EECS 144/244: Fundamental Algorithms for System Modeling, Analysis, and Optimization

Timed Systems

Lecture: Timed Discrete Event Systems

Stavros Tripakis

University of California, Berkeley

Recap: Views on systems

	operational	denotational
monolithic	automata, machines, transition systems, ...	functions (on values, streams, signals, ...), equations
compositional	products of automata, etc.	function composition, sets of equations, fixpoints

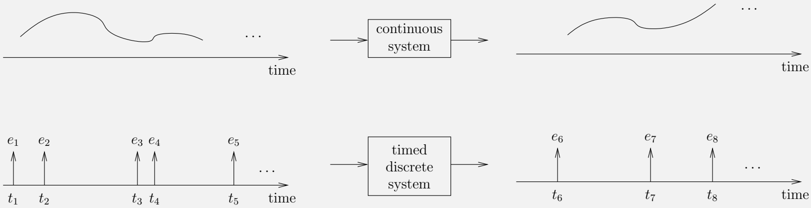
From Continuous to Timed Discrete Event Systems

- Continuous systems: functions on **continuous signals**.
  Continuous signal x = continuous function of dense time ($\mathbb{R}_+$)

  $$x : \mathbb{R}_+ \to V$$

  $x(t)$: value of x at time t; belongs to some set of values V (e.g., $\mathbb{R}$)

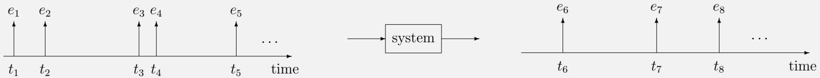
- Timed Discrete Event Systems: deal with **timed discrete-event signals**.
  Timed discrete-event signal: sequence of timed events.



Timed Discrete Event Systems

Main notion: **event**

- something occurring at some point in time
- may also carry a value
- time could be discrete/"logical" ($\mathbb{N}$), continuous/dense ($\mathbb{R}_+$), or even "superdense" ($\mathbb{R}_+ \times \mathbb{N}$)
- systems are viewed as consumers/producers of event streams



Tagged signals

In the *tagged signal model* (TSM) [Lee and Sangiovanni-Vincentelli(1998)], a *signal* s is a set of events with timestamps:

$$s \subseteq T \times V$$

where T is a set of tags (think timestamps) and V is a set of values.

Continuous and (timed) discrete(-event) signals

Let $T = \mathbb{R}_+$, the set of non-negative reals.

Let

$$s \subseteq \mathbb{R}_+ \times V$$

be a signal.

Let $t(s)$ be the set of all tags of signal s:

$$t(s) = \{t \mid \exists(t, v) \in s\}$$

Then, according to the TSM:

- s is *continuous* if $t(s) = \mathbb{R}_+$.
- s is *discrete* if $t(s)$ is *order-isomorphic* to a subset of $\mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, ...\}$ is the set of natural numbers.

Order-isomorphisms

A order-isomorphic B means there is an order-preserving bijection $f : A \to B$.

In our case the order is the usual $<$ on $\mathbb{R}_+$ and $\mathbb{N}$. So:

- f must be a bijection: (1) $f(a)$ must be defined for all $a \in A$, (2) for all $b \in B$ there must exist $a \in A$ such that $f(a) = b$, and (3) $a \neq a' \Rightarrow f(a) \neq f(a')$.
- f must be order-preserving: $a < a' \Rightarrow f(a) < f(a')$.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n,v) \mid n \in \mathbb{N}, v \in V\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n, v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n, v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0, 0), (1, 1,), (2, 2)\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n, v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0, 0), (1, 1,), (2, 2)\}$? Discrete.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n, v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0, 0), (1, 1,), (2, 2)\}$? Discrete.

$s_4 = \{(0.3, 0), (1.27, 1,), (2\pi, 2)\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n,v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0,0),(1,1,),(2,2)\}?$ Discrete.

$s_4 = \{(0.3,0),(1.27,1,),(2\pi,2)\}?$ Discrete.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n,v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0,0), (1,1,), (2,2)\}$? Discrete.

$s_4 = \{(0.3,0), (1.27,1,), (2\pi,2)\}$? Discrete.

$s_5 = \{\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n, v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0, 0), (1, 1,), (2, 2)\}?$ Discrete.

$s_4 = \{(0.3, 0), (1.27, 1,), (2\pi, 2)\}?$ Discrete.

$s_5 = \{\}$? Discrete.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n,v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0,0),(1,1,),(2,2)\}?$ Discrete.

$s_4 = \{(0.3,0),(1.27,1,),(2\pi,2)\}?$ Discrete.

$s_5 = \{\}$? Discrete.

$s_6 = \{(t,t) \mid t \in [0,1]\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n,v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0,0),(1,1,),(2,2)\}$? Discrete.

$s_4 = \{(0.3,0),(1.27,1,),(2\pi,2)\}$? Discrete.

$s_5 = \{\}$? Discrete.

$s_6 = \{(t,t) \mid t \in [0,1]\}$? Neither, according to the TSM definition. Could be considered a continuous signal on the interval $[0,1]$.

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t, t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n, v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0, 0), (1, 1,), (2, 2)\}$? Discrete.

$s_4 = \{(0.3, 0), (1.27, 1,), (2\pi, 2)\}$? Discrete.

$s_5 = \{\}$? Discrete.

$s_6 = \{(t, t) \mid t \in [0, 1]\}$? Neither, according to the TSM definition. Could be considered a continuous signal on the interval $[0, 1]$.

$s_7 = \{(t, \lfloor t \rfloor) \mid t \in \mathbb{R}_+\}$?

Continous and discrete signals

Are the signals below discrete or continuous?

$s_1 = \{(t,t) \mid t \in \mathbb{R}_+\}$? Continuous.

$s_2 = \{(n,v) \mid n \in \mathbb{N}, v \in V\}$? Discrete.

$s_3 = \{(0,0),(1,1,),(2,2)\}$? Discrete.

$s_4 = \{(0.3,0),(1.27,1,),(2\pi,2)\}$? Discrete.

$s_5 = \{\}$? Discrete.

$s_6 = \{(t,t) \mid t \in [0,1]\}$? Neither, according to the TSM definition. Could be considered a continuous signal on the interval $[0,1]$.

$s_7 = \{(t,\lfloor t \rfloor) \mid t \in \mathbb{R}_+\}$? Continuous, according to the TSM definition. It could also be considered discrete since it's piecewise constant. This is how discrete signals are modeled in Simulink.

DIGRESSION:

EVENTS and STATES

From States to Events

If my formalism only has the notion of state, can I define events?

From States to Events

If my formalism only has the notion of state, can I define events?

Event = change of state

From States to Events

If my formalism only has the notion of state, can I define events?

Event = change of state

Example: Lustre program

```
node UpwardEdge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

From States to Events

If my formalism only has the notion of state, can I define events?

Event = change of state

Example: Lustre program

```
node UpwardEdge (X : bool) returns (E : bool);
let
  E = false -> X and not pre X ;
tel
```

- In a Kripke structure, every transition (except perhaps self-loops) is in principle an event.
- We may also choose to observe changes in only some variables.

From Events to States

If my formalism only has events as primitives, can I define state?

From Events to States

If my formalism only has events as primitives, can I define state?

State = history of events observed so far

From Events to States

If my formalism only has events as primitives, can I define state?

State = history of events observed so far
Formally:

- Σ: set of events
- Σ^*: set of finite event sequences = histories
- Every $s \in \Sigma^*$ can be seen as a state

From Events to States

If my formalism only has events as primitives, can I define state?

State = history of events observed so far
Formally:

- Σ: set of events
- Σ^*: set of finite event sequences = histories
- Every $s \in \Sigma^*$ can be seen as a state

C.f. a famous theorem:

Theorem (Myhill-Nerode theorem)

A language $L \subseteq \Sigma^*$ is regular iff the equivalence relation over words

$$s \sim_L s' \quad \widehat{=} \quad \forall s'' \in \Sigma^* : s \cdot s'' \in L \Leftrightarrow s' \cdot s'' \in L$$

has a finite set of equivalence classes. The number of equivalence classes of $\sim_L$ is the number of states in the smallest DFA recognizing L.

END DIGRESSION

What about timed events?

- "Logical time" (order of events) is already modeled in discrete systems
- Discrete time can also be modeled:

What about timed events?

- "Logical time" (order of events) is already modeled in discrete systems
- Discrete time can also be modeled: using counters

Example: a watchdog that checks that an event happens within a deadline

```
node Watchdog (X : bool, D : int) returns (W : bool);
let
  counter = 0 -> (pre counter) + 1;
  seenX = X or (false -> pre seenX);
  W = (not seenX) and (counter > D);
tel
```

What about timed events?

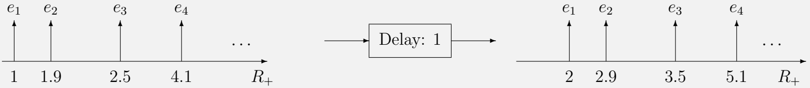
- "Logical time" (order of events) is already modeled in discrete systems
- Discrete time can also be modeled: using counters

Example: a watchdog that checks that an event happens within a deadline

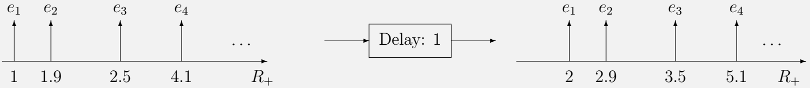
```
node Watchdog (X : bool, D : int) returns (W : bool);
let
  counter = 0 -> (pre counter) + 1;
  seenX = X or (false -> pre seenX);
  W = (not seenX) and (counter > D);
tel
```

Homework: how can you model discrete time in Spin?

Dense-Time Delay

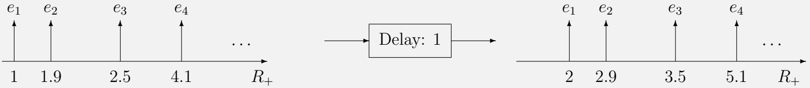


Dense-Time Delay



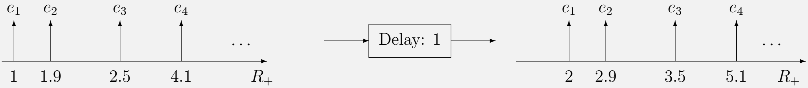
- Can this system be modeled in discrete time?

Dense-Time Delay



- Can this system be modeled in discrete time?
- No: must choose some Δt as time step, but input events could arrive closer.

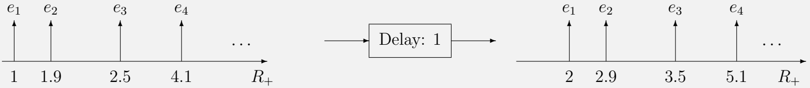
Dense-Time Delay



- Can this system be modeled in discrete time?
- No: must choose some Δt as time step, but input events could arrive closer.

- Is the Delay a realistic component?
  - Can you find a real system that behaves like a delay?

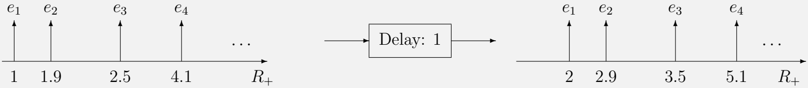
Dense-Time Delay



- Can this system be modeled in discrete time?
- No: must choose some Δt as time step, but input events could arrive closer.

- Is the Delay a realistic component?
  - Can you find a real system that behaves like a delay?
  - (Claim) No: physical systems can handle only limited burstiness.
  - If too many events arrive in too short time, some will be discarded or delayed more than a constant delay.
- Why then use Delay?

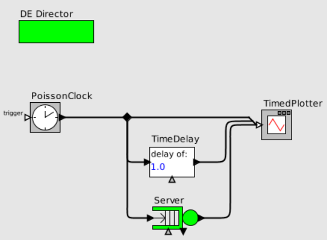
Dense-Time Delay

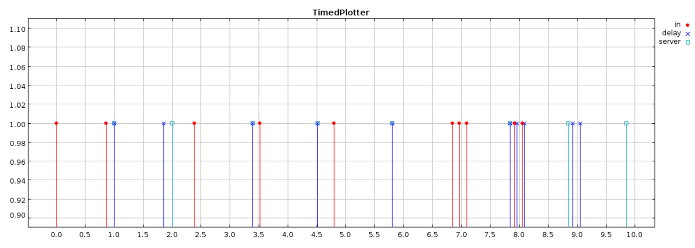


- Can this system be modeled in discrete time?
- No: must choose some Δt as time step, but input events could arrive closer.

- Is the Delay a realistic component?
  - Can you find a real system that behaves like a delay?
  - (Claim) No: physical systems can handle only limited burstiness.
  - If too many events arrive in too short time, some will be discarded or delayed more than a constant delay.
- Why then use Delay?
  - For the same reason Turing machines are a useful abstraction of reality.

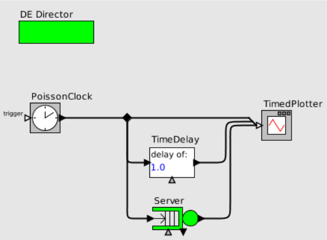
Delay vs. Server





Timed Discrete-Event Models (DE)

Networks of *timed actors*, such as Delay, Server, sources, sinks, ...



Used in many industrial tools (e.g., IBM Rhapsody, Simulink/Stateflow, ...).

Exposition here inspired from Ptolemy DE domain.

Simulation vs. Verification of DE Systems

- We will look at simulation of DE systems.
- Exhaustive verification (model-checking): open research problem!

Simulation vs. Verification

- Verification (by exhaustive state-space exploration): check that *all* possible behaviors of the system satisfy a certain property.
- Simulation: generate *one* possible behavior of the system, and
  - let the user look at it,
  - or have a tool that automatically checks a property (c.f. STL tool Breach).

Simulation vs. Verification

- Verification (by exhaustive state-space exploration): check that *all* possible behaviors of the system satisfy a certain property.
- Simulation: generate *one* possible behavior of the system, and
  - let the user look at it,
  - or have a tool that automatically checks a property (c.f. STL tool Breach).
  - Note: can only generate finite prefixes of infinite behaviors. So simulation often cannot verify exhaustively even 1 behavior.

Simulation vs. Verification

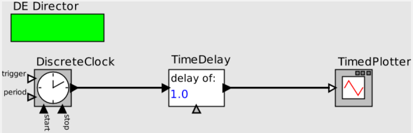
- Verification (by exhaustive state-space exploration): check that *all* possible behaviors of the system satisfy a certain property.
- Simulation: generate *one* possible behavior of the system, and
  - let the user look at it,
  - or have a tool that automatically checks a property (c.f. STL tool Breach).
  - Note: can only generate finite prefixes of infinite behaviors. So simulation often cannot verify exhaustively even 1 behavior.
- Note: industry / different communities use different terms for these methodologies, e.g.:
  - in HW industry:
    - "simulation": generate a behavior and allow user to observe it
    - "verification": generate one or more behaviors (e.g., randomly) and check that each satisfies a property
    - "formal verification": exhaustive or bounded model-checking

Discrete-Event Simulation: Basic Idea

Standard DE simulation scheme (e.g., see [Misra(1986), Banks et al.(2005)Banks, Carson, Nelson, and Nicol]).

$t := 0$; // *initialize simulation time to 0*
initialize global event queue Q with a set of initial events;
                    // *events in Q ordered by timestamp*
while Q is not empty **do**
    remove earliest event $e = (v_e, t_e)$ from Q;
    $t := t_e$; // *advance global time*
    execute event e: update system state, generate possible future events, and add them to Q, ordered by timestamps;
end while

Example: Clock and Delay



Clock period: 0.6
c_i: events generated by Clock
d_i: events generated by Delay

point in algo	t	Q	current event e
after initialization (step 2)	0	$[(c_0, 0), (c_1, 0.6), (c_2, 1.2), ...]$	
after step 4		$[(c_1, 0.6), (c_2, 1.2), ...]$	$(c_0, 0)$
after step 5		$[(c_1, 0.6), (d_0, 1.0), (c_2, 1.2), ...]$	
after step 6	0		
after step 4		$[(d_0, 1.0), (c_2, 1.2), ...]$	$(c_1, 0.6)$
after step 5		$[(d_0, 1.0), (c_2, 1.2), (d_1, 1.6), ...]$	
after step 6	0.6		
after step 4		$[(c_2, 1.2), (d_1, 1.6), ...]$	$(d_0, 1.0)$
after step 5		Q does not change, but something gets printed	
after step 6	1.0		

$\cdots$

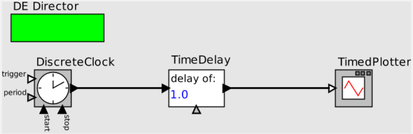
Discrete-Event Simulation: Issues

1: $t := 0$;
2: initialize global event queue Q with a set of initial events;
3: **while** Q is not empty **do**
4: remove earliest event $e = (v_e, t_e)$ from Q;
5: $t := t_e$; *// advance global time*
6: execute event e: update system state, generate possible future events, and add them to Q, ordered by timestamps;
7: **end while**

- Appears intuitive, but details are left unspecified.
- Scheme is not *modular*: no notion of *actor*, step 5 appears to work on the entire system state.
- How to make such a scheme completely modular is still a research topic.
- Also active topic in terms of standards, c.f., the *Functional Mock-up Interface* (FMI) – `https://www.fmi-standard.org/`.

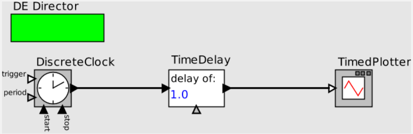
Modeling Source Actors

Source actor = an actor with no inputs.

Clock is a source: 

Modeling Source Actors

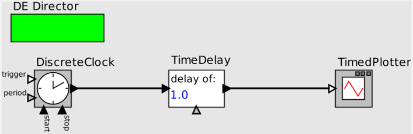
Source actor = an actor with no inputs.

Clock is a source: 

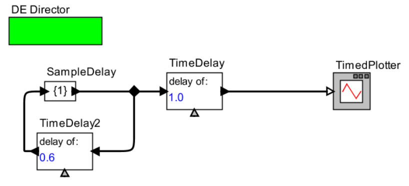
- Option 1 – sources generate all their events at initialization.
    - Simulation time is finite, so presumably only finite number of events.
    - But it may be very large.

Modeling Source Actors

Source actor = an actor with no inputs.

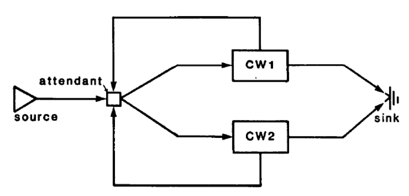
Clock is a source: 

- Option 1 – sources generate all their events at initialization.
  - Simulation time is finite, so presumably only finite number of events.
  - But it may be very large.
- Option 2 – model sources using feedback loops with initial events:



In general: feedback loops are very useful

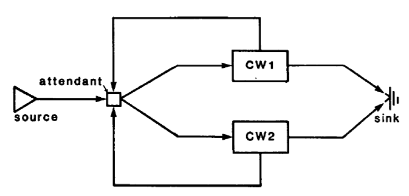
Example: car wash (taken from [Misra(1986)]):



- Source generates car arrivals at some arbitrary times (e.g., at times 3, 8, 9, 14, 16, 22)
- Attendant directs cars to car wash stations CW1 or CW2:
  - if both CW1 and CW2 are free, then to CW1
  - if only one is free, then to this one
  - otherwise car waits until a station becomes free
  - cars are served by attendant in FIFO order
- CW1 (a server actor) spends 8 mins to wash a car
- CW2 (a server actor) spends 10 mins to wash a car

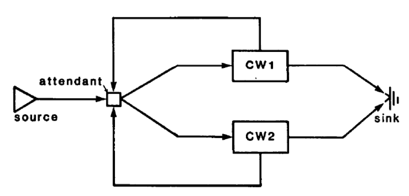
Issues with Feedback Loops

Can you see any problems with the feedback loops of this example?

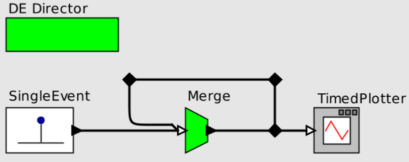


Issues with Feedback Loops

Can you see any problems with the feedback loops of this example?

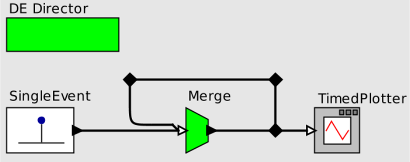


What about this example?



Zeno Systems

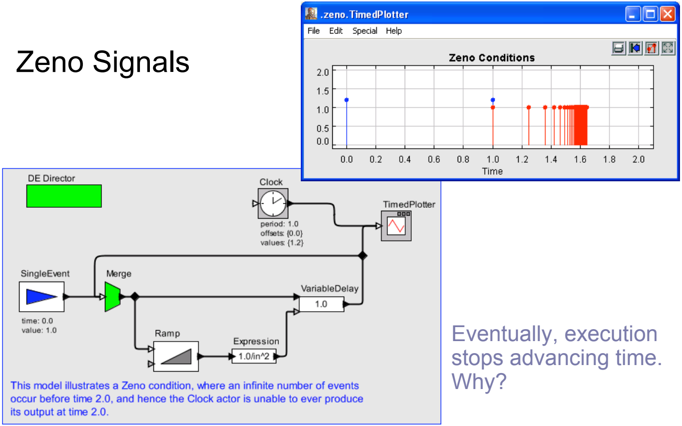
A system is "zeno" if it generates an infinite number of events in a finite amount of time.



This system is zeno.

Another Zeno System

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.

Lee 12: 19

Slide from Edward Lee.

Zeno

Zeno

Zeno's "Achilles and the tortoise" paradox:

- Achilles and the tortoise enter a race. Achilles runs of course much faster. He graciously allows the tortoise a head start of 1 meter. Who will win?

Zeno

Zeno's "Achilles and the tortoise" paradox:

- Achilles and the tortoise enter a race. Achilles runs of course much faster. He graciously allows the tortoise a head start of 1 meter. Who will win?

  "In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead."

(from wikipedia)

Zeno signals in the TSM

Let

$$s = \{(1, \_), (1.5, \_), (1.75, \_), ...\} \cup \{(2, \_)\}$$

Is s continuous? Is it discrete?

Zeno signals in the TSM

Let

$$s = \{(1, \_), (1.5, \_), (1.75, \_), ...\} \cup \{(2, \_)\}$$

Is s continuous? Is it discrete?

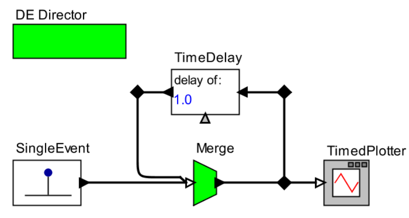
s is not continuous because $t(s) \neq \mathbb{R}_+$.

s is not discrete: there exists a bijection from $t(s)$ to $\mathbb{N}$, but not an order-preserving bijection.

s is a zeno signal.

Avoiding Zeno Systems

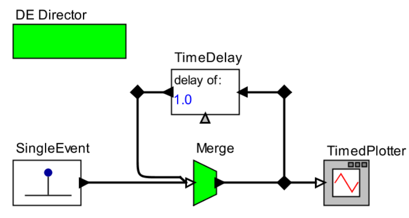
Suppose we add a non-zero delay in every feedback loop:



Is it sufficient to avoid zenoness?

Avoiding Zeno Systems

Suppose we add a non-zero delay in every feedback loop:

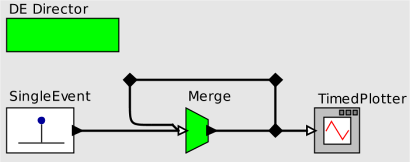


Is it sufficient to avoid zenoness?

Yes: **homework:** prove it.

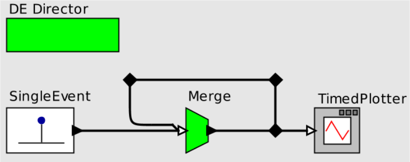
Handling Feedback Loops

- Option 1 – avoid zero-delay loops as above (this also ensures non-zenoness).
- Option 2 – use a fixpoint semantics, as in synchronous systems.
  What would Option 2 give for this example?



Handling Feedback Loops

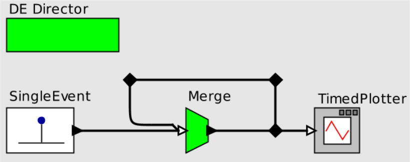
- Option 1 – avoid zero-delay loops as above (this also ensures non-zenoness).
- Option 2 – use a fixpoint semantics, as in synchronous systems.
  What would Option 2 give for this example?



Semantics undefined (unknown values remain at fixpoint).

Handling Feedback Loops

- Option 1 – avoid zero-delay loops as above (this also ensures non-zenoness).
- Option 2 – use a fixpoint semantics, as in synchronous systems.
  What would Option 2 give for this example?



Semantics undefined (unknown values remain at fixpoint).

We will not discuss the fixpoint semantics for DE.
We assume Option 1.

Another Example: Alarm

- Alarm actor: produces event at given time t, unless it receives input at time $t' \leq t$.

Another Example: Alarm

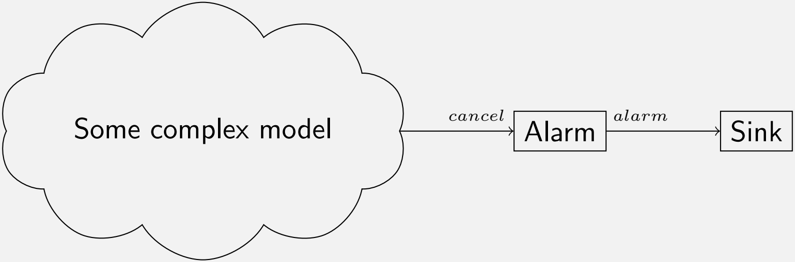
- Alarm actor: produces event at given time t, unless it receives input at time $t' \leq t$.
- How does the DE simulation algorithm handle this example?

Another Example: Alarm

- Alarm actor: produces event at given time t, unless it receives input at time $t' \leq t$.
- How does the DE simulation algorithm handle this example?
- It appears that Alarm should post an initial event with time t

  ... but this event may then have to be **canceled** during simulation if something arrives at the input before t.
- Canceling events = removing them from the event queue.

Another Example: Alarm



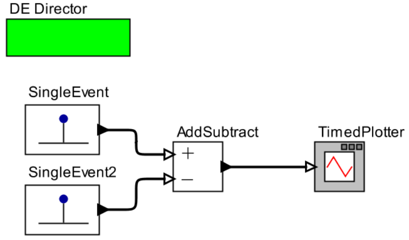
Note:

- Whether an event will be generated before t is not always easy to determine.
- DE algorithm must work independently of how Alarm is connected.
- That's what modular means.

Discrete-Event Simulation – version 2

$t := 0$;
initialize global event queue Q with a set of initial events;
while Q is not empty **do**
    remove earliest event $e = (v_e, t_e)$ from Q;
    $t := t_e$;
    execute event e: update system state, generate possible future events, and add them to Q, ordered by timestamps; possibly remove events from Q;
end while

Issues with Simultaneous Events



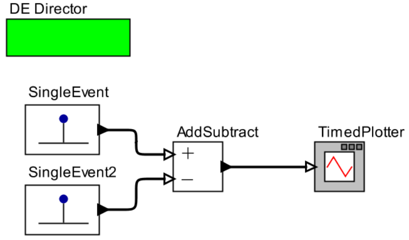
The *AddSubtract* actor is supposed to behave as follows:

- If it receives two simultaneous events, it adds/subtracts their values and produces a *single* event at its output with the resulting value.
- If it receives an event in just one of the two inputs, it simply forwards it.

Suppose the two *SingleEvent* actors produce two simultaneous events.

What should the output be?

Issues with Simultaneous Events



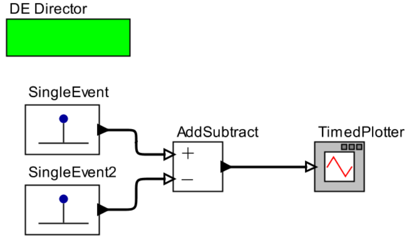
The *AddSubtract* actor is supposed to behave as follows:

- If it receives two simultaneous events, it adds/subtracts their values and produces a *single* event at its output with the resulting value.
- If it receives an event in just one of the two inputs, it simply forwards it.

Suppose the two *SingleEvent* actors produce two simultaneous events.

What should the output be? An event with value $x - x = 0$.

Issues with Simultaneous Events



The *AddSubtract* actor is supposed to behave as follows:

- If it receives two simultaneous events, it adds/subtracts their values and produces a *single* event at its output with the resulting value.
- If it receives an event in just one of the two inputs, it simply forwards it.

Suppose the two *SingleEvent* actors produce two simultaneous events.

What should the output be? An event with value $x - x = 0$.
How to achieve this with the DE simulation algorithm?

Discrete-Event Simulation – version 3

It appears that the DE simulation algorithm must execute *sets* of simultaneous events, instead of one event at a time:

```
t := 0;
initialize global event queue Q with a set of initial events;
while Q is not empty do
    remove ~~earliest event e = (v_e, t_e)~~ set E of all (?) simultaneous
    earliest events from Q;
    t := t_e;
    execute ~~event e~~ set of events E: update system state, generate
    possible future events, and add them to Q, ordered by timestamps;
    possibly remove events from Q;
end while
```

Discrete-Event Simulation – version 3

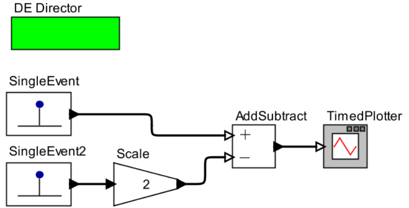
It appears that the DE simulation algorithm must execute *sets* of simultaneous events, instead of one event at a time:

$t := 0$;
initialize global event queue Q with a set of initial events;
while Q is not empty **do**
    remove ~~earliest event~~ $\cancel{e = (v_e, t_e)}$ set E of all (?) simultaneous earliest events from Q;
    $t := t_e$;
    execute ~~event~~ $\cancel{e}$ set of events E: update system state, generate possible future events, and add them to Q, ordered by timestamps; possibly remove events from Q;
end while

Not as simple ...

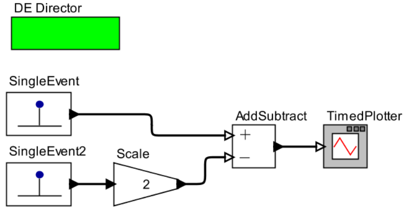
Issues with Simultaneous Events

Suppose the two SingleEvent sources produce two simultaneous events.
Should these be processed together by the algorithm?



Issues with Simultaneous Events

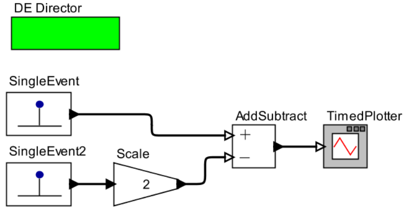
Suppose the two SingleEvent sources produce two simultaneous events.
Should these be processed together by the algorithm?



No: AddSubtract needs to wait for the output of Scale.

Issues with Simultaneous Events

Suppose the two SingleEvent sources produce two simultaneous events.
Should these be processed together by the algorithm?



No: AddSubtract needs to wait for the output of Scale.

Processing a set of simultaneous events E may result in new simultaneous events not in E ...

We need some systematic way to do this ...

Back to the Alarm Example

- Alarm actor: produces event at given time t, unless it receives input at time $t' \leq t$

Back to the Alarm Example

- Alarm actor: produces event at given time t, unless it receives input at time $t' \leq t$
- What if Source produces an event also at time t?

Back to the Alarm Example

- Alarm actor: produces event at given time t, unless it receives input at time $t' \leq t$
- What if Source produces an event also at time t?
- According to the semantics of Alarm, it should not raise an alarm event.
- Does the DE simulation algorithm guarantee this?

Back to the Alarm Example

$t := 0$;
initialize global event queue Q with $\{(alarm, t), (cancel, t)\}$;
while Q is not empty **do**
    remove earliest event $e = (v_e, t_e)$ from Q;
    $t := t_e$;
    execute event e: update system state, generate possible future events, and add them to Q, ordered by timestamps; possibly remove events from Q;
end while

- Non-determinism!
  - Different results depending on which of the two instantaneous events $(alarm, t)$ and $(cancel, t)$ is first removed from Q.

Dealing with Simultaneous Events

- Chronological ordering (= ordering by timestamps) of events in the queue is not enough
- Must also respect dependencies between simultaneous events
  - Alarm's output event at time t depends on Source's output event at time t
- How to define event dependencies?

Dealing with Simultaneous Events

- Chronological ordering (= ordering by timestamps) of events in the queue is not enough
- Must also respect dependencies between simultaneous events
  - Alarm's output event at time t depends on Source's output event at time t
- How to define event dependencies?
- First let's formalize actor dependencies.

Dependency Relation among Actors

Let A_1, A_2 be two actors.

Define the precedence relation $A_1 \rightarrow A_2$ (A_2 depends on A_1) as follows:

$A_1 \rightarrow A_2$ iff A_1 is zero-delay and there is a connection from an output of A_1 to an input of A_2 in the model.

Dependency Relation among Actors

Let A_1, A_2 be two actors.

Define the precedence relation $A_1 \rightarrow A_2$ (A_2 depends on A_1) as follows:

$A_1 \rightarrow A_2$ iff A_1 is zero-delay and there is a connection from an output of A_1 to an input of A_2 in the model.

Claim: $\rightarrow$ is acyclic.

Why?

Dependency Relation among Actors

Let A_1, A_2 be two actors.

Define the precedence relation $A_1 \to A_2$ (A_2 depends on A_1) as follows:

$A_1 \to A_2$ iff A_1 is zero-delay and there is a connection from an output of A_1 to an input of A_2 in the model.

Claim: $\to$ is acyclic.

Why? Because every loop is assumed to have a non-zero-delay actor.

Actor Dependencies – Examples

Alarm $\rightarrow$ Sink.

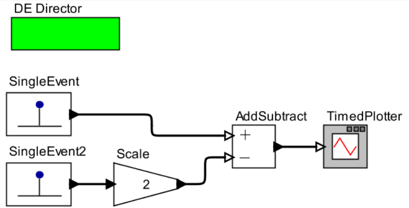
We will not need Source $\rightarrow$ Alarm.

Actor Dependencies – Examples

$$\boxed{\text{Source}} \xrightarrow{\;cancel\;} \boxed{\text{Alarm}} \xrightarrow{\;alarm\;} \boxed{\text{Sink}}$$

Alarm $\rightarrow$ Sink.

We will not need Source $\rightarrow$ Alarm.



Scale $\rightarrow$ AddSubtract $\rightarrow$ TimedPlotter.

Precedence Relation on Events

Let $e_1 = (v_1, t_1)$ and $e_2 = (v_2, t_2)$ be two events.

Let A_1 and A_2 be the recipient actors of e_1 and e_2:

- This information can be encoded in v_1, v_2.
- We assume a unique recipient per event.
  - No loss of generality: can view fan-out junctions as zero-delay actors which copy every input event to all their outputs.

Precedence Relation on Events

Let $e_1 = (v_1, t_1)$ and $e_2 = (v_2, t_2)$ be two events.

Let A_1 and A_2 be the recipient actors of e_1 and e_2:

- This information can be encoded in v_1, v_2.
- We assume a unique recipient per event.
  - No loss of generality: can view fan-out junctions as zero-delay actors which copy every input event to all their outputs.

Then:

$$e_1 \prec e_2 \text{ iff } t_1 < t_2 \text{ or } t_1 = t_2 \text{ and } A_1 \to^* A_2 \text{ and } A_1 \neq A_2$$

where $\to^*$ is the transitive closure of $\to$.

Precedence Relation on Events

Let $e_1 = (v_1, t_1)$ and $e_2 = (v_2, t_2)$ be two events.

Let A_1 and A_2 be the recipient actors of e_1 and e_2:

- This information can be encoded in v_1, v_2.
- We assume a unique recipient per event.
  - No loss of generality: can view fan-out junctions as zero-delay actors which copy every input event to all their outputs.

Then:

$$e_1 \prec e_2 \text{ iff } t_1 < t_2 \text{ or } t_1 = t_2 \text{ and } A_1 \rightarrow^* A_2 \text{ and } A_1 \neq A_2$$

where $\rightarrow^*$ is the transitive closure of $\rightarrow$.

Claim: $\prec$ is acyclic.

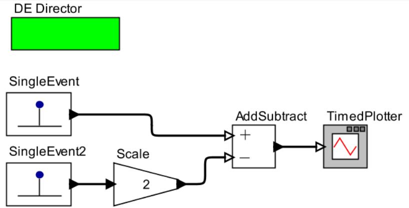
Event Dependencies – Examples

$cancel \prec alarm.$

Event Dependencies – Examples

$$\boxed{\text{Source}} \xrightarrow{cancel} \boxed{\text{Alarm}} \xrightarrow{alarm} \boxed{\text{Sink}}$$

$cancel \prec alarm$.



Suppose there are 3 events, e_1, e_2, e_3, pending at the input port of Scale and the two input ports of AddSubtract, respectively. Then:

$$e_1 \prec e_2 \text{ and } e_1 \prec e_3.$$

e_2 and e_3 are independent.

Discrete-Event Simulation – final version

$t := 0$;

initialize global event queue Q with a set of initial events;

// Q is always implicitly ordered w.r.t. timestamps
// and among events with same timestamp
// w.r.t. event dependencies

while Q is not empty **do**

  remove set E of all minimal events w.r.t. $\prec$ from Q;

  // these are earliest and simultaneous events,
  // which depend on no other events

  $t := t_e$;

  execute set of events E: update system state, generate possible future events, and add them to Q, ordered by timestamps; possibly remove events from Q;

end while

Discrete-Event Simulation – final version

1: $t := 0$;
2: initialize global event queue Q with a set of initial events;
    // Q is always implicitly ordered w.r.t. timestamps
    // and among events with same timestamp
    // w.r.t. event dependencies
3: **while** Q is not empty **do**
4:     remove set E of all minimal events w.r.t. $\prec$ from Q;
        // these are earliest and simultaneous events,
        // which depend on no other events
5:     $t := t_e$;
6:     execute set of events E: update system state, generate possible future events, and add them to Q, ordered by timestamps; possibly remove events from Q;
7: **end while**

Claim: any new event e produced in step 5 is guaranteed to be greater than all events in set E w.r.t. $\prec$. That is, either e has greater timestamp than all events in E, or it depends on some event in E.

DE Simulation and HDLs

- HDLs: Hardware Description Languages
- Verilog, VHDL, SystemC, ...
- Real-world languages
- EDA (Electronics Design Automation) industry: billions of $$$
- Simulation tools: based on DE simulation
- But note: many variants, details, ...
  - E.g., SystemC specification[1] is > 600 pages long.
  - Description of the simulation algorithm (in English) is 16 pages long.

[1] IEEE Standard 1666 - 2011, freely available online

SystemC

- Set of C++ libraries.
- Simulation algorithm: variant of DE simulation
  - kernel (scheduler) manipulates queue of *runnable processes*

SystemC

- Set of C++ libraries.
- Simulation algorithm: variant of DE simulation
  - kernel (scheduler) manipulates queue of *runnable processes*
- Main phases of the algorithm:
  - Elaboration phase (= initialization):
    - instantiate processes and signals for inter-process communication (processes & signals are C++ objects)
    - connect processes to signals (port binding)
    - select runnable processes
    - start simulation
  - Simulation phase:
    - choose a process P among the set of runnable processes
    - run/resume P until P completes or calls `wait` (or something similar)
    - repeat until no more runnable processes
    - advance time (may make new processes runnable) and repeat

SystemC

Remarks:

- *Co-operative multitasking*: processes must release control back to the kernel/scheduler
  - Process executes forever $\Rightarrow$ zeno system!
- Processes may generate instantaneous events and the same process may become runnable multiple times without time advancing – *immediate* and *delta* steps
- *"The order in which process instances are selected from the set of runnable processes is implementation-defined."*

SystemC

Remarks:

- *Co-operative multitasking*: processes must release control back to the kernel/scheduler
  - Process executes forever $\Rightarrow$ zeno system!
- Processes may generate instantaneous events and the same process may become runnable multiple times without time advancing – *immediate* and *delta* steps
- *"The order in which process instances are selected from the set of runnable processes is implementation-defined."*
- Execution apparently not ordered w.r.t. dependencies.
  $\Rightarrow$ non-determinism!

Bibliography

J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol.
Discrete-event system simulation – 5th ed.
Prentice Hall, 2005.

M. Broy and K. Stølen.
Specification and development of interactive systems: focus on streams, interfaces, and refinement.
Springer, 2001.

E. Lee and A. Sangiovanni-Vincentelli.
A unified framework for comparing models of computation.
IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, 17(12):1217–1229, December 1998.

E. A. Lee.
Modeling concurrent real-time processes using discrete events.
Annals of Software Engineering, 7:25–45, 1999.

J. Misra.
Distributed discrete-event simulation.
ACM Comput. Surv., 18(1):39–65, March 1986.