#07: Dynamics of Boolean Networks

In this tutorial, we study the *dynamics* of Boolean networks. Building on the construction and structural analysis from previous tutorials, we now focus on how Boolean networks evolve over time and how their long-term behavior can be characterized.

You will learn how to:

- simulate Boolean network dynamics under different updating schemes,
- compute and classify attractors,
- analyze basins of attraction,
- relate network structure to dynamical behavior.

---

0. Setup

```python
import boolforge
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

---

1. State space of a Boolean network

A Boolean network with N nodes defines a dynamical system on the discrete state space $\{0,1\}^N$.

Each state is a binary vector

$$\mathbf{x} = (x_0, \ldots, x_{N-1}) \in \{0,1\}^N,$$

where x_i denotes the state of node i.

We use a small Boolean network as a running example.

```python
string = """
x = y
y = x OR z
z = y
"""

bn = boolforge.BooleanNetwork.from_string(string, separator="=")

print("Variables:", bn.variables)
print("N:", bn.N)
print("bn.I:", bn.I)
print("bn.F:", bn.F)
```

```
Variables: ['x' 'y' 'z']
N: 3
bn.I: [array([1]), array([0, 2]), array([1])]
bn.F: [BooleanFunction(f=[0, 1]), BooleanFunction(f=[0, 1, 1, 1]), BooleanFu
nction(f=[0, 1])]
```

All state vectors follow the variable order given by `bn.variables`. For small networks, we can enumerate all 2^N states explicitly.

```
all_states = boolforge.get_left_side_of_truth_table(bn.N)
pd.DataFrame(all_states, columns=bn.variables)
```

Out[3]:

	x	y	z
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

---

2. Dynamics of synchronous Boolean networks

Under *synchronous updating*, all nodes are updated simultaneously, defining a deterministic update map

$$\mathbf{x}(t+1) = F(\mathbf{x}(t)).$$

2.1 Exact computation

```
for state in all_states:
    print(state, "-->", bn.update_network_synchronously(state))
```

```
[0 0 0] --> [0 0 0]
[0 0 1] --> [0 1 0]
[0 1 0] --> [1 0 1]
[0 1 1] --> [1 1 1]
[1 0 0] --> [0 1 0]
[1 0 1] --> [0 1 0]
[1 1 0] --> [1 1 1]
[1 1 1] --> [1 1 1]
```

This output matches the synchronous truth table representation:

```
bn.to_truth_table()
```

Out[5]:

	x(t)	y(t)	z(t)	x(t+1)	y(t+1)	z(t+1)
0	0	0	0	0	0	0
1	0	0	1	0	1	0
2	0	1	0	1	0	1
3	0	1	1	1	1	1
4	1	0	0	0	1	0
5	1	0	1	0	1	0
6	1	1	0	1	1	1
7	1	1	1	1	1	1

Each state has exactly one successor, so the dynamics consist of transient trajectories leading into *attractors* (steady states or cycles).

In this example, the network has:

- two steady states: $(0, 0, 0)$ and $(1, 1, 1)$,
- one cyclic attractor of length 2: $(0, 1, 0) \leftrightarrow (1, 0, 1)$.

Exhaustive attractor computation

```
dict_dynamics = bn.get_attractors_synchronous_exact()
dict_dynamics
```

```
Out[6]: {'Attractors': [[0], [2, 5], [7]],
         'NumberOfAttractors': 3,
         'BasinSizes': array([0.125, 0.5  , 0.375]),
         'AttractorID': array([0, 1, 1, 2, 1, 1, 2, 2], dtype=int32),
         'STG': array([0, 2, 5, 7, 2, 2, 7, 7])}
```

The returned dictionary contains:

- `STG` : the synchronous state transition graph,
- `NumberOfAttractors` ,

- `Attractors`,
- `AttractorDict`,
- `BasinSizes`.

The state transition graph can be decoded as follows:

```
for state in range(2 ** bn.N):
    next_state = dict_dynamics["STG"][state]
    print(
        state,
        "=",
        boolforge.dec2bin(state, bn.N),
        "-->",
        next_state,
        "=",
        boolforge.dec2bin(next_state, bn.N),
    )
```

```
0 = [0, 0, 0] --> 0 = [0, 0, 0]
1 = [0, 0, 1] --> 2 = [0, 1, 0]
2 = [0, 1, 0] --> 5 = [1, 0, 1]
3 = [0, 1, 1] --> 7 = [1, 1, 1]
4 = [1, 0, 0] --> 2 = [0, 1, 0]
5 = [1, 0, 1] --> 2 = [0, 1, 0]
6 = [1, 1, 0] --> 7 = [1, 1, 1]
7 = [1, 1, 1] --> 7 = [1, 1, 1]
```

Attractors can be printed in binary representation:

```
for attractor in dict_dynamics["Attractors"]:
    print(f"Attractor of length {len(attractor)}:")
    for state in attractor:
        print(state, boolforge.dec2bin(state, bn.N))
    print()
```

```
Attractor of length 1:
0 [0, 0, 0]

Attractor of length 2:
2 [0, 1, 0]
5 [1, 0, 1]

Attractor of length 1:
7 [1, 1, 1]
```

Basin sizes count how many states flow into each attractor. They always sum to 2^N.

2.2 Monte Carlo simulation

For larger networks, exhaustive enumeration is infeasible. Monte Carlo simulation approximates the attractor landscape.

```python
dict_dynamics = bn.get_attractors_synchronous(n_simulations=100)
dict_dynamics
```

```
Out[9]: {'Attractors': [[7], [5, 2], [0]],
         'NumberOfAttractors': 3,
         'BasinSizes': [40, 46, 14],
         'AttractorDict': {7: 0, 5: 1, 2: 1, 6: 0, 3: 0, 4: 1, 0: 2, 1: 1},
         'InitialSamplePoints': [3,
          5,
          5,
          6,
          3,
          4,
          6,
          5,
          6,
          4,
          0,
          7,
          7,
          5,
          4,
          7,
          4,
          3,
          1,
          1,
          1,
          4,
          7,
          1,
          0,
          3,
          2,
          6,
          3,
          7,
          0,
          0,
          1,
          2,
          5,
          6,
          6,
          5,
          5,
          1,
          2,
          6,
          2,
          0,
          1,
          7,
          7,
          1,
          5,
          2,
          5,
          7,
             2,
             7,
             3,
             3,
             4,
             4,
             7,
             0,
             0,
             2,
             1,
             0,
             3,
             3,
             3,
             0,
             0,
             2,
             4,
             7,
             0,
             7,
             2,
             4,
             7,
             3,
             7,
             6,
             7,
             3,
             6,
             7,
             0,
             2,
             4,
             2,
             7,
             2,
             0,
             4,
             5,
             0,
             7,
             4,
             1,
             3,
             1,
             4],
            'STG': {3: 7, 5: 2, 6: 7, 4: 2, 0: 0, 7: 7, 1: 2, 2: 5},
            'NumberOfTimeouts': 0}
```

The simulation returns additional information:

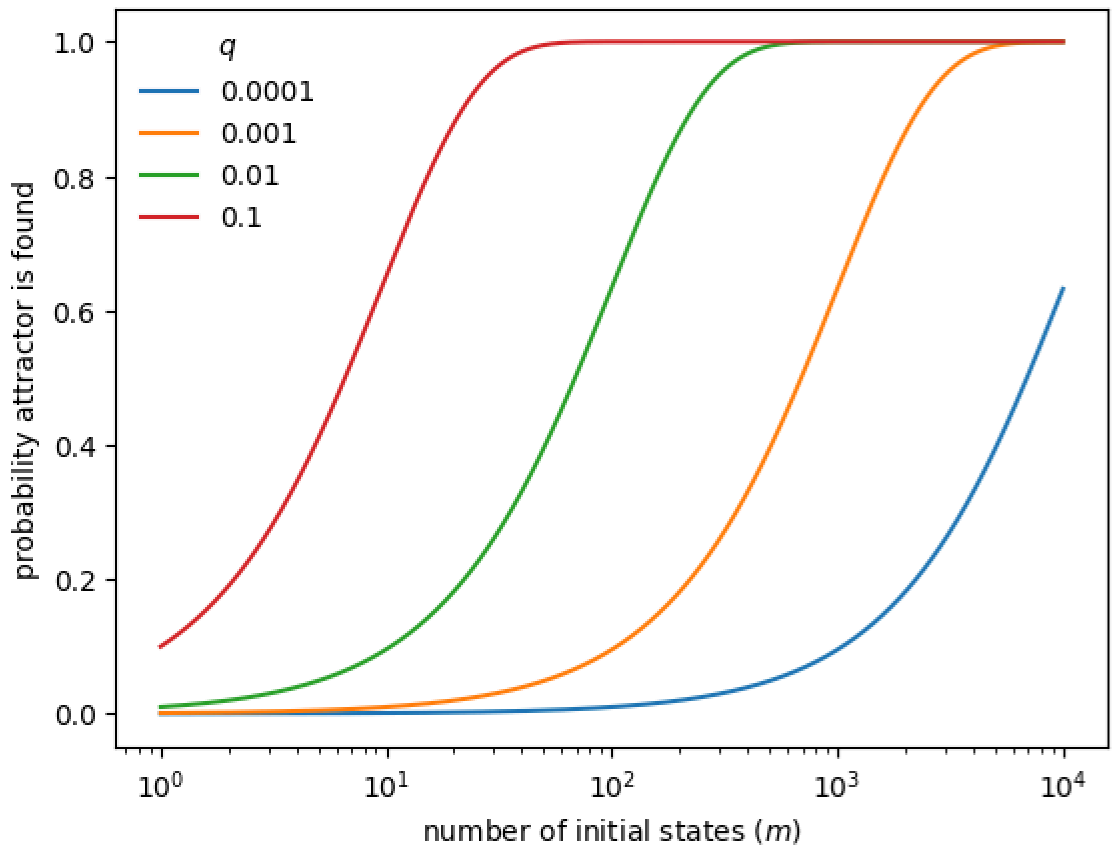
- sampled initial states,
- the number of timeouts (trajectories not reaching an attractor in time).

If an attractor has relative basin size q, the probability that it is found after m random initializations is $1 - (1 - q)^m$.

```
qs = [0.0001, 0.001, 0.01, 0.1]
ms = np.logspace(0, 4, 1000)

fig, ax = plt.subplots()
for q in qs:
    ax.semilogx(ms, 1 - (1 - q) ** ms, label=str(q))

ax.legend(title=r"$q$", frameon=False)
ax.set_xlabel("number of initial states ($m$)")
ax.set_ylabel("probability attractor is found")
plt.show()
```



---

3. Dynamics of asynchronous Boolean networks

Synchronous updating is computationally convenient but biologically unrealistic. Asynchronous updating assumes that only one node is updated at a time.

3.1 Steady states under general asynchronous update

BoolForge can compute steady states under general asynchronous updating.

```
dict_dynamics = bn.get_steady_states_asynchronous_exact()
dict_dynamics
```

```
Out[11]: {'SteadyStates': [0, 7],
          'NumberOfSteadyStates': 2,
          'BasinSizes': array([0.33333333, 0.66666667]),
          'STGAsynchronous': {0: {0: np.float64(1.0)},
           1: {1: np.float64(0.3333333333333333),
            3: np.float64(0.3333333333333333),
            0: np.float64(0.3333333333333333)},
           2: {6: np.float64(0.3333333333333333),
            0: np.float64(0.3333333333333333),
            3: np.float64(0.3333333333333333)},
           3: {7: np.float64(0.3333333333333333), 3: np.float64(0.666666666666666
         6)},
           4: {0: np.float64(0.3333333333333333),
            6: np.float64(0.3333333333333333),
            4: np.float64(0.3333333333333333)},
           5: {1: np.float64(0.3333333333333333),
            7: np.float64(0.3333333333333333),
            4: np.float64(0.3333333333333333)},
           6: {6: np.float64(0.6666666666666666), 7: np.float64(0.333333333333333
         3)},
           7: {7: np.float64(1.0)}},
          'FinalTransitionProbabilities': array([[1.        , 0.        ],
                 [0.5       , 0.5       ],
                 [0.33333333, 0.66666667],
                 [0.        , 1.        ],
                 [0.5       , 0.5       ],
                 [0.33333333, 0.66666667],
                 [0.        , 1.        ],
                 [0.        , 1.        ]])}
```

This reveals the same two steady states as in the synchronous case. In addition, the full asynchronous transition graph and absorption probabilities are returned.

BoolForge currently does not detect complex cyclic attractors under asynchronous updating; for those, specialized tools such as `pystablemotifs` are recommended.

Monte Carlo approximation

```
dict_dynamics = bn.get_steady_states_asynchronous(n_simulations=500)
dict_dynamics
```

```
Out[12]: {'SteadyStates': [7, 0],
          'NumberOfSteadyStates': 2,
          'BasinSizes': [348, 152],
          'STGAsynchronous': {(3, 1): 3,
           (3, 2): 3,
           (3, 0): 7,
           (7, 0): 7,
           (7, 2): 7,
           (7, 1): 7,
           (1, 1): 3,
           (4, 2): 4,
           (4, 0): 0,
           (0, 0): 0,
           (0, 2): 0,
           (0, 1): 0,
           (1, 0): 1,
           (1, 2): 0,
           (4, 1): 6,
           (6, 1): 6,
           (6, 0): 6,
           (6, 2): 7,
           (5, 1): 7,
           (2, 2): 3,
           (2, 1): 0,
           (5, 0): 1,
           (2, 0): 6,
           (5, 2): 4},
          'InitialSamplePoints': [3,
           1,
           4,
           3,
           0,
           1,
           4,
           5,
           4,
           2,
           3,
           2,
           7,
           6,
           5,
           3,
           5,
           4,
           6,
           6,
           2,
           0,
           3,
           7,
           3,
           1,
           2,
           0,
           3,
7,
7,
0,
4,
5,
7,
1,
0,
7,
5,
5,
7,
1,
6,
1,
6,
2,
6,
5,
5,
5,
1,
3,
3,
0,
3,
1,
7,
5,
7,
7,
4,
1,
5,
7,
2,
4,
3,
3,
4,
6,
3,
7,
3,
0,
3,
7,
3,
6,
0,
7,
5,
1,
7,
3,
4,
4,
6,
7,
3,
2,
3,
0,
0,
7,
0,
6,
2,
1,
5,
4,
2,
3,
2,
3,
1,
3,
1,
1,
6,
6,
0,
7,
5,
6,
1,
2,
4,
4,
6,
3,
2,
7,
5,
5,
5,
2,
1,
5,
0,
2,
0,
0,
3,
6,
3,
6,
0,
6,
3,
5,
6,
6,
0,
2,
2,
0,
7,
4,
6,
6,
2,
1,
0,
4,
6,
6,
6,
3,
0,
0,
2,
0,
2,
5,
3,
7,
1,
4,
1,
6,
3,
4,
1,
7,
2,
2,
2,
7,
6,
3,
4,
2,
5,
0,
6,
5,
5,
7,
0,
3,
5,
3,
4,
7,
0,
2,
3,
5,
7,
6,
4,
0,
2,
5,
4,
1,
5,
2,
6,
5,
1,
6,
3,
2,
1,
7,
0,
3,
6,
3,
7,
2,
5,
1,
2,
0,
6,
3,
1,
2,
3,
2,
3,
0,
1,
4,
2,
3,
1,
1,
2,
2,
4,
2,
2,
5,
1,
6,
6,
1,
6,
5,
7,
1,
7,
7,
2,
2,
6,
0,
6,
4,
1,
3,
3,
2,
0,
5,
3,
4,
4,
1,
6,
2,
6,
7,
3,
7,
2,
2,
5,
7,
0,
3,
2,
7,
7,
6,
1,
4,
0,
1,
7,
7,
7,
4,
0,
6,
5,
4,
7,
0,
3,
1,
5,
0,
1,
0,
6,
7,
7,
5,
3,
3,
1,
3,
5,
6,
5,
0,
6,
1,
7,
2,
7,
6,
0,
7,
6,
7,
5,
5,
7,
3,
2,
5,
2,
3,
4,
4,
4,
3,
6,
6,
3,
2,
0,
3,
7,
7,
4,
3,
1,
6,
4,
6,
5,
5,
0,
1,
2,
5,
3,
3,
5,
4,
7,
3,
7,
7,
2,
1,
7,
3,
4,
4,
3,
7,
1,
4,
2,
0,
2,
3,
3,
2,
5,
3,
5,
0,
4,
7,
1,
3,
7,
6,
6,
7,
6,
1,
6,
5,
5,
7,
5,
0,
4,
0,
3,
1,
7,
6,
1,
3,
6,
0,
3,
7,
6,
2,
7,
4,
1,
5,
0,
1,
5,
7,
2,
1,
5,
5,
6,
2,
6,
0,
7,
7,
1,
2,
1,
5,
6,
2,
6,
2,
7,
2,
2,
1,
4,
4,
7,
4,
3,
1,
6,
1,
0,
4,
4,
1,
2,
0,
2,
0,
5,
1,
1,
1,
5,
1,
1,
1,
7,
4,
7,
             1,
             2,
             3,
             0,
             5,
             5,
             4,
             4,
             4,
             5,
             5,
             2,
             2,
             0,
             2,
             4,
             6,
             6,
             2,
             1,
             0,
             0,
             5]}
```

The simulation provides:

- a lower bound on the number of steady states,
- approximate basin size distributions,
- samples of the asynchronous state transition graph.

Sampling from a fixed initial condition

```
dict_dynamics = bn.get_steady_states_asynchronous_given_one_initial_condition(
    initial_condition=[0, 0, 1], n_simulations=500
)
dict_dynamics
```

```
Out[13]: {'SteadyStates': [7, 0, 2, 9],
          'NumberOfSteadyStates': 4,
          'BasinSizes': [73, 260, 80, 87],
          'TransientTimes': [[2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
            2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2,
  2],
 [1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
  1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
            1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1,
                                1],
[2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 2],
[3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3,
  3]],
'STGAsynchronous': {(1, 0): 1,
 (1, 1): 3,
 (3, 0): 7,
 (7, 1): 7,
 (7, 2): 7,
 (7, 0): 7,
 (1, 2): 0,
 (0, 0): 0,
 (0, 2): 0,
 (0, 1): 0,
 (3, 2): 2,
 (2, 1): 2,
 (2, 2): 2,
 (2, 0): 2,
 (3, 1): 5,
 (5, 2): 5,
 (5, 1): 5,
 (5, 0): 9,
 (9, 2): 9,
 (9, 1): 9,
 (9, 0): 9},
'UpdateQueues': [[1, 3, 7],
 [1, 0],
 [1, 3, 7],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 7],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 7],
 [1, 3, 2],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 3, 7],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 5, 9],
 [1, 3, 7],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 2],
 [1, 0],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 3, 5, 9],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 3, 2],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 3, 5, 9],
 [1, 0],
 [1, 0],
 [1, 3, 7],
 [1, 3, 7],
 [1, 0],
 [1, 3, 7],
 [1, 0],
 [1, 3, 2],
 [1, 3, 7],
 [1, 0],
 [1, 3, 5, 9],
        [1, 0],
        [1, 0],
        [1, 3, 7],
        [1, 3, 7],
        [1, 0],
        [1, 3, 2],
        [1, 0],
        [1, 3, 7],
        [1, 0],
        [1, 0],
        [1, 3, 7],
        [1, 0],
        [1, 3, 2],
        [1, 3, 2],
        [1, 0],
        [1, 3, 2]]}
```

Summary and outlook

In this tutorial you learned how to:

- simulate Boolean network dynamics,
- compute synchronous attractors exactly and approximately,
- analyze basin sizes,
- compute steady states under asynchronous updating.

This concludes the function- and network-level analysis. Subsequent work focuses on large-scale dynamical analysis, perturbations, and stability in Boolean network models.