-to-one.*

Examples

1. Suppose we have a signal s whose tag set is

   $$\{(\tau, 0) \mid \tau \in R\}$$

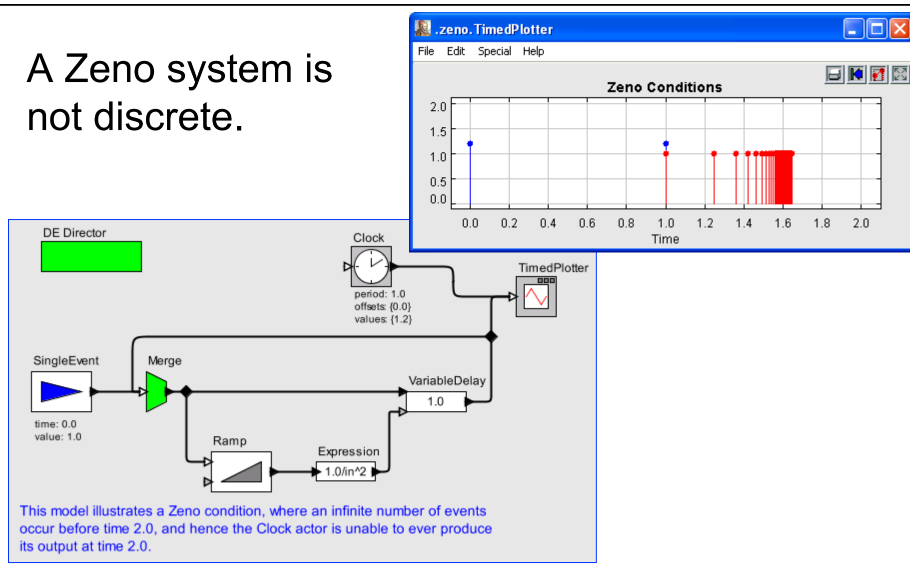
   (this is a *continuous-time* signal). This signal is not discrete.

2. Suppose we have a signal s whose tag set is

   $$\{(\tau, 0) \mid \tau \in Rationals\}$$

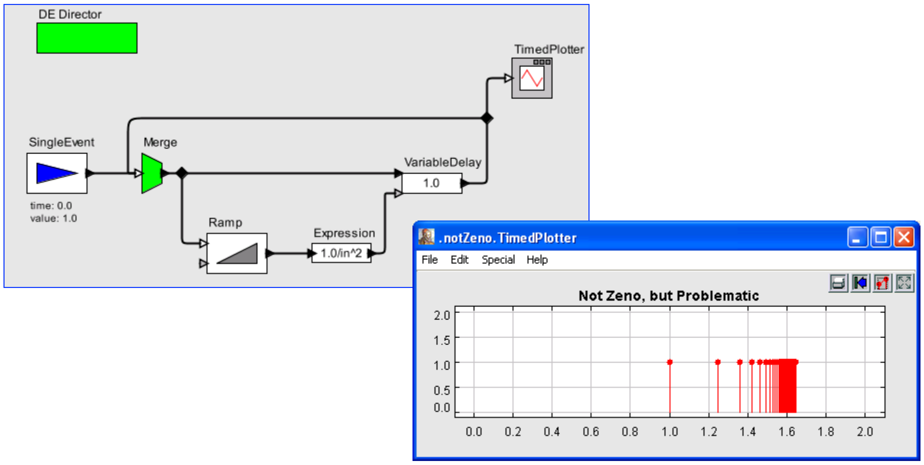
   This signal is also not discrete.

A Zeno system is not discrete.



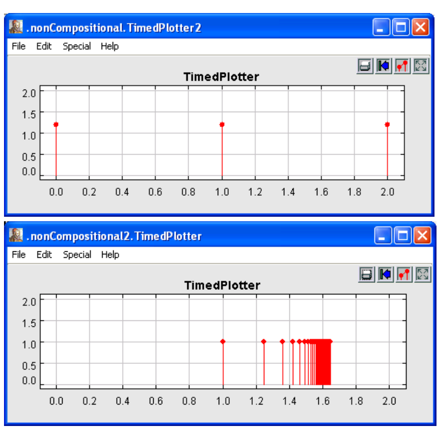
The tag set here includes $\{0, 1, 2, \ldots\}$ and $\{1, 1.25, 1.36, 1.42, \ldots\}$.
Exercise: Prove that this system is not discrete.

Is the following system discrete?



Discreteness is Not a Compositional Property

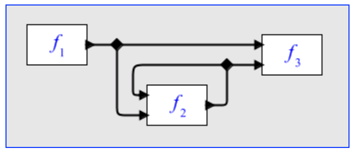
Given two discrete signals s, s' it is not necessarily true that $S = \{ s, s' \}$ is a discrete system.



Putting these two signals in the same model creates a Zeno condition.

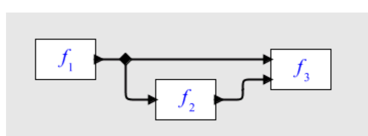
Question 1:

Can we find necessary and/or sufficient conditions to avoid Zeno systems?



Question 2:

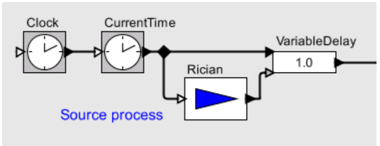
In the following model, if f_2 has no delay, should f_3 see two simultaneous input events with the same tag? Should it react to them at once, or separately?



In Verilog, it is nondeterministic. In VHDL, it sees a sequence of two distinct events separated by "delta time" and reacts twice, once to each input. In the Ptolemy II DE domain, it sees the events together and reacts once.

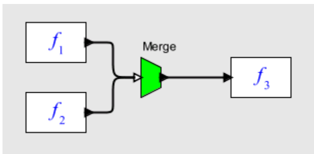
Example

In the following segment of a model, clearly we wish that the VariableDelay see the output of Rician when it processes an input from CurrentTime.



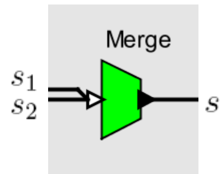
Question 3:

What if the two sources in the following model deliver an event with the same tag? Can the output signal have distinct events with the same tag?



Recall that we require that a signal be a partial function $s : T \to V$, where V is a set of possible event values (a data type), and T is a totally ordered set of *tags*.

One Possible Semantics for DE Merge



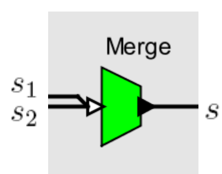
At time t, input sequences are interleaved. That is, if the inputs are s_1 and s_2 and

$$s_1(t,0) = v_1,$$
$$s_2(t,0) = w_1, \quad s_1(t,1) = w_2$$

(otherwise absent) then the output s is

$$s(t,0) = v_1, \quad s(t,1) = w_1, \quad s(t,2) = w_2.$$

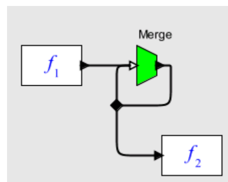
Implementation of DE Merge




```
private List pendingEvents;
fire() {
  foreach input s {
    if (s is present) {
      pendingEvents.append(event from s);
    }
  }
  if (pendingEvents has events) {
    send to output (pendingEvents.first);
    pendingEvents.removeFirst();
  }
  if (pendingEvents has events) {
    post event at the next index on the event queue;
  }
}
```

Question 4:

What does this mean?



The Merge presumably does not introduce delay, so what is the meaning of this model?

Conclusions

- *Discrete-event* models compose components that communicate timed *events*. They are widely used for simulation (of hardware, networks, and complex systems).
- *Superdense time* uses tags that have a real-valued *time-stamp* and a natural number *index*, thus supporting sequences of causally-related simultaneous events.
- A *discrete system* is one where the there is an order embedding from the set of tags in the system to the integers.