Boolean Networks & Cellular Automaton

Short introduction to python

The tutorial will be based on `python`. A basic understanding of programming and the python language will be sufficient to follow the examples.

Many good basic python tutorials and introductions exist, for instance see

- https://www.learnpython.org/ (https://www.learnpython.org/) - interactive python tutorial
- https://docs.python.org/3/tutorial/index.html (https://docs.python.org/3/tutorial/index.html) - official tutorial, more in depth

For installation instructions see for instance https://wiki.python.org/moin/BeginnersGuide/Download (https://wiki.python.org/moin/BeginnersGuide/Download)

This tutorial and all information related to it is available online at https://github.com/matthiaskoenig/modelling_ws2018 (https://github.com/matthiaskoenig/modelling_ws2018)

To follow the tutorial a basic `python3` installation with the packages listed in `requirements.txt` is needed.

```
numpy
pandas
matplotlib
```

If you have any questions or need help please contact konigmatt@googlemail.com

Boolean Networks

- A boolean network consists of nodes (which have a boolean state) and connections between the states (inputs for nodes).
- The boolean states can be either 0 or 1.
- Every node (state) in the boolean network has a rule which specifies the output of the node (state) for all possible combinations of inputs.
- Based on the given rule for a node the node state is updated.
- Simulations start from an initial state of the network. This is the state of all nodes at the begin of the simulation.

General Properties of Boolean Networks:

- Fixed topology (doesn't change with time)
- Dynamic (states
- Synchronous (update of all states occurs at the same time)
- Node States: Deterministic, discrete (binary)
- Gate Function: Boolean (rules which calculate the update for the state)
- Flow: Information

For one dimensional input the unary boolean operators are

- IDENTITY (`[0]->[0], [1]->[1]`)
- INVERSE (`[0]->[1], [1]->[0]`)
- ZERO (`[0]->[0], [1]->[0]`)
- ONE (`[0]->[1], [1]->[1]`)

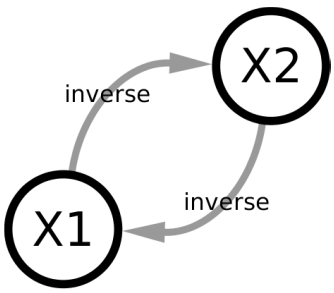
For two dimensional inputs possible logical operations (rules) are for instance

- AND (`[1,1]->[1], [1,0]->[0], [0,1]->[1], [0,0]->[0]`)
- OR
- XOR
- NOR
- ...

An overview over the truth tables (boolean rules) for unary and binary operations can be found here
https://en.wikipedia.org/wiki/Truth_table (https://en.wikipedia.org/wiki/Truth_table)

Task 1

Within this task we will simulate a boolean network by applying the rules repeatedly starting from an intial state, thereby updating the state vector `[X1, X2]` .



- Write a computer program which simulates the simple boolean networks consisting of the two nodes `X1` and `X2` with the initial state `[X1, X2](0) = [0, 1]` . The boolean rules for updating `X1` based on the input from `X2` , and for updating X2 based on the input of `X1` are the unary `INVERSE` rule. Simulate the model for 20 steps. What is the final state of the boolean network?
- What are the possible trajectories of the boolean network, i.e. which sequence of states are possible? (hint: simulate the network for all possible initial states)

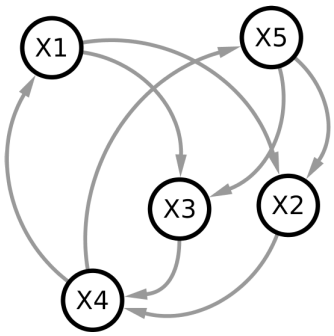
In [ ]:
```
N = 20  # number of steps
states = [[0, 1]]  # initial state
for k in N:
    # get last states
    X = states[i]

    # TODO: update state based on rules
    Xnew = ...

    # store state
    states.append(Xnew)

print(states)
```

Task 2



- Simulate the following more complex boolean network consisting of 5 nodes.
- The update rules are given by

```
X1 = INVERSE(X4)
X5 = IDENTIY(X4)
X2 = OR(X1, X5)
X3 = OR(X1, X5)
X4 = XOR(X3, X2)
```

- What are the possible trajectories of the boolean network, i.e. which final states (or cycles of states) are reached? (hint: simulate the network for all possible initial states)

Cellular automaton

A cellular automaton is a discrete model studied in computer science, mathematics, physics, complexity science, theoretical biology and microstructure modeling.

- A cellular automaton consists of a regular grid of cells, each in one of a finite number of states, such as on and off.
- For each cell, a set of cells called its neighborhood is defined relative to the specified cell.
- An initial state (time t = 0) is selected by assigning a state for each cell.
- A new generation is created (advancing t by 1), according to some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood.
- Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid simultaneously.

For more information see https://en.wikipedia.org/wiki/Cellular_automaton (https://en.wikipedia.org/wiki/Cellular_automaton) A more in depth introduction with hints how to solve the task is given in https://natureofcode.com/book/chapter-7-cellular-automata/ (https://natureofcode.com/book/chapter-7-cellular-automata/)

Task 3

Build and simulate the simple cellular automaton with the following characteristics:

- Grid. We will use the simplest possible grid, which is one-dimensional: a line of cells.
- States. The simplest set of states (beyond having only one state) are the two possible states per cell: 0 or 1.
- Neighborhood. The simplest neighborhood in one dimension for any given cell would be the cell itself and its two adjacent neighbors: one to the left and one to the right.
- Rules: Define how a cell is updated based on its neighborhood. The simplest rules are identical rules for every cell.

For the update rules we need to define an outcome (new state value 0 or 1) for a cell based on its current neighborhood. The following rules are applied:

```
neighborhood (with cell in the center) -> new state of cell
000 -> 0
001 -> 1
010 -> 0
011 -> 1
100 -> 1
101 -> 0
110 -> 1
111 -> 0
```

- the one dimensional grid contains `101` cells
- the initial state of all cells is 0, with exception of the cell in the middle of the grid which starts with state 1