Discrete Time Models

SYS-611: Simulation and Modeling

Paul T. Grogan, Ph.D.
Assistant Professor
School of Systems and Enterprises

Agenda

1. Discrete Time Simulation
2. Cellular Automata
3. Switching Automata

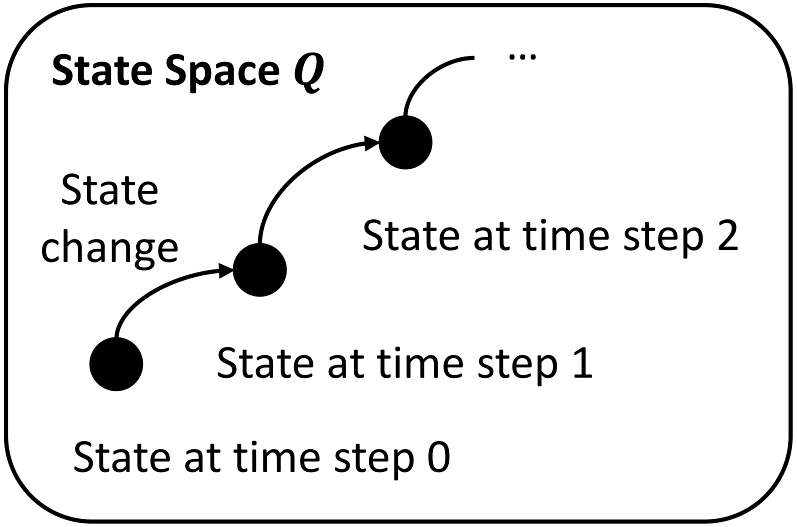
Reading: B.P. Zeigler, H. Praehofer, and T.G. Kim, “Modeling Formalisms and Their Simulators,” Ch. 3 in *Theory of Modeling and Simulation,* Academic Press, 2000, pp. 37-49.

H. Sayama, “Discrete-Time Models I: Modeling” Ch. 4 and “Cellular Automata I: Modeling,” Ch. 11 in *Introduction to Modeling and Analysis of Complex Systems*, Open SUNY Textbooks, 2015. (Free eBook online)

Discrete Time Simulation

Discrete Time Models

- Discrete or continuous state spaces (variables)
- **Dynamic**: time advances in discrete steps
  - Integer multiples of some base: 1 second, 1 hour, 1 day, 1 month, 1 year
  - No temporal unit fractions
- Applications:
  - Digital systems (clock cycle)
  - Abstract systems



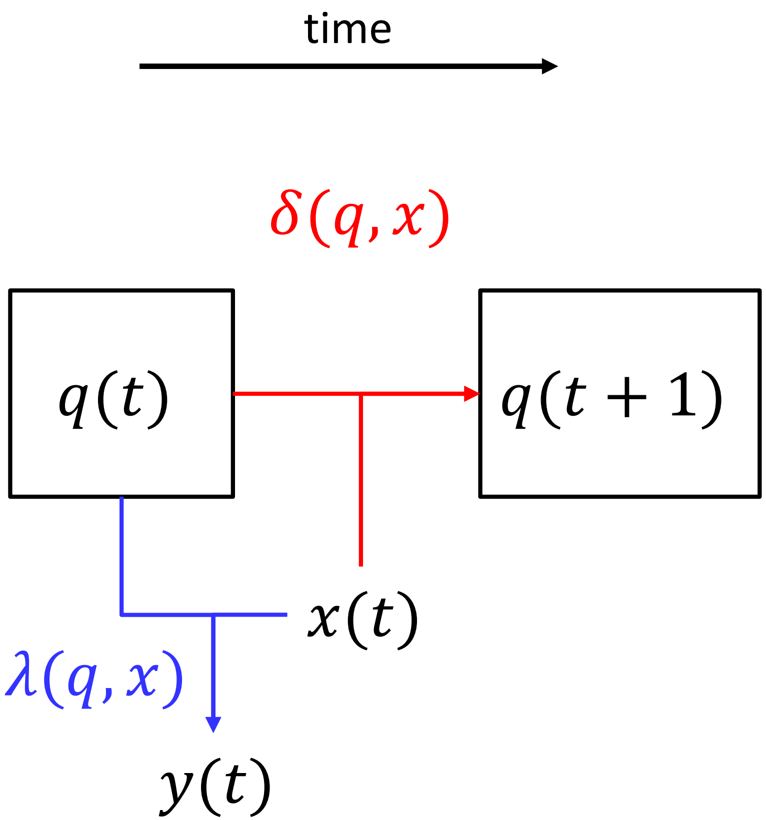
Discrete Time Notation

- $q(t)$: **state trajectory**, time history of states
- $x(t)$**: input trajectory**, time history of inputs
- $y(t)$**: output trajectory**, time history of outputs
- Next state determined by **state transition** function

$$\delta(q, x) = q(t+1)$$

- Outputs determined by **output function**

$$\lambda(q, x) = y(t)$$



Example: Delay System

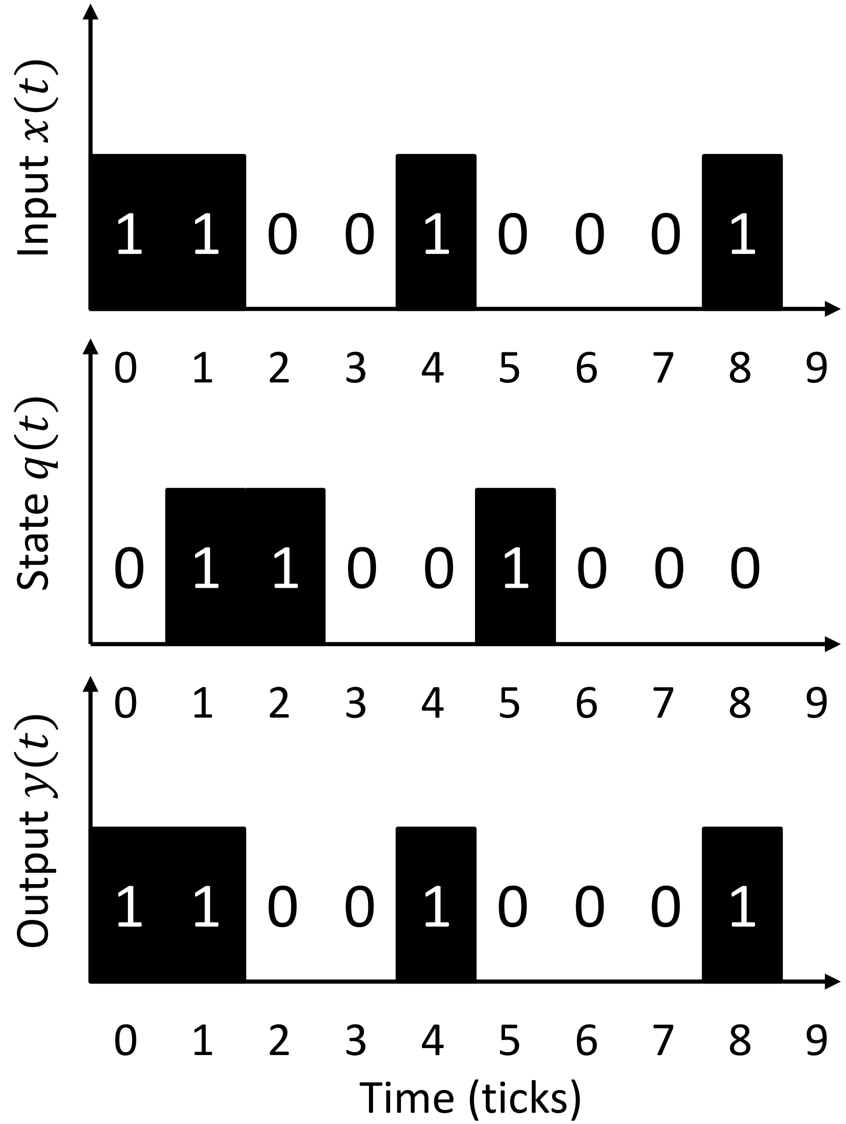
- Transition/output functions:

$$\lambda(q, x) = x$$

$$\delta(q, x) = x$$

- Transition/output (truth) table:

Current Input $x(t)$	Current State $q(t)$	Current Output $\lambda(q, x) = y(t)$	Next State $\delta(q, x) = q(t+1)$
0	**0**	0	0
1	**0**	1	1
0	**1**	0	0
1	**1**	1	1



Example: Binary Counter

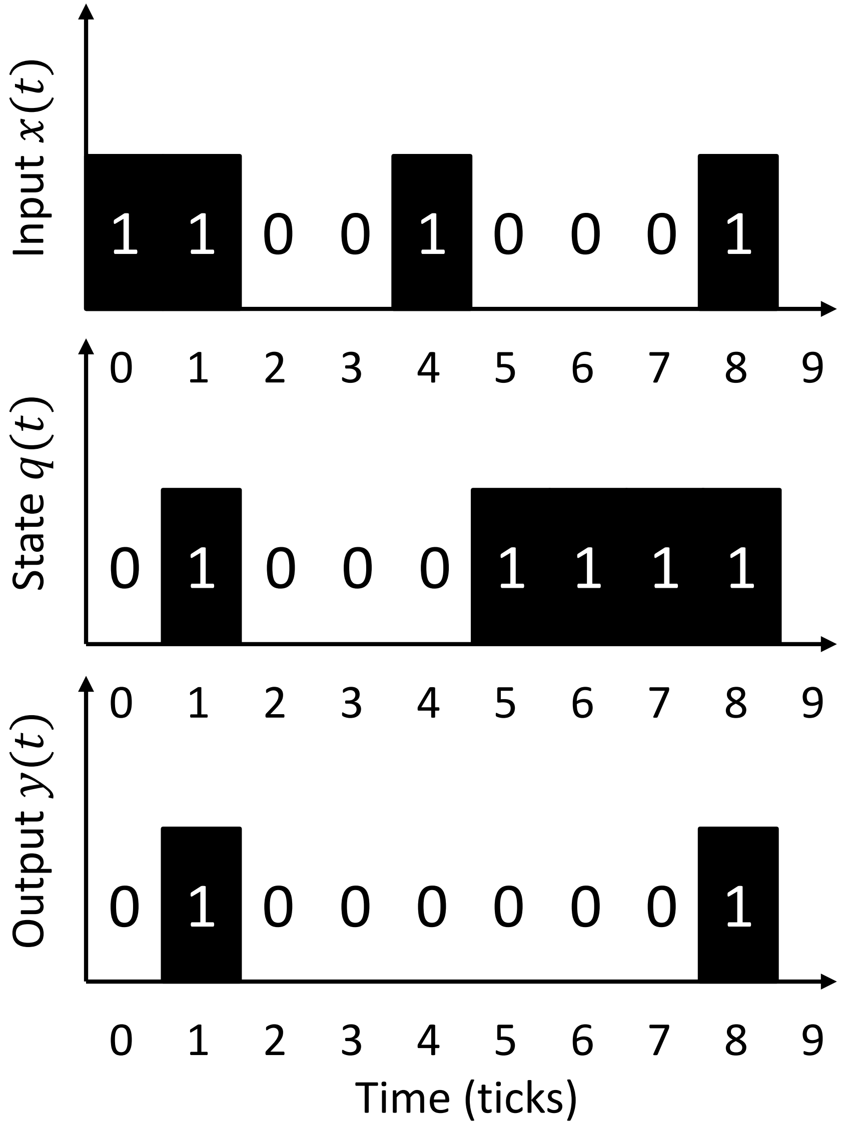
- Transition/output functions:

$$\delta(q, x) = q \text{ xor } x$$

$$\lambda(q, x) = q \text{ and } x$$

- Transition/output (truth) table:

Current Input $x(t)$	Current State $q(t)$	Current Output $\lambda(q, x) = y(t)$	Next State $\delta(q, x) = q(t + 1)$
0	**0**	0	0
1	**0**	0	1
0	**1**	0	1
1	**1**	1	0



Discrete Time Simulation

- Initialize time and state variables

$$t = 0, \qquad q(0) = q_0$$

- While terminal conditions not met:
  - Record output values

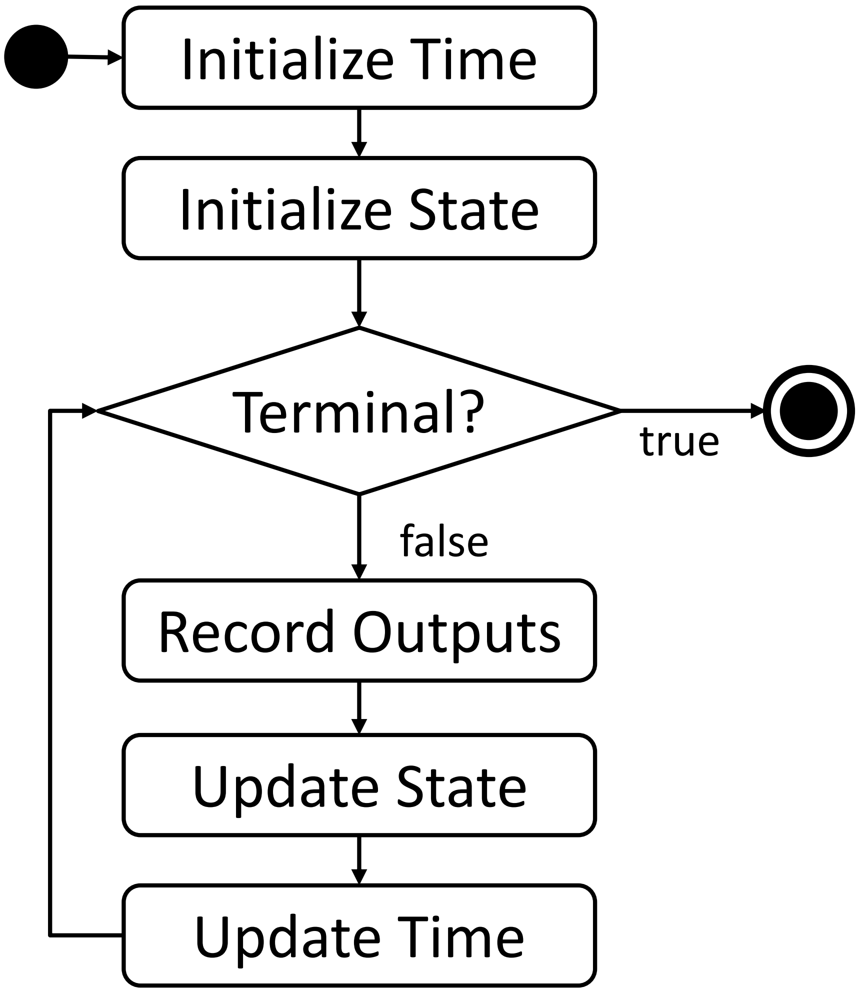
  $$y(t) = \lambda(y, x)$$

  - Compute next state

  $$q(t+1) = \delta(q, x)$$

  - Increment time

  $$t = t + 1$$



Implementing a Binary Counter

$$\delta(q,x) = q \text{ xor } x, \qquad \lambda(q,x) = q \text{ and } x$$

```
def _delta(q, x):
  return q != x
def _lambda(q, x):
  return q and x

x = [1,1,0,0,1,0,0,0,1]
y = [0,0,0,0,0,0,0,0,0]
q = [0,0,0,0,0,0,0,0,0,0]

t = 0
q[0] = 0
while t <= 8:
  y[t] = _lambda(q[t],x[t])
  q[t+1] = _delta(q[t],x[t])
  t += 1
```

9	**Binary Counter**									
10										
11	Input x(t)	1	1	0	0	1	0	0	0	1
12	State q(t)	0	=IF(XOR(B12,B11),1,0)						1	1
13	Output y(t)	0	1	0	0	0	0	0	0	1
14										
15	Time	0	1	2	3	4	5	6	7	8

9	**Binary Counter**									
10										
11	Input x(t)	1	1	0	0	1	0	0	0	1
12	State q(t)	0	1	0	0	0	1	1	1	1
13	Output y(t)	=IF(AND(B12,B11),1,0)						0	0	1
14										
15	Time	0	1	2	3	4	5	6	7	8

Modeling Dice Fighters

- Initial conditions

$$t = 0, \qquad R_0 = 20, \qquad B_0 = 10$$

- Terminal condition

$$R_t \leq 0 \text{ or } B_t \leq 0$$

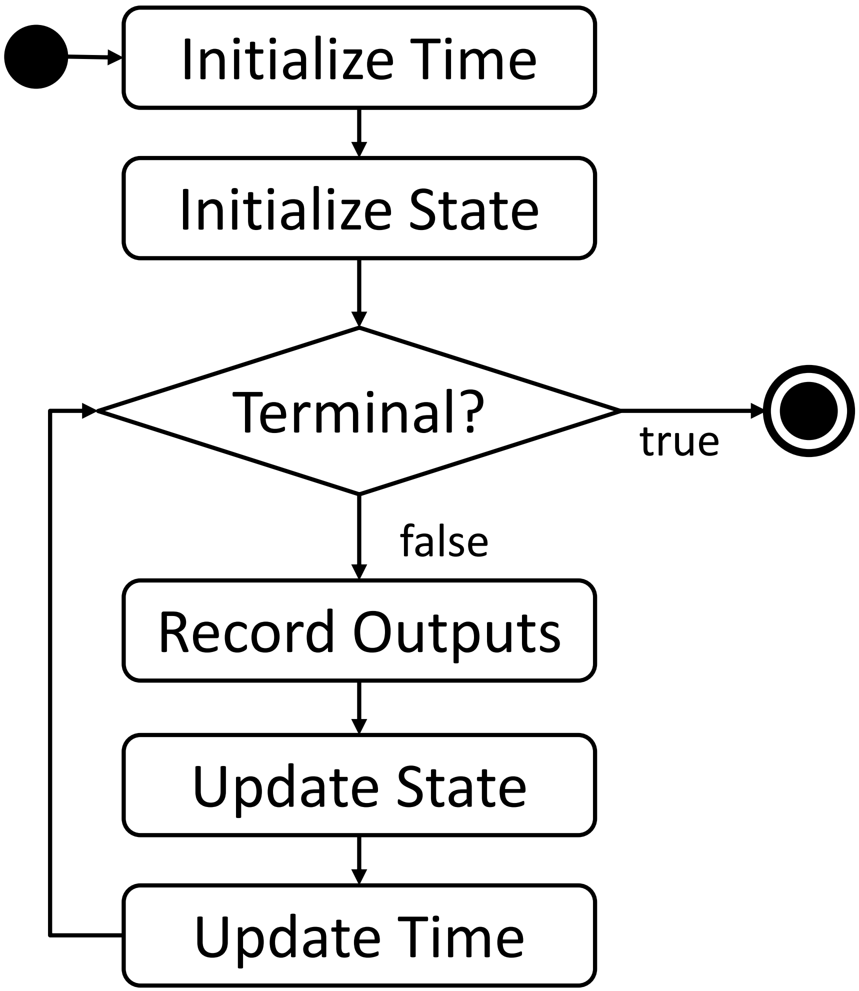
- Record outputs

$$W = \begin{cases} 2 & \text{if } R_f > 0 \\ 1 & \text{if } B_f > 0 \\ 0 & \text{otherwise} \end{cases}$$

- State transition/change

$$R_{t+1} = R_t - h_B, \qquad B_{t+1} = B_t - h_R$$

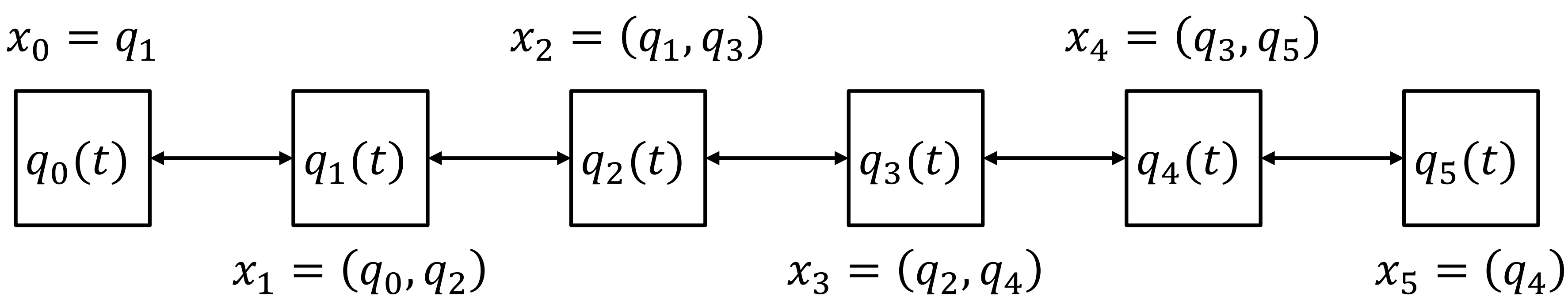
$$t = t + 1$$



Cellular Automata

Cellular Automaton

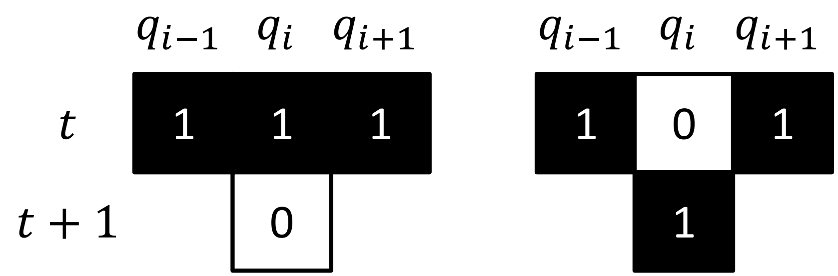
- **Cellular automaton**: idealized physical phenomenon with discrete space and time
  - Cells: components with identical state transitions
  - Local influence in a *neighborhood* of cells
  - Idealization of biological self-production



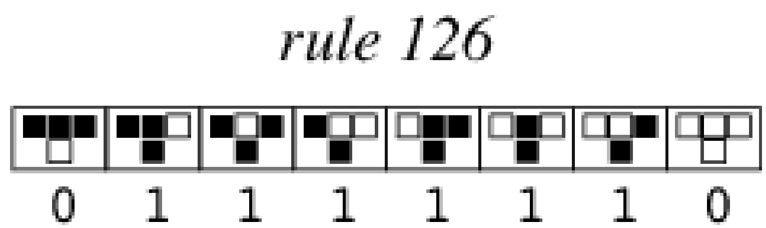
1-D Cellular Automata

Transition/output (truth) table

Current Left Input $q_{i-1}(t)$	Current Center State $q_i(t)$	Current Right Input $q_{i+1}(t)$	Next State $\delta_i(q,x) = q_i(t+1)$
0	**0**	**0**	0
0	**0**	**1**	1
1	**0**	**0**	1
1	**0**	**1**	1
0	**1**	**0**	1
0	**1**	**1**	1
1	**1**	**0**	1
1	**1**	**1**	0



- Rule #126 from Wolfram



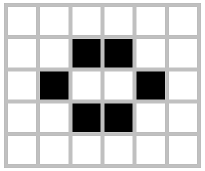
- Claim a column and compute the next states: [goo.gl/6Yw3AQ](goo.gl/6Yw3AQ)

http://mathworld.wolfram.com/ElementaryCellularAutomaton.html

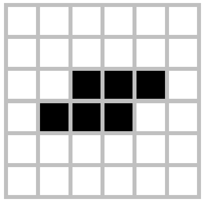
Conway’s Game of Life

- Two-dimensional cellular automata:
  - A live cell remains alive if it has between 2 and 3 live cells in its neighborhood
  - A live cell will die of overcrowding if there are more than 3 live cells in its neighborhood
  - A live cell will die of isolation if it has fewer than 2 live neighbors
  - A dead cell will become alive if it has exactly 3 alive neighbors

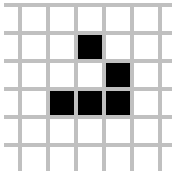
“Beehive”



“Toad”



“Glider”



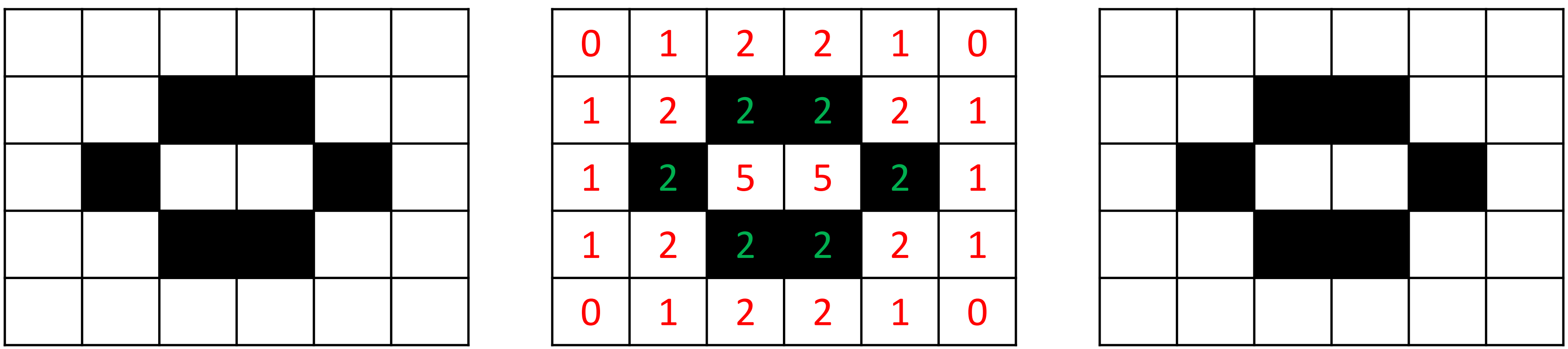
Example: Beehive

- Derived state variable: Neighbors

$$N(i,j) = q_{i-1,j-1} + q_{i-1,j} + q_{i-i,j+1} + q_{i,j-1} + q_{i,j+1} + q_{i+i,j-1} + q_{i+1,j} + q_{i+1,j+1}$$

- State transition function:

$$\delta(q_{i,j}) = \begin{cases} 1 \text{ if } N(i,j) = 3 \text{ or } \left(q_{i,j} = 1 \text{ and } N(i,j) = 2\right) \\ 0 \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$



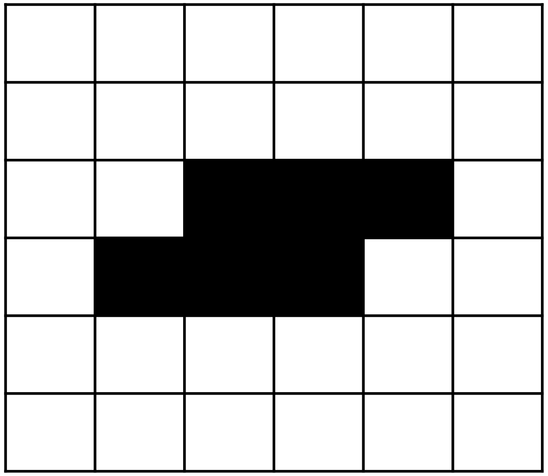
Example: Toad

- Derived state variable: Neighbors

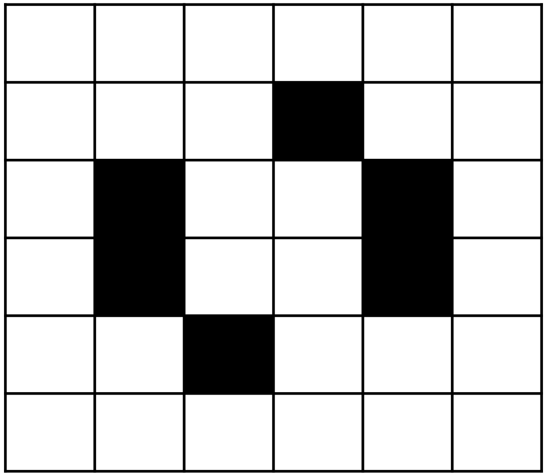
$$N(i,j) = q_{i-1,j-1} + q_{i-1,j} + q_{i-i,j+1} + q_{i,j-1} + q_{i,j+1} + q_{i+i,j-1} + q_{i+1,j} + q_{i+1,j+1}$$

- State transition function:

$$\delta(q_{i,j}) = \begin{cases} 1 \text{ if } N(i,j) = 3 \text{ or } \left(q_{i,j} = 1 \text{ and } N(i,j) = 2\right) \\ 0 \qquad\qquad \text{otherwise} \end{cases}$$



0	0	0	0	0	0
0	1	2	3	2	1
1	3	4	4	2	1
1	2	4	4	3	1
1	2	3	2	1	0
0	0	0	0	0	0



Conway's Game of Life

Transition/output (truth) table

Current Input (Neighbors) $x_{ij}(t) = N(i,j)$	Current State $q_{ij}(t)$	Next State $\delta(q_{ij}, x_{ij}) = q_{ij}(t+1)$
0	**0**	0
1	**0**	0
2	**0**	0
3	**0**	1
4	**0**	0
5	**0**	0
6	**0**	0
7	**0**	0
8	**0**	0

Transition/output (truth) table

Current Input (Neighbors) $x_{ij}(t) = N(i,j)$	Current State $q_{ij}(t)$	Next State $\delta(q_{ij}, x_{ij}) = q_{ij}(t+1)$
0	**1**	0
1	**1**	0
2	**1**	1
3	**1**	1
4	**1**	0
5	**1**	0
6	**1**	0
7	**1**	0
8	**1**	0

Switching Automata

Elementary Boolean Functions

not

Input $x(t)$	Output $y(t)$
1	0
0	1

and

Input 1 $x_1(t)$	Input 2 $x_2(t)$	Output $y(t)$
0	**0**	0
0	**1**	0
1	**0**	0
1	**1**	1

or

Input 1 $x_1(t)$	Input 2 $x_2(t)$	Output $y(t)$
0	**0**	0
0	**1**	1
1	**0**	1
1	**1**	1

xor

Input 1 $x_1(t)$	Input 2 $x_2(t)$	Output $y(t)$
0	**0**	0
0	**1**	1
1	**0**	1
1	**1**	0

Mealy and Moore Machines

Mealy Machine

- Outputs are determined by its current state and inputs:

$$y(t) = \lambda(q, x)$$

- Inputs can propagate throughout a network of components in zero time

Moore Machine

- Outputs are determined solely by its current state:

$$y(t) = \lambda(q)$$

- Inherent delay between inputs and outputs (1 tick)
- Realistic model

Delay Flip-Flop

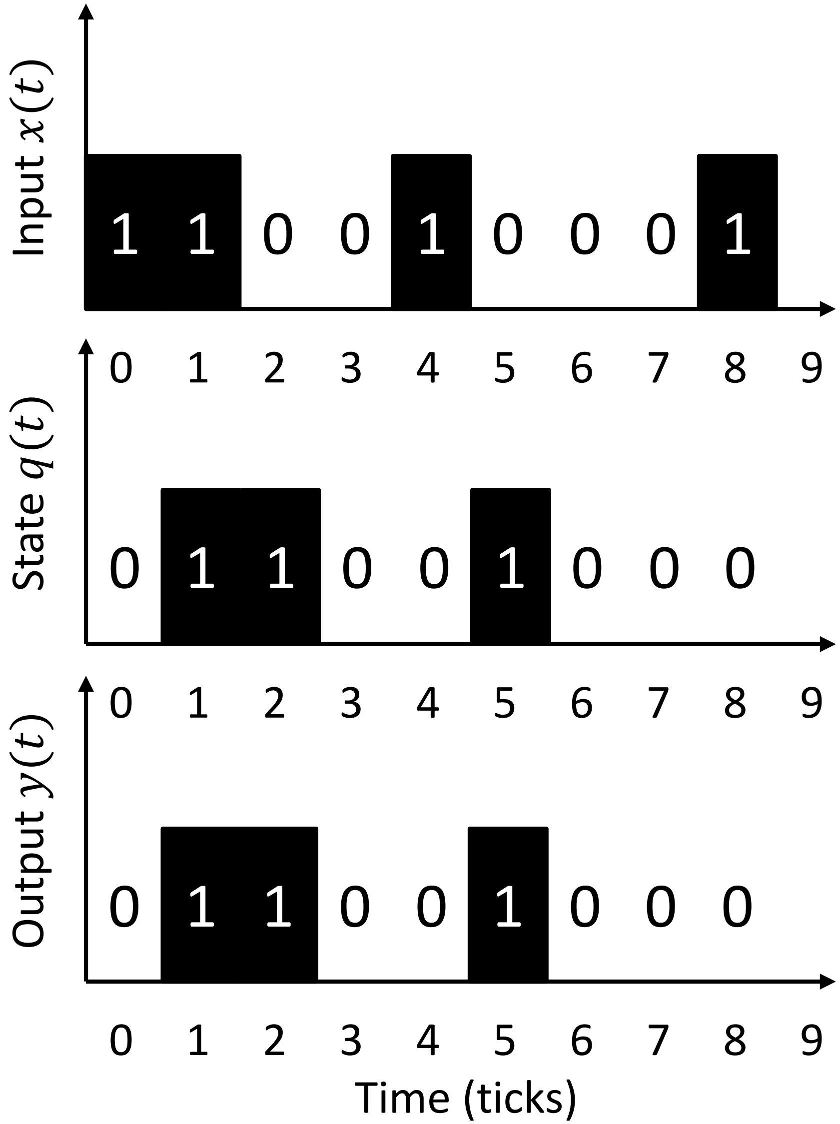
- Transition/output functions:

$$\delta(q, x) = x$$

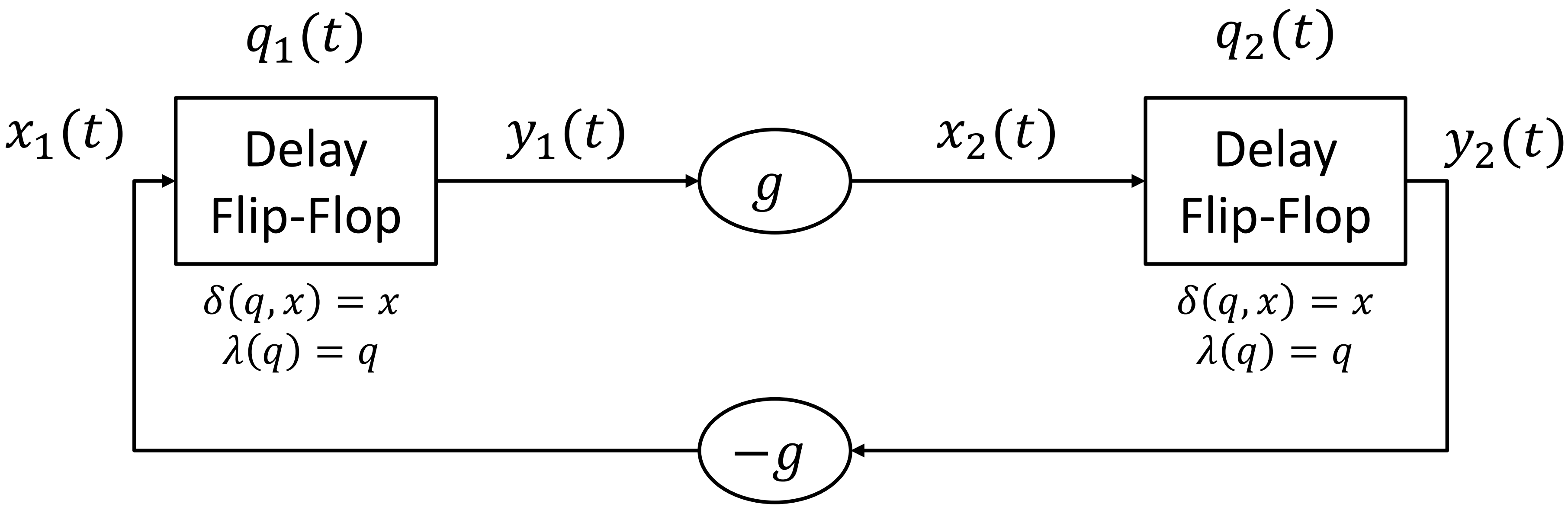
$$\lambda(q) = q$$

- Transition/output (truth) table:

Current Input $x(t)$	Current State $q(t)$	Current Output $\lambda(q) = y(t)$	Next State $\delta(q, x) = q(t+1)$
0	**0**	0	0
1	**0**	0	1
0	**1**	1	0
1	**1**	1	1



Linear Moore Network



t	0	1	2	3	4
$q_1(t)$	1	$-g$	g^2	$-g^3$	g^4
$q_2(t)$	1	g	$-g^2$	g^3	$-g^4$

$$x_1(t) = -g \cdot y_2(t)$$

$$x_2(t) = g \cdot y_1(t)$$

Linear Moore Network

```
def _delta(q, x):
  return x
def _lambda(q):
  return q

q_1 = [0,0,0,0,0,0,0,0,0,0]
q_2 = [0,0,0,0,0,0,0,0,0,0]

gain = 0.8
t = 0
q_1[0] = 0
q_2[0] = 0
while t <= 8:
  q_1[t+1] = _delta(q_1[t],
    -gain*_lambda(q_2[t]))
  q[t+1] = _delta(q_2[t],
    gain*_lambda(q_1[t]))
  t += 1
```