Discrete Time and Discrete Event Modeling Formalisms and Their Simulators

Dr. Xiaolin Hu
Outline

• Review of last class
• Discrete time modeling and simulation
• Cellular Automata, Game of Life
• Discrete event modeling and simulation
Ways To Study A System

System

Experiment with actual system

Experiment with a model of actual system

Physical model

Mathematical model

Analytical Solution

Simulation

*Simulation, Modeling & Analysis (3/e) by Law and Kelton, 2000, p. 4, Figure 1.1*
Model Taxonomy

- **System Model**
  - Deterministic
    - Static
    - Dynamic
      - Continuous
      - Discrete
  - Stochastic
    - Static
    - Dynamic
      - Continuous
      - Discrete
        - Monte Carlo Simulation
        - Discrete-event Simulation

Dr. Xiaolin Hu
Modeling formalisms and their simulators
(chapter 3 of Zeigler’s book)

• Discrete Time model and their simulators
• Differential Equation models and their simulators
• Discrete Event models and their simulators
Discrete Time Models and Simulators

- Discrete time models are usually the most intuitive to grasp of all forms of dynamic models.
- This formalism assumes a stepwise mode of execution.
- At a particular time the model is in a particular state and it defines how its state changes. The next state usually depends on the current state and also what the environment’s influences currently are.
Discrete Time Model

- Discrete time systems have numerous applications.
- The most popular are in digital systems where the clock defines the discrete time steps.
- It is also frequently used as approximations of continuous systems.
- To build a discrete time model, we have to define how the current state and the input from the environment determine the next state of the model.
- This can be done using a table

<table>
<thead>
<tr>
<th>Current state</th>
<th>Current input</th>
<th>Next state</th>
<th>Current output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The transition/output table for a delay system
Discrete Time Model

• In discrete time models, time advances in discrete steps, which we assume are integer multiples of some basic period such as 1s, 1 day.
• The transition/output table can also be described as follows:
  – If the state at time $t$ is $q$ and the input at 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 $\lambda(q,x)$.
• Here $\delta$ is called the state transition function and is the more abstract concept for the first three columns of the table. $\lambda$ is called the output function.

<table>
<thead>
<tr>
<th>Current state</th>
<th>Current input</th>
<th>Next state</th>
<th>Current output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Dr. Xiaolin Hu
Discrete Time Model

<table>
<thead>
<tr>
<th>Current state</th>
<th>Current input</th>
<th>Next state</th>
<th>Current output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Question: specify $\delta(q,x)$ and $\lambda(q,x)$ for the table above

\[
\delta(q,x) = x \\
\lambda(q,x) = x
\]

- The function $\delta(q,x)$ and $\lambda(q,x)$ are more general than the table.
- Now add time in the above specification.
Discrete Time Model

- A sequence of state, $q(0), q(1), q(2), \ldots$ is called a state trajectory. Having an arbitrary initial state $q(0)$, subsequent states in the sequence are determined by
  \[ q(t+1) = \delta(q(t), x(t)) = x(t). \]
- Similarly, a corresponding output trajectory is given by
  \[ y(t) = \lambda(q(t), x(t)) = x(t). \]

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input trajectory</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>State trajectory</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Output trajectory</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Discrete Time Model – Discussion

\[ q(t+1) = \delta(q(t), x(t)) = x(t). \]
\[ y(t) = \lambda(q(t), x(t)) = x(t) \]

- Assuming the system (device) is represented as a black box, what is the behavior of this black box?
- The output is the input with no delay (delay=0)

- Can you design a system to make the delay=1?
  \[ q(t+1) = \delta(q(t), x(t)) = x(t). \]
  \[ y(t) = \lambda(q(t), x(t)) = q(t) \]

- This example illustrates the two relations in systems theory
  - From structure to behavior: Knowing the structure allows us to deduce (analyze, simulate) its behavior.
  - From behavior to structure: the modeling (design) process
How to Simulate a Discrete Time Model

- A sequence of state, \( q(0), q(1), q(2), \ldots \) is called a state trajectory. Having an arbitrary initial state \( q(0) \), subsequent states in the sequence are determined by
  \[
  q(t+1) = \delta(q(t), x(t)) = x(t).
  \]
- Similarly, a corresponding output trajectory is given by
  \[
  y(t) = \lambda(q(t), x(t)) = x(t)
  \]

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input trajectory</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>State trajectory</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Output trajectory</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Question: write an algorithm to compute the state and output trajectories of a discrete time model given its input trajectory and its initial state.
Discrete Time Simulation

• The following algorithm is an example of a simulator for a discrete time model:

\[ T_i=0, \ T_f=9 \] 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

\[ t=T_i \]
While \( (t<=T_f) \) \{
\[ y(t) = \lambda(q(t), \ x(t)) \]
\[ q(t+1) = \delta(q(t), \ x(t)) \]
\[ t = t+1 \]
\}

Question: The computation complexity of this algorithm
One-Dimensional Cell Space

- What if we connect the above systems in a row with each system connected to its left and right neighbors?
- Imagine that each system has two states and gets the states of its neighbors as inputs. Then there are eight combinations of states and inputs as listed in the Table.
- Each complete assignment of a 0 or 1 to the eight rows results in a new transition function. There are $2^8=256$ such functions.
- What would we observe if we choose one such function and start each component with an initial state?

<table>
<thead>
<tr>
<th>Current state</th>
<th>Current left input</th>
<th>Current right input</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>
Cellular Automata

• A cellular automaton is an idealization of a physical phenomenon in which space and time are discredited and the state sets are discrete and finite.
• Cellular automata have components, called cells, which are all identical with identical computational apparatus.
• They are geometrically located on a one-, two-, or multidimensional grid and connected in a uniform way.
• The cells influencing a particular cell, called the neighborhood of the cell, are often chosen to be the cells located nearest in the geometrical sense.
• Time is also discrete, and the state of a cell at time $t$ is a function of the states of a finite number of cells (called its neighborhood) at time $t - 1$. Each time the rules are applied to the whole grid a new generation is created.
• Cellular automata were originally introduced by von Neumann and Ulam as idealization of biological self-production.
An Example set of Rules

<table>
<thead>
<tr>
<th>current pattern</th>
<th>111</th>
<th>110</th>
<th>101</th>
<th>100</th>
<th>011</th>
<th>010</th>
<th>001</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>new state for center cell</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Time Step

Let's simulate it together.

Dr. Xiaolin Hu
One Dimensional Cellular Automata

Below are tables defining the "rule 30 CA" and the "rule 110 CA" (in binary, 30 and 110 are written 11110 and 1101110, respectively) and graphical representations of them starting from a 1 in the center of each image:

Rule 30 exhibits *class 3* behavior, meaning even simple input patterns such as that shown lead to chaotic, seemingly random histories.

![Rule 30 cellular automaton](https://en.wikipedia.org/wiki/Cellular_automaton)

<table>
<thead>
<tr>
<th>current pattern</th>
<th>111</th>
<th>110</th>
<th>101</th>
<th>100</th>
<th>011</th>
<th>010</th>
<th>001</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>new state for center cell</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
One Dimensional Cellular Automata (cont.)

Rule 110, like the Game of Life, exhibits what Wolfram calls *class 4* behavior, which is neither completely random nor completely repetitive. Localized structures appear and interact in various complicated-looking ways.

https://en.wikipedia.org/wiki/Cellular_automaton

<table>
<thead>
<tr>
<th>current pattern</th>
<th>111</th>
<th>110</th>
<th>101</th>
<th>100</th>
<th>011</th>
<th>010</th>
<th>001</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>new state for center cell</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Cellular Automata

- Wolfram systematically investigated all possible transition functions of one-dimensional cellular automata.
- He found out that there exist four types of cellular automata that differ significantly in their behavior:
  - Automata where any dynamic soon die out.
  - Automata that soon come to periodic behavior
  - Automata that show chaotic behavior
  - And the most interesting ones, automata whose behaviors are unpredictable and non-periodic but that showing interesting, regular patterns.
Game of Life

• Conway’s Game of Life is framed within a two-dimensional cell space structure
• Each cell is in one of two possible states, live or dead. Every cell interacts with its eight neighbours, which are the cells that are directly horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:
  – Any live cell with fewer than two live neighbours dies, as if by loneliness.
  – Any live cell with more than three live neighbours dies, as if by overcrowding.
  – Any live cell with two or three live neighbours lives, unchanged, to the next generation.
  – Any dead cell with exactly three live neighbours comes to life.
The game of life

  - Conway originally conjectured that no pattern can grow indefinitely—i.e., that for any initial configuration with a finite number of living cells, the population cannot grow beyond some finite upper limit. In the game’s original appearance in "Mathematical Games", Conway offered a $50 prize to the first person who could prove or disprove the conjecture before the end of 1970. The prize was won in November of the same year by a team from the Massachusetts Institute of Technology, led by Bill Gosper; the "Gosper glider gun" produces its first glider on the 15th generation, and another glider every 30th generation from then on. For many years this glider gun was the smallest one known. In 2015 a period-120 gun was discovered that has fewer live cells but a larger bounding box.

- [http://www.bitstorm.org/gameoflife/](http://www.bitstorm.org/gameoflife/)
Why Is Game of Life Interesting
(http://math.com/students/wonders/life/life.html)

- It is one of the simplest examples of what is sometimes called "emergent complexity" or "self-organizing systems."
- It is the study of how elaborate patterns and behaviors can emerge from very simple rules. It helps us understand, for example, how the petals on a rose or the stripes on a zebra can arise from a tissue of living cells growing together. It can even help us understand the diversity of life that has evolved on earth.
- In Life, as in nature, we observe many fascinating phenomena. Nature, however, is complicated and we aren't sure of all the rules. The game of Life lets us observe a system where we know all the rules.
- The rules described above are all that's needed to discover anything there is to know about Life, and we'll see that this includes a great deal. Unlike most computer games, the rules themselves create the patterns, rather than programmers creating a complex set of game situations.
Cellular Automata Simulation Algorithm

- How to develop a CA simulation algorithm?
- The basic procedure for simulating a cellular automaton follows the discrete time simulation algorithm introduced earlier.
- At every time step we scan all cells, applying the state transition function to each, and saving the next state in a second copy of the global state data structure.
- Then the clock advances to the next step.
- To do this we need to limit the space to a finite region.
- How to take care of the boundary cells?
- What is the computation complexity of this algorithm?
The Algorithm

```java
// Assume 2D array board[][] has the initial state
for(int time=0; time<EndTime; time++) {
    int[][] next = new int[columns][rows];
    // Looping but skipping the edge cells
    for (int x = 1; x < columns - 1; x++) {
        for (int y = 1; y < rows - 1; y++) {
            // Add up all the neighbor states to calculate the number of live neighbors.
            int neighbors = 0;
            for (int i = -1; i <= 1; i++) {
                for (int j = -1; j <= 1; j++) {
                    neighbors += board[x+i][y+j];
                }
            }
            // Correct by subtracting the cell state itself.
            neighbors -= board[x][y];
            // The rules of life!
            if ((board[x][y] == 1) && (neighbors < 2)) next[x][y] = 0;
            else if ((board[x][y] == 1) && (neighbors > 3)) next[x][y] = 0;
            else if ((board[x][y] == 0) && (neighbors == 3)) next[x][y] = 1;
            else next[x][y] = board[x][y];
        }
    }
    // The 2D array "next" is now the current board.
    board = next;
}
```

Now consider an agent-based pedestrian crowd simulation. At each time step, each agent makes a decision of movement (based on its current state and its surrounding situation) and then carries out the movement.

- Write an algorithm to simulate this model.
- Before we do that, let’s formulate the problem first.
- How to formulate the problem in a way similar to the discrete time model discussed before?
- What is q? what is x? what is y?
- What is The δ function? What is the λ function?
Discussion (cont.) – Start form a single agent

The robot wanders around and avoids obstacles ahead of it.

- How to formulate the problem in a way similar to the discrete time model discussed before?
- What is q? what is x? what is y?
- What is the δ function? What is the λ function?

\[ q = <\text{position, action}>, \quad x = <\text{environment Input}>, \quad y = <\text{position}> \]

\[ \delta = \{ \text{decide the next action based on current position and current environment input; update the next position based on current position and next action} \} \]

\[ \lambda = \text{current position} \]
Discussion

Now consider an agent-based pedestrian crowd simulation. At each time step, each agent makes a decision of movement (based on its current state and its surrounding situation) and then carry out the movement.

- \( q = \langle \text{position, action} \rangle, \ x = \langle \text{other agents’ positions} \rangle, \ y = \langle \text{position} \rangle \)
- The \( \delta \) function
  - Make a decision…
  - Execute the action …
- Write an algorithm to simulate this model.

- When to execute the action?
  - Execute after all agents finish their decision makings (do not update position until all agents finishes decision making).
- The computation complexity of this algorithm?
  - \( O(N^T) \) or \( O(N^N T) \)?
- How to reduce the computation complexity?
Why is the computation complexity $O(N^2T)$ instead of $O(N^2T)$?
Cellular Automata Simulation Algorithm

- The basic procedure for simulating a cellular automaton follows the discrete time simulation algorithm introduced earlier.
- At every time step we scan all cells, applying the state transition function to each, and saving the next state in a second copy of the global state data structure.
- Then the clock advances to the next step.
- What is the computation complexity of this algorithm?
- A more efficient approach?
Towards Discrete Event Approach to Cellular Automata Simulation

• In discrete time systems, at every time step each component undergoes a “state transition”; this occurs whether or not its state actually changes.
• Often, only small number of components really change.
• Define an event as a change in state (e.g., births and deaths in the Game of Life).
• A discrete event simulation algorithm concentrates on processing events rather than cells and is inherently more efficient.
• How to design the algorithm?
Towards Discrete Event Approach to Cellular Automata Simulation

• The basic idea is to try to predict whether a cell will possibly change state or will definitely be left unchanged in the next global state transition.

• A cell will not change state at the next state transition time, if none of its neighboring cells changed state at the current state transition time.

• Why?

\[
q(t+1) = \delta(q(t), X(t))
\]
\[
X(t) = \{q_{\text{neighbor}}(t)\}
\]

If \( q(t) \) and \( X(t) \) no change, \( \delta() \rightarrow \) same result.

In a state transition mark those cells which actually changed state. From those, collect the cells that are their neighbors. The set collected contains all cells that can possibly change at the next step. All other cells will definitely be left unchanged.
// Assume 2D array board[][] has the initial state
List activecell;
add all cells into the activecell list
for(int time=0;time<EndTime; time++){
    List new_activecell;
    int[][] next = new int[columns][rows];
    //Looping but skipping the edge cells
    for (every cell in the activecell list) {
        //Add up all the neighbor states to calculate the number of live neighbors.
        int neighbors = 0;
        for (int i = -1; i <= 1; i++) {
            for (int j = -1; j <= 1; j++) {
                neighbors += board[x+i][y+j];
            }
        }
        //Correct by subtracting the cell state itself.
        neighbors -= board[x][y];
        //The rules of life!
        if ((board[x][y] == 1) && (neighbors <  2)) next[x][y] = 0;
        else if ((board[x][y] == 1) && (neighbors >  3)) next[x][y] = 0;
        else if ((board[x][y] == 0) && (neighbors == 3)) next[x][y] = 1;
        else next[x][y] = board[x][y];
        if(board[x][y]!=next[x][y])
            add cell_x_y and its neighbors into the new_activell list, remove duplicate
    }
    //Prepare for the next iteration; assign the active cells’ new states to the board[][]
    for (every cell in the activecell list)
        board[x][y]=next[x][y];
    activecell = new_activecell;
}

• Note: this is still a discrete time model.
• Question: What is the computation complexity of this approach for cellular automata simulation?
Discrete Event Model

- Discrete event models have many applications. Examples:
  - Queue Model (Example: the service lines in a bank)
  - Process workflow system such as manufactory system, supply chain
  - Control systems (Example: rail road dispatch control)
  - Ecological systems with the happenings of significant phenomenon (such as occurrence of a fire)
  - Computer networks (driven by arrival of packages or completion of tasks)
  - Any system where the concept of “change” is important even they are traditionally modeled by continuous models or discrete time models. (Example: decision making of a human being driven by changes in perception).

- Discrete time model can be thought as a special case of discrete event model with each time step as an event.
A Cellular Automata With Fitness

• The original version of Game of Life assumes that all births and deaths take the same time (equal to a time step)

• A more accurate representation assumes birth and death dependent on a quantity called fitness.

• A cell attains positive fitness when its neighborhood is supportive, that is, when it has exactly 3 neighbors, and the fitness will diminish rapidly when its environment is hostile (<2 or >3 neighbors).

• Whenever the fitness reaches 0, the cell will die. A dead cell will have a negative fitness -2. When the environment is supportive and the fitness crosses the zero level, the cell will born.

• Assuming the fitness increase rate is 1 per second, decrease rate is -3 per second. A cell can have maximum fitness of 6.
An example

- What is needed to model such a process?
Discussion

• Discrete Time Modeling approach
  – What are the q, x, y, delta function, and output function?
  – (fitness should be part of q)

• Discrete Event Modeling approach
  – Consider one cell first
  – Then the whole cell space

For the discrete time model, what happens if the birth or death time is not integer?
Event-based Approach

- Concentrate on the interesting events only, namely births and deaths, as well as the changes in the neighborhood.
- To do that, we need a means to determine when interesting things happen.
- Events can be caused by the environment, such as the changes of the sum of alive neighbors. The occurrence of such external events are not under the control of the model component itself.
- On the other side, the component may schedule events to occur. Those are called internal events.
- Given a particular state, e.g., a particular fitness of the cell, a time advance is specified as the time it takes until the next internal event occurs, supposing that no external event happens in the meantime.
The example in the discrete event format
Scheduling

• How does time advance (move forward) in a discrete time model and a discrete event model?
• The concept of “scheduling” is central in discrete event modeling and simulation.
• In a discrete time model, scheduling is implicit because the time advances in a fixed time step fashion.
• In a discrete event model, at any state, the model needs to explicitly schedule the next event.
  – Given a particular state, a time advance is specified as the time it takes until the next internal event occurs, supposing that no external event happens in the meantime.
Discrete Event Simulation

• In discrete event simulation, one has to execute the scheduled internal events of the different cells at their event times.
• Moreover, at any state change through an internal event we must take care to examine the cell’s neighbors for possible state changes.
• A change in state may affect waiting times as well as result in scheduling of new events and cancellation of events.
Discrete Event Simulation

• We see that the effect of a state transition may not only be to schedule new events, but also to cancel events that were scheduled in the past. (see from (a) to (b) in the figure)
• Furthermore, see from (b) to (c) in the figure, the system can jump from the current time 1 to next event time 3. This illustrates efficiency advantage in discrete event simulation – during times when no events are scheduled, no components need to be scanned.
• The situation at time 3 (Figure (b) also illustrates a problem in discrete event simulation – that of simultaneous events.
• Who goes first?
  – All simultaneous events undergo their state transitions together.
  – Define a priority among the components.
Event Scheduling

- Event scheduling is a basic approach in discrete event simulation.
- Because of its simplicity, event scheduling simulation is the preferred strategy when implementing customized simulation systems in procedural programming languages.
- The event scheduling utilizes a event list, which stores a list of events that are ordered by increasing scheduling times.
- The event with earliest scheduled time is removed from the list and the clock is advanced to the time of this imminent event. The routine associated with the imminent event is executed.

<table>
<thead>
<tr>
<th>event name</th>
<th>scheduled time</th>
</tr>
</thead>
<tbody>
<tr>
<td>event1</td>
<td>5</td>
</tr>
<tr>
<td>event2</td>
<td>23</td>
</tr>
<tr>
<td>event3</td>
<td>45</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

routine associated with the event

…..
…..
…..
A tie-breaking procedure is employed if there is more than one such imminent events.

Execution of the event routine may cause new events to be added in the proper place on the list. Also, existing events may be rescheduled or even canceled.

What does an event routine do?

- In general, if an event is associated with a component (e.g., a cell), execution of the event routine will cause this component to schedule a new event. Meanwhile, state variables may be updated.
- Furthermore, for all the components influenced by this component (e.g., neighboring cells), execution of the event routine may change the events associated with those components.

The next cycle now begins with the clock advance to the earliest scheduled time.
### Event list Scheduling

<table>
<thead>
<tr>
<th>Event</th>
<th>Scheduled Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>3</td>
</tr>
<tr>
<td>e2</td>
<td>5</td>
</tr>
<tr>
<td>e3</td>
<td>10</td>
</tr>
</tbody>
</table>

- Clock = 3
- Remove and execute e1
- State variables may be updated
- May cause the scheduling of e4 at time 6

<table>
<thead>
<tr>
<th>Event</th>
<th>Scheduled Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>e2</td>
<td>5</td>
</tr>
<tr>
<td>e4</td>
<td>6</td>
</tr>
<tr>
<td>e3</td>
<td>10</td>
</tr>
</tbody>
</table>

- Clock = 5

---

Dr. Xiaolin Hu
**Event Scheduling Example**

**Event: Generate_Job**
- \( nr\text{-waiting} = nr\text{-waiting} +1 \)
- schedule a Generate_Job in inter-gen-time
- if \( nr\text{-waiting} = 1 \) then
  - schedule a Process_Job in service-time

**Event: Process_Job**
- \( nr\text{-waiting} = nr\text{-waiting} -1 \)
- if \( nr\text{-waiting} >0 \) then
  - schedule a Process_Job in service-time

**Break-Ties** by: Process_Job then Generate_Job

Dr. Xiaolin Hu
Event list Scheduling

Exercise: Hand execute the simulation algorithm for several cases: for example

- (1) inter-gen-time > service-time;
- (2) inter-gen-time = service-time;
- (3) inter-gen-time < service-time < 2*inter-gen-time

**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

inter-gen-time = 7; service-time = 5 (how about 10?)

---

Dr. Xiaolin Hu
Event list Scheduling Exercise

- (1) inter-gen-time = 7; service-time = 5

Event: Generate_Job

\[ nr\text{-}waiting = nr\text{-}waiting + 1 \]

schedule a Generate_Job in inter-gen-time

if \( nr\text{-}waiting = 1 \) then

schedule a ProcessJob in service-time

Event: Process_Job

\[ nr\text{-}waiting = nr\text{-}waiting - 1 \]

if \( nr\text{-}waiting > 0 \) then

schedule a Process_Job in service-time
Event list Scheduling Exercise

- (2) **inter-gen-time = 7; service-time = 10**

**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
Event list Scheduling Exercise

• (1) inter-gen-time = 7; service-time = 5
• (2) inter-gen-time = 7; service-time = 10

Event: Generate_Job
nr-waiting = nr-waiting + 1
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

---

### Exercise (1)

- t = 0
- Gen_1, 7

- t = 7
- Pro_1, 12
- Gen_2, 14

- t = 12
- Pro_2, 19
- Gen_3, 21

- t = 14
- Gen_3, 21

- t = 19
- Pro_3, 26
- Gen_4, 28

### Exercise (2)

- t = 0
- Gen_1, 7

- t = 7
- Pro_1, 17
- Gen_3, 21

- t = 14
- Pro_2, 27
- Gen_3, 28

- t = 17
- Gen_3, 28

- t = 21
- Pro_3, 37
- Gen_5, 42
Discussion of Event Scheduling

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

Break-Ties by: Process_Job then Generate_Job

- This is a non-modular approach.
- The model and simulator are not separated.