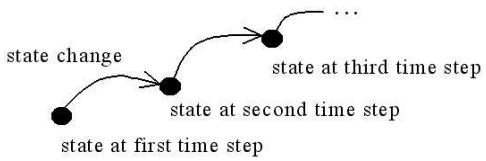
Slide 0

Chap.3 Modeling Formalisms and Their Simulators

Discrete time models and their simulators

- stepwise mode of execution



Slide 1

  - at a particular time instant, the model is in a particular state and it defines how this state changes
  - next state usually depends on
    - the current state and also
    - what the environment's influences currently are
  - applications
    - the most popular application is digital systems where the clock defines the discrete time steps
    - the frequent application is the approximations of continuous systems

Discrete time models and their simulators (Cont'd)

Slide 2

- discrete time model
  - define how the current state and the input from the environment determine the next state of the model

Current State	Current Input	Next State	Current Output
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

Discrete time models and their simulators (Cont'd)

Slide 3

  - state change over time
    - informal description
      - if the state at time t is q and the input time t is x, then
      - the state at time $t+1$ will be $\delta(q,x)$ and the output y at time t will be $\lambda(q,x)$ where δ is the *state transition function* and λ is *output function*
    - compact formal form (states and inputs must be finite)
      $\delta(q,x) = x$ and
      $\lambda(q,x) = x$
    - *state trajectory*
      - a sequence of states, $q(0), q(1), q(2), \ldots$
      - given a initial state $q(0)$, then the subsequent state are determined by $q(t+1) = \delta(q(t), x(t))$
    - *output trajectory*
      - a sequence of outputs which is determined by $y(t) = \lambda(q(t), x(t))$

Discrete time models and their simulators (Cont'd)

Slide 4

* state and output trajectories

time	0	1	2	3	4	5	6	7	8	9	
input trajectory	1	0	1	0	1	0	1	0	1	0	
state trajectory	0	1	0	1	0	1	0	1	0	1	0
output trajectory	1	0	1	0	1	0	1	0	1	0	

Discrete time simulation

Slide 5

- a little algorithm to compute the state and output trajectories

$Ti = 0,\ Tf = 9$ `the starting and ending times, here 0 and 9`
$x(0) = 1,\ \ldots,\ x(9) = 0$ `the input trajectory`
$q(0) = 0$ `the initial state`

$t = Ti$
`while (`$t <= Tf$`) {`
  $y(t) = \lambda(q(t),\ x(t))$
  $q(t+1) = \delta(q(t),\ x(t))$
}

Cellular Automata

- one-dimensional cell space

Slide 6

  - each has two states (0 and 1)
  - each gets the states of its neighbors as inputs
  - eight combinations of states and inputs $\rightarrow 2^8 = 256$ functions
  - suppose
    - choose one function and one of its states
    - apply appropriate simulation algorithm
    - what would we observe? $\rightarrow$cellular automata

Cellular Automata (Cont'd)

  - *cellular automata*
    - idealization of a physical phenomenon in which
      - space and time are discretized
      - the state sets are discrete and finite
    - its components are called *cells*
      - geometrically located on a one-,two-, or multidimensional grid and
      - connected in a uniform way
      - the *neighborhood* are often chosen nearest in the geometrical sense

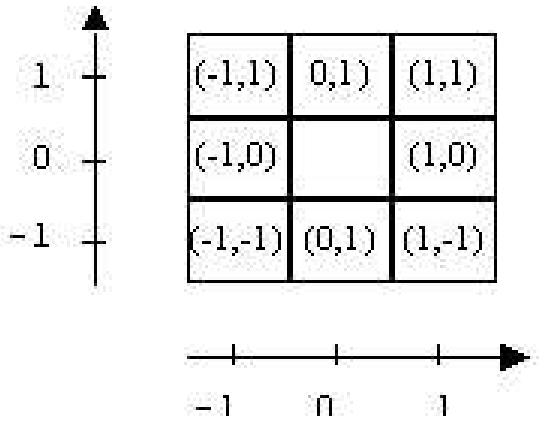
Slide 7

- interesting property
  - four types
    - automata where any dynamic soon dies out
    - automata that soon come to periodic behavior
    - automata that show chaotic behavior
    - the most interesting ones, automata whose behaviors are unpredictable and nonperiodic but that show interesting, regular patterns

Cellular Automata (Cont'd)

- Conway's Game of Life
  - 2D cell space
  - (i, j)'s eight neighbors: $(i, j+1), (i+1, j), (i, j-1), (i-1, j), (i+1, j+1), (i+1, j-1), (i-1, j+1), (i-1, j-1)$

Slide 8



Cellular Automata (Cont'd)

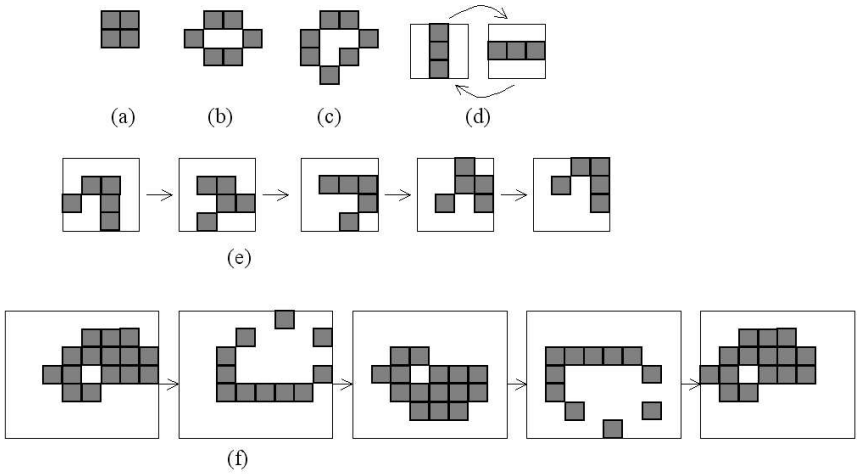
Slide 9

- each cell state
  - 0 (*dead*)
  - 1 (*alive*)
- state transition rule (function)
  - 1 →1 : survive
    if it has between 2 and 3 live cells in its neighborhood
  - 0 →1 : born
    if it has exactly 3 alive neighbors
  - 1 →0 : die
    if it has more than 3 live cells in its neighborhood (overcrowding)
    if it has fewer than 2 live neighbors (isolation)

Cellular Automata (Cont'd)

Slide 10

- interesting patterns



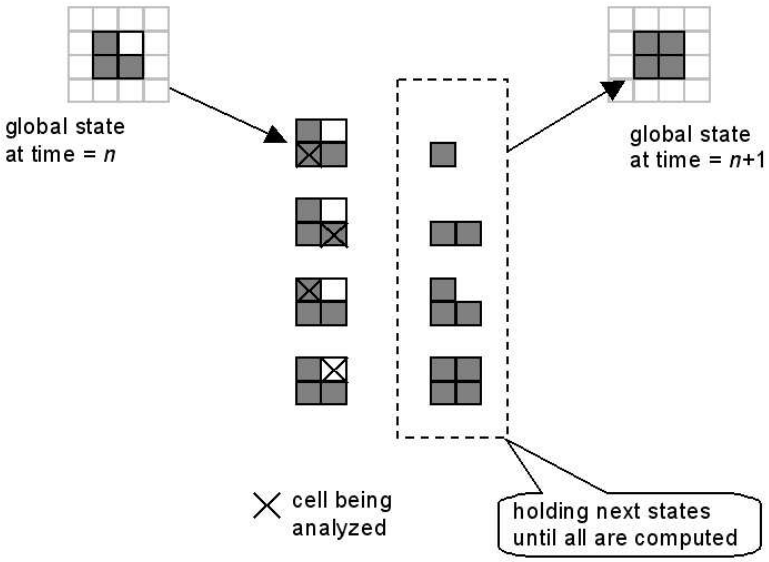
Cellular automaton simulation algorithms

Slide 11

- basic procedure for simulating a cellular automaton
  - at every time step
  - scan all cells
  - apply the state transition function to each
  - save the next state in a second copy of the global state data structure
- for example
  - let's start with the three live cells

Cellular automaton simulation algorithms (Cont'd)

Slide 12



Cellular automaton simulation algorithms (Cont'd)

Slide 13

- cell space
  - the cell space is infinite but, we can't scan all the cells

  → examined part of the space
  - boundary
    - assume fixed values for all boundary cells (for example, all dead)
    - to wrap the space in toroidal fashion

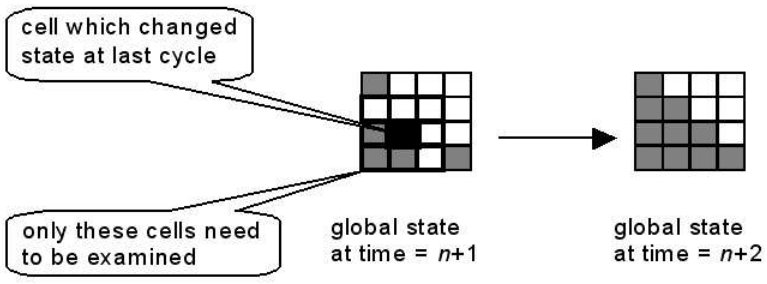
Discrete event approach to cellular automaton simulation

Slide 14

- discrete time systems
  - at every time step each component undergones a "state transition"
  - this occurs whether or not its state actually changes
  - only a small number of components really change
- discrete event simulation
  - if *events* are defined as changes in state
  - a discrete event simulation algorithm **concentrates on processing events** rather than cells
  - criterion under which cells can change
    - a cell will not change state at the next state transition time,
    - if none of its neighboring changed state at the current state transition time

Discrete event approach to cellular automaton simulation (Cont'd)

Slide 15

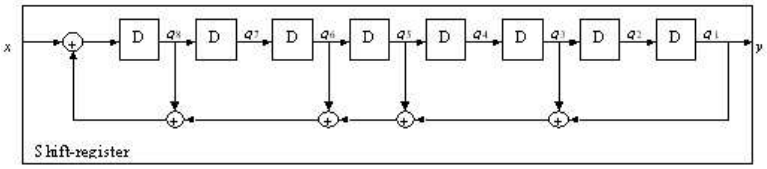


  - note that
    - we can limit the scanning
    - but we cannot predict which cells will actually have events without actually applying the transition function

Switching automata/sequential machines

Slide 16

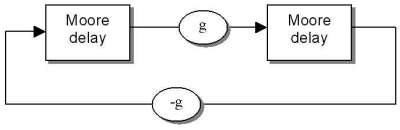
- switching automata
  - no requirements of uniform composition and interconnection patterns of cellular automata
  - also called digital circuits
  - constructed from flip-flop components and logical gates
  - memoryless systems
    - no flip-flop (no internal states)
  - an example

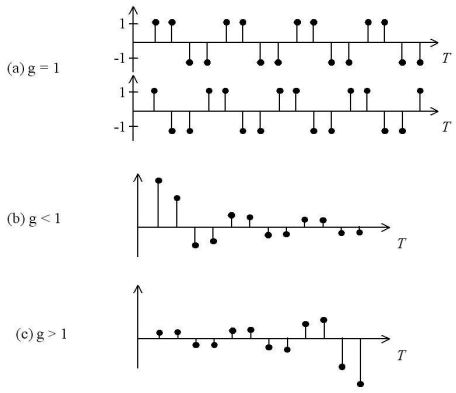


Linear discrete time networks and their state behavior

Slide 17

- a Moore network with two delay elements and two memoryless elements





Linear discrete time networks and their state behavior (Cont'd)

- matrix representation

$$\begin{bmatrix} 0 & g \\ -g & 0 \end{bmatrix}$$

- next state (started in a state $[1, 1]$)

Slide 18

$$\begin{bmatrix} g \\ -g \end{bmatrix} = \begin{bmatrix} 0 & g \\ -g & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- represented by

  $\delta(q, x) = Aq + Bx$

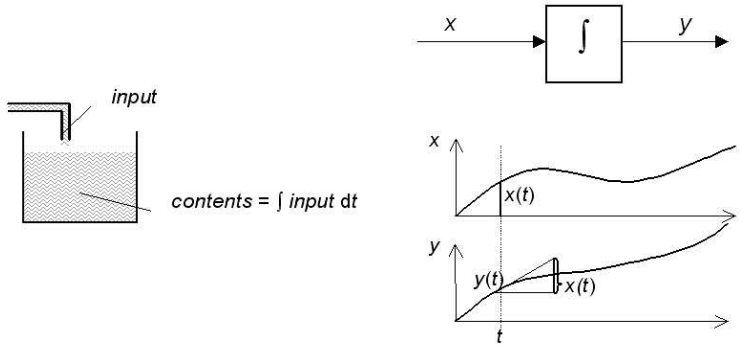
  $\lambda(q) = Cq$

- the delays in the discrete domain correspond to integrators in the continuous domain

Differential equation models and their simulators

- for differential models
  - do not specify a next state directly
  - but use a *derivative function* to specify the rate of changes of states
  - at any time
    - ∗ given a state and an input, we only know the rate of change of the state
- the most elementary continuous system (the simple integrator)

Slide 19



Differential equation models and their simulators (Cont'd)

- represented by

  $dq(t)/dt = x(t)$

  $y(t) = q(t)$

Slide 20

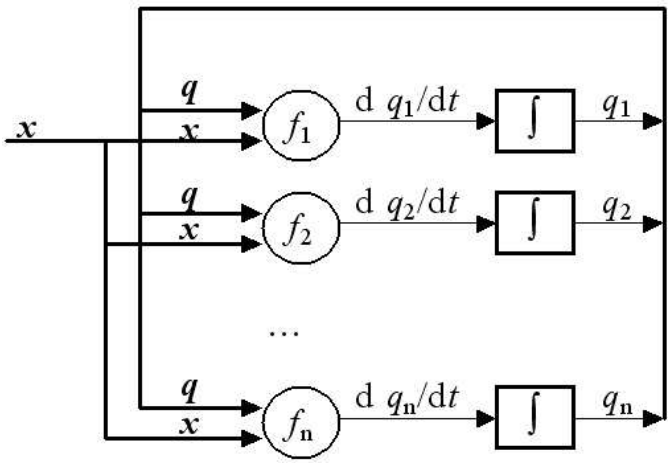
  - usually continuous systems are expressed

$$dq_1(t)/dt = f_1(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t))$$
$$dq_2(t)/dt = f_2(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t))$$
$$\ldots$$
$$dq_n(t)/dt = f_n(q_1(t), q_2(t), \ldots, q_n(t), x_1(t), x_2(t), \ldots, x_m(t))$$

Differential equation models and their simulators (Cont'd)

  - state variables q_i are computed

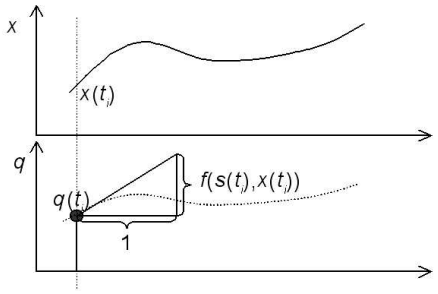
Slide 21



Continuous system simulation

- fundamental problem occurs
  - when a continuous system is simulated on a digital computer
  - given a state vector q and an input vector x for a particular time t_i we know only dq_i/dt
  - how do we obtain the state values after this time?
  - computer program must estimate the values at t_{i+1} without knowledge of states between t_i and t_{i+1}

Slide 22



Continuous system simulation (Cont'd)

- numerical integration method
  - *Euler* or *rectangular* method

$$\frac{dq}{dt} = \lim_{h \to 0} \frac{q(t+h) - q(t)}{h}$$

  - for small enough h

$$q(t+h) = q(t) + h \cdot \frac{dq(t)}{dt}$$

Slide 23

  - problems
    - h must be sufficiently small
    - the smaller the step size, the more integrations are needed
- basic idea of numerical integration
  - employ estimated past and/or future vales of states, inputs, derivatives

$$\begin{aligned} q(t_i) \quad = \quad & integration\_method(\ldots, q(t_{i-2}), f(q(t_{i-2}), x(t_{i-2})), \\ & q(t_{i-1}), f(q(t_{i-1}), x(t_{i-1}), q^{'}(t_i), f(q^{'}(t_i), x^{'}(t_i)), \\ & q^{'}(t_{i+1}), f(q^{'}(t_{i+1}), x^{'}(t_{i+1})), \ldots) \end{aligned}$$

Continuous system simulation (Cont'd)

Slide 24

- *causal* or non-causal methods
  - causal: only employ values at prior computation instants
  - noncausal: in addition, it also employ estimated values
- causal methods
  - for example, Adams method

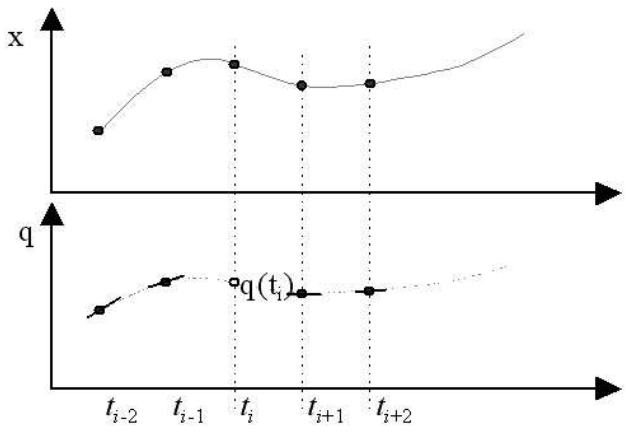
$$q(t_i) = q(t_{i-1}) + h(3f(q(t_{i-1}), x(t_{i-1})) - f(q(t_{i-2}), x(t_{i-2})))$$

  - *startup problem*
    - how to determine state, input, derivative values for the times t_{1-m} to t_0
    - use lower orders in the startup phase (one solution)
    - use noncausal in the startup phase (more reliable)

Continuous system simulation (Cont'd)

Slide 25

- noncausal methods



  - use the future values of states, derivatives and inputs
  - the values at the present time may be computed twice (predictor phase and corrector phase)

Continuous system simulation (Cont'd)

* *Heun* method

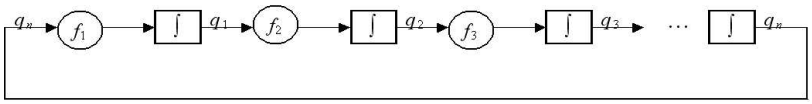
$$q^{'}(t_i) = q(t_{i-1}) + hf(q(t_{i-1}, x(t_{i-1}))$$
$$q(t_i) = q(t_{i-1}) + h/2(f(q^{'}(t_i), x^{'}(t_i)) + f(q(t_{i-1}, x(t_{i-1})))$$

Slide 26

* *variable stepsize* methods
  - the difference between two values is too large →the step size is decreased
  - predictor-corrector cycle repeated until a step size is found for which the difference is below the criterion
* the most popular noncausal methods →*Runge-Kutta* methods
  - do not use past values but only use the current state and input value
  - can be employed in the startup phase of causal methods

Feedback in continuous systems

- feedback
  - state variables are fed back to influence their own rates of change



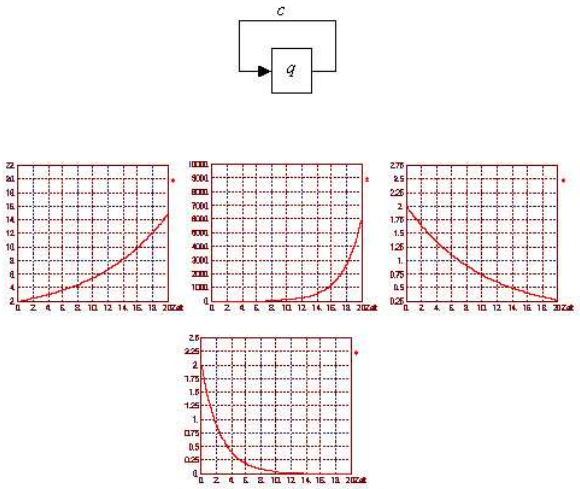
Slide 27

  - positive and negative feedback
    * positive feedback
      - positive influence
      - a even number of negative influences
      - cause the state to grow indefinitely (unstable and undesirable)
    * negative feedback
      - negative influence
      - a odd number of negative influences
      - tend to stabilize a system

Elementary linear systems

- simple continuous systems
  - show quite interesting behavior caused by feedback

Slide 28



Elementary linear systems (Cont'd)

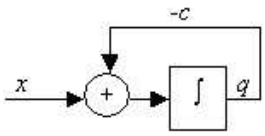
Slide 29

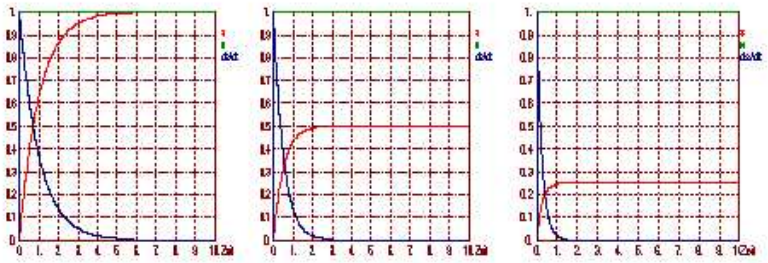
- if c is positive
  - positive feedback
  - grow exponentially (exponential growth)
- if c is negative
  - negative feedback
  - shrink approaching zero exponentially (exponential decay)
- if c is zero
  - remain constant

Elementary linear systems (Cont'd)

- first order systems

Slic





Elementary linear systems (Cont'd)

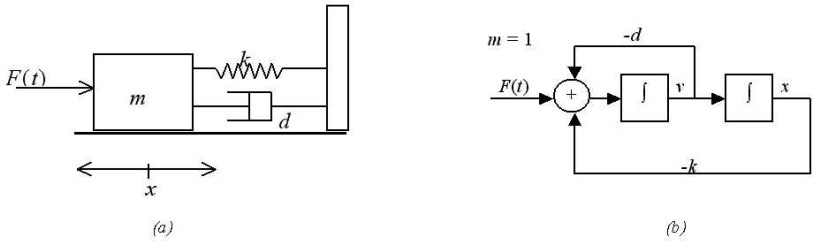
Slide 31

- apply an input to an exponential decay (exponential decay of first order)
  - play an important role in many real-world phenomena
  - input represents the driving force
  - the feedback through negative factor c models the damping
  - characteristic of the system
    - it reaches an equilibrium state when the damping equals the driving input
- real-world phenomena
  - heating systems
    - input is the heat supply
    - the damping represents the heat losses to the environment
  - mechanical systems
    - input is the force supplied
    - the damping represents the friction

Elementary linear systems (Cont'd)

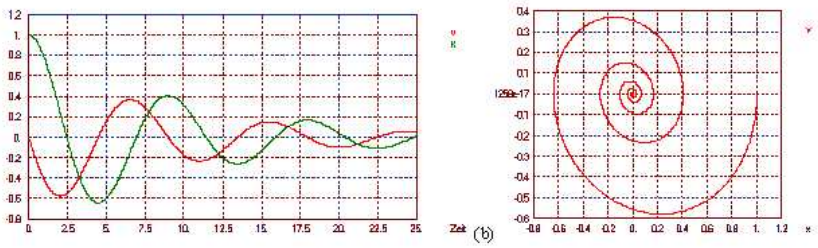
Slide 32

- second order systems
  - simple systems are one state variable with a direct feedback loop
  - consider two variables, *linear (or harmonic) oscillator of second order*
  - mechanical oscillator



Elementary linear systems (Cont'd)

Slide 33

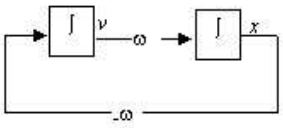


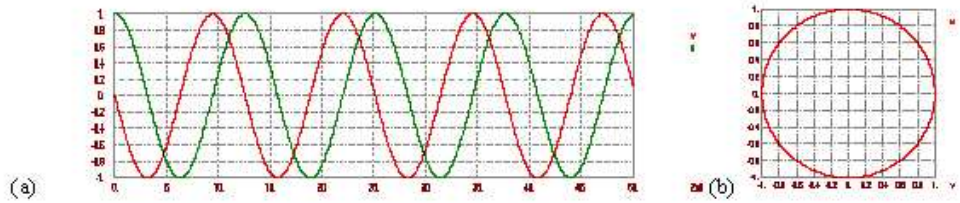
* $F(t)$ is applied to the mass over some time period
* the system is modeled by the current position x and the current velocity v ($v = dx/dt$)
* the negative feedback from x to v occurs through the force of spring
  - cause the system to oscillate

Elementary linear systems (Cont'd)

Slide

* if we remove the damping →a pure oscillator





- neutrally stable since it is neither damped nor undamped

Elementary linear systems (Cont'd)

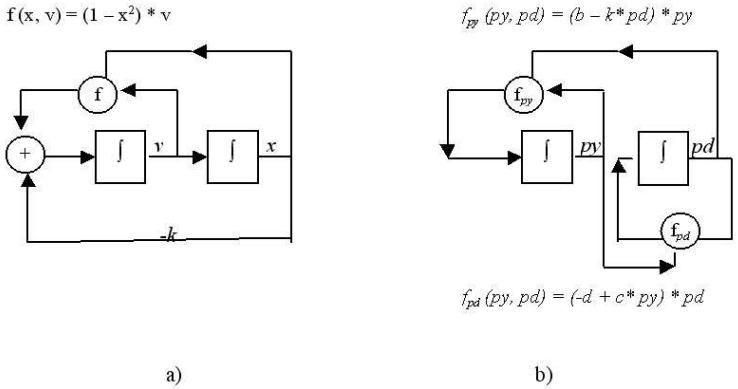
Slide 35

– discrete and continuous oscillation
  * correspondence
    - have a negative feedback, cause oscillation
  * difference
    - inputs
      discrete: inputs in the delays define new state values
      continuous: inputs represent rate of changes values
    - gains
      discrete: determines the amplification or decay of the state values
      continuous: defines how fast the change of the state (frequency)
  * frequency in discrete oscillator is determined by the amount of delay

Nonlinear oscillators: limit cycles and chaotic behaviors

- nonlinear system
  - so-called Van der Pol system
  - a nonlinear feedback is added to the linear oscillator
    - instead of the linear damping factor $-d$
    - a nonlinear feedback $(1 - x^2) * v$ is added

Slide 36

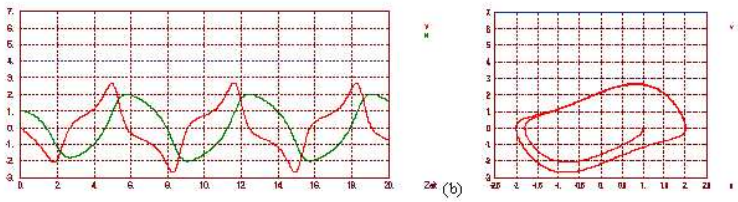


Nonlinear oscillators: limit cycles and chaotic behaviors (Cont'd)

- astonishing properties
  - independent of its initial state values
  - the system quickly comes into a stable oscillation
    - for small values of $x \rightarrow (1 - x^2)$ is positive $\rightarrow$amplification of v
    - for large values of $x \rightarrow (1 - x^2)$ is negative $\rightarrow$fast damping of v

Slide 37

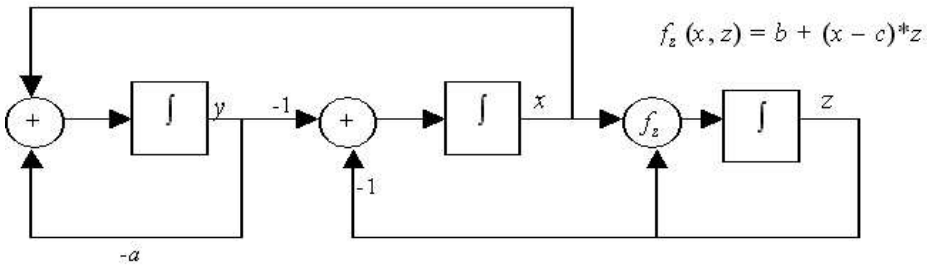
☛ stable oscillation



Nonlinear oscillators: limit cycles and chaotic behaviors (Cont'd)

- chaotic system
  - nonlinear systems can show remarkable stability properties
  - some nonlinear systems exhibit highly unstable behaviors that appear chaotic
  - an example (Rössler system)

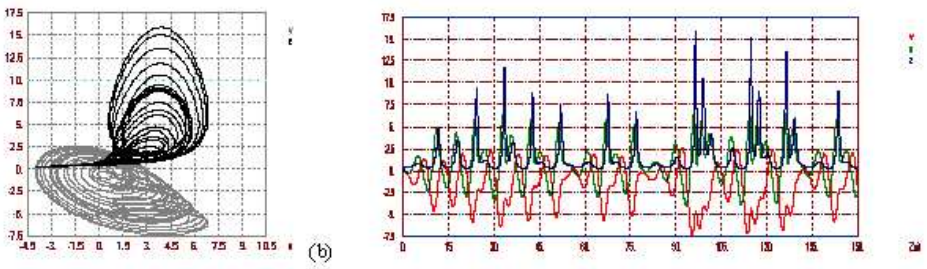
Slide 38



Nonlinear oscillators: limit cycles and chaotic behaviors (Cont'd)

- characteristic
  - it shows neither to rest nor to some periodic behavior (chaotic behavior)
  - its behavior appears to be unpredictable but a pattern may be discerned

Slide 39



Continuous system simulation languages and systems

Slide 40

- two major classes
  - Continuous System Simulation Language (CSSL)
  - block-oriented simulation systems
- CSSL
  - ACSL, an CSSL language
    - $x = integ(v, x_0)$
    - Fortran syntax

Continuous system simulation languages and systems (Cont'd)

Slide 41

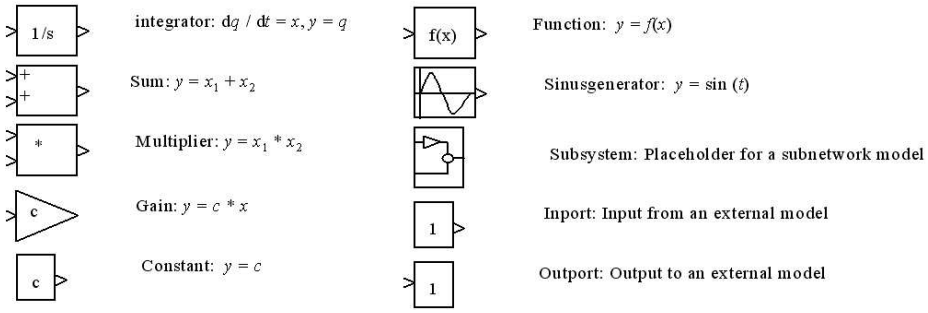
- an example, (Van der Pol equation)

```
PROGRAM Van der Pol
  INITIAL
    constant
      k = -1, x0 = 1, v0 = 0,
      tf = 20
  END
  DYNAMIC
    DERIVATIVE
      x = integ(v, x0)
      v = integ((1-x**2)*v - k*x, v0)
    END
    termt (t.ge.tf)
  END
END
```

Continuous system simulation languages and systems (Cont'd)

- block-oriented simulation systems
  - similar to the programming of analog computers
  - primarily used by control engineers
  - modeling is done by coupling together primitive components and elementary building blocks
  - elementary building blocks from *simulink*

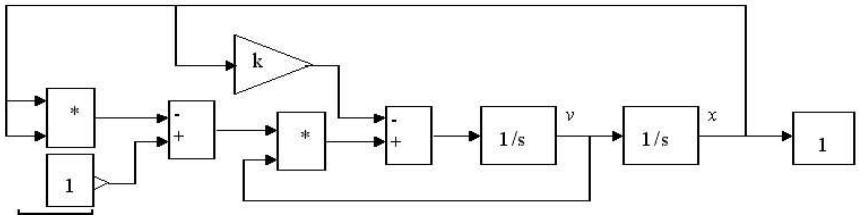
Slide 42



Continuous system simulation languages and systems (Cont'd)

- simulink
  - a superset of Matlab package
  - provide many other blocks beyond pure continuous modeling (difference equations and discrete control systems)
  - the most basic block is the *integrator*
    - receive one real value and output the integral of input starting from an initial state
    - realize a memory element with a single state variable
  - allow hierarchical modeling by providing the *subsystem*
  - model specification of Van der Pol equations

Slide 43



Discrete event models and their simulators

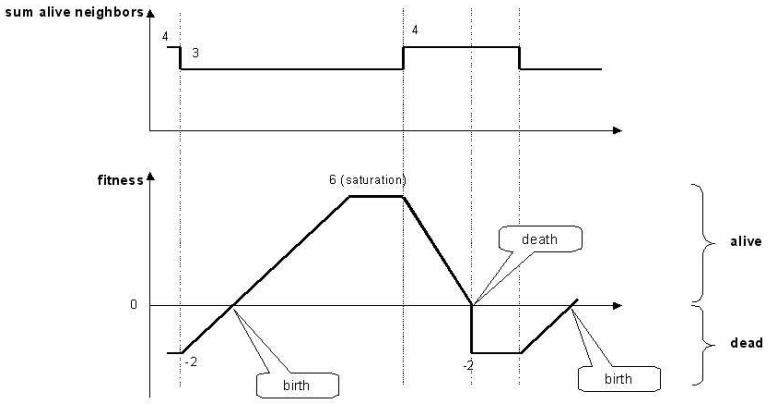
- introduction
  - discrete event modeling is an attractive formalism
  - play more and more of a role in all kinds of modeling in the future

Slide 44

- discrete event cellular automata
  - the birth and death are dependent on its fitness
    - a cell attains positive fitness when it has exactly 3 neighbors (when the fitness reaches 0, the cell will birth)
    - the fitness of a cell will diminish rapidly when its environment is hostile (when the fitness reaches 0, the cell will die)
    - a dead cell will have a negative initial fitness (for example, -2)

Discrete event models and their simulators (Cont'd)

Slide 45



- computation of trajectories of fitness is continuous simulation

→ need combined discrete event and continuous simulation (Chapter 9)

→ computationally inefficient because continuous trajectory is computed at every point

- a much more efficient way is to adopt a pure event-based approach

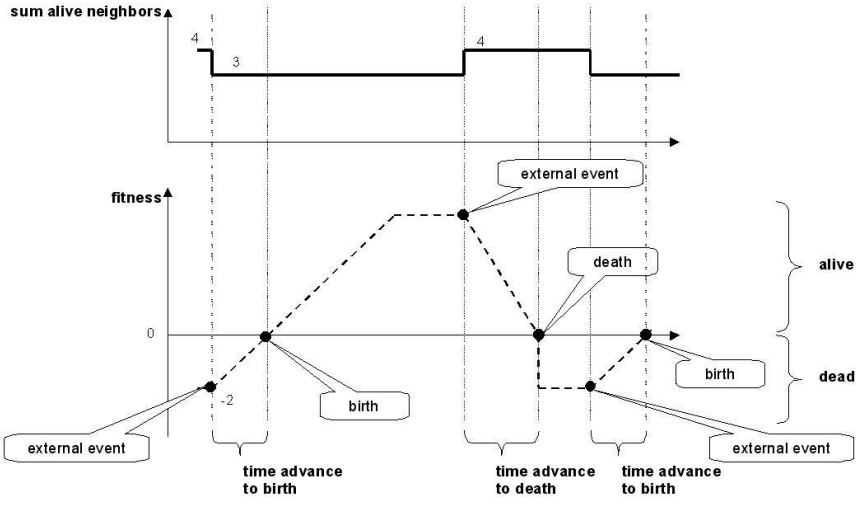
Discrete event models and their simulators (Cont'd)

Slide 46

- a pure event-based approach
  - * prerequisite of discrete event modeling is to determine when interesting things happen
  - * events can be caused by the environment, such as the changes of the sum of alive neighbors
  - * the external events are not under the control of the model component
  - * but, it may schedule events (*internal events*) to occur
  - * the component itself determines the time according to its state by time advance
  - * the component receives the internal event on the time if no external events occur

Discrete event models and their simulators (Cont'd)

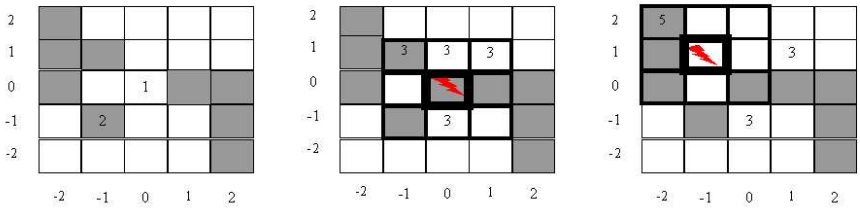
Slide 47



Discrete event models and their simulators (Cont'd)

Slide 48

* a change in state may affect neighbors' waiting times as well as scheduling of new events and cancellation of events



* (a)
  - (0,0) a birth at time 1 and (-1,-1) a death at time 2
* (b)
  - (0,0) alive
    →(0,1), (0,-1), (1,1) a birth at time 3
    →(-1,1) a death at time 3
  - (-1,-1) a death at time 2 must be canceled

Discrete event models and their simulators (Cont'd)

Slide 49

* no events at time 2
  - illustrates an important efficiency advantage in discrete event simulation
  - during times when no events are scheduled, no components need be scanned
  - in discrete time simulation, scanning must be performed at each time step
* (c)
  - a problem of simultaneous events (three birth and one death)
  - who goes first?
  - if (-1,1) goes first, (0,1) have 2 neighbors →a birth event must be canceled
  - if (0,1) goes first, (0,1) will be alive

☛ results will be different for different ordering of activation

  - solutions
    1) all simultaneous events undergo their state transitions together (parallel DEVS approach)
    2) define priority among components (employed by most packages and classic DEVS)

Discrete event world views

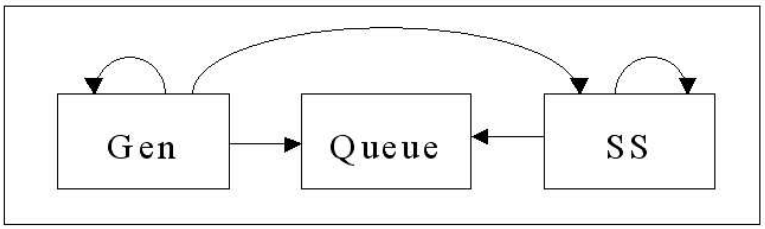
Slide 50

- three common types of simulation strategies
  - event scheduling
  - activity scanning
  - process interaction: event scheduling + activity scanning
  - nowadays, most simulation environments employ a combination of these strategies
- event scheduling world view
  - event oriented models *preschedule* all events
  - scheduling an event is straightforward when all the conditions necessary for its occurrence can be known in advance
  - this is not always the case

Discrete event world views (Cont'd)

Slide 51

- an example



  * Gen generates jobs with a time advance, *inter-gen-time*
  * SS server processes the jobs with a time advance, *service-time*
  * Queue is a buffer for queuing the jobs (passive elements, time advance to infinity)

Discrete event world views (Cont'd)

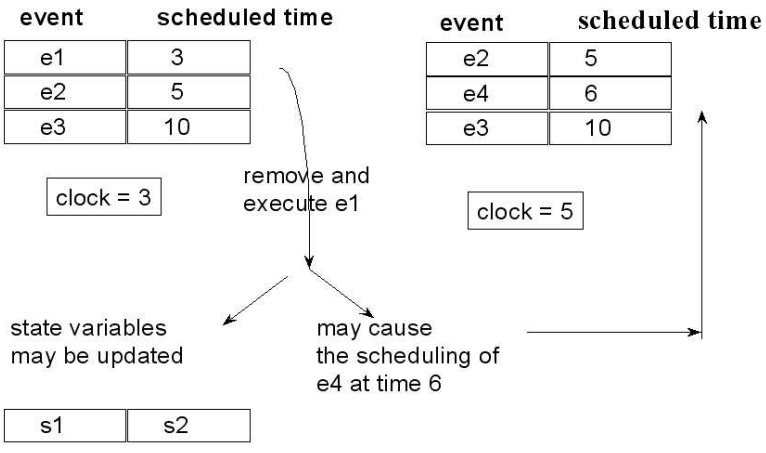
Slide 52

* two event routines and a tie-breaking function

```
Event: Generate_Job
  nr-waiting = nr-waiting + 1
  schedule a Generate_Job in inter-gen-time
  if nr-waiting = 1 then
    schedule a Process_Job in service-time

Event: Process_Job
  nr-waiting = nr-waiting - 1
  if nr-waiting > 0 then
    schedule a Process_Job in service-time

Note that the phrase "schedule an event in T" means the same
as "schedule an event at time = clock time + T".

Break-Ties by: Process_Job then Generate_Job
```

Discrete event world views (Cont'd)

Slide 53



Discrete event world views (Cont'd)

Slide 54

- the earliest scheduled time (e_1) is removed from the list
- the clock is advanced to the time of this event (3)
- the routine associated with this event is executed
- if there is more than one imminent event, then tie-breaking procedure is employed
- execution of the event may cause new events to be added in the proper place on the list (for example, e_4 is scheduled at time 6)
- existing events may be rescheduled or even canceled
- the next cycle begins with the earliest schedule time (5)

Discrete event world views (Cont'd)

Slide 55

* event list scheduling (at simple workflow example)
  - initially, the clock is set to 0 and *Generate_Job* event is place on the list with time 0
  - the state of the system is represented by *nr-waiting*, which is initially 0
  - the *Generate_Job* event is executed →causes two events
    - a *Generate_Job* at time *inter-gen-time*
    - a *Process_Job* at time *service-time* since *nr-waiting* becomes 1
    - if *inter-gen-time* is smaller than *service-time*
      →*Generate_Job* is executed
      →only new *Generate_Job* is scheduled since the *nr-waiting* is 2 the *Process_Job* is not scheduled
    - if *service-time* $<$ *2 * inter-gen-time*
      →the next event is a *Process_Job*
  -