Modelling Emotion Contagion – from Excel to L2-Python

Dynamic Modelling for Human-centered Systems

lecture 5

Course outline

- First lecture: what is dynamic modelling?
- Second lecture: creating models of physiological processes
- Previous lecture: creating models of social processes
- Currently: modelling behaviour
- Even later: embedding models within intelligent systems…

Excel models

Logical formalism

Today

- Explanation of the model of Emotion Contagion
  - First in Excel-representation
- Presentation of L2-Python
- Introduction to Assignment 3:
  - Emotion Contagion model in L2-Python

Syllabus chapter 7

EMOTION CONTAGION

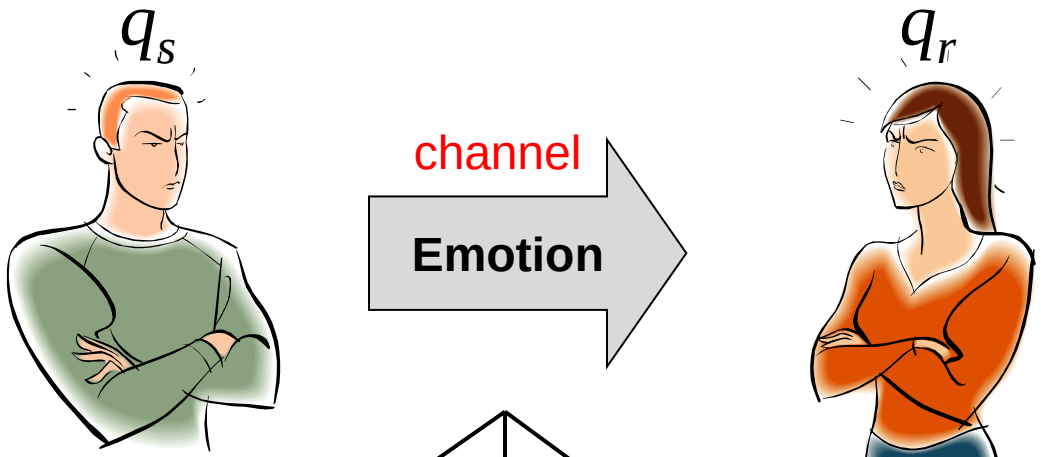
What is Emotion Contagion?

- Influencing others through conscious or unconscious induction of emotion states
- Tendency to mimic the nonverbal behavior of others; synchronization of facial expressions, vocalizations, postures, and movements" with others, in order to "converge emotionally"

Hatfield, E., Cacioppo, J.T., and Rapson, R.L. (1994). Emotional contagion. New York: Cambridge University Press.

Domain model

main concepts



sender

receiver

ε_s

expressiveness of sender

α_{sr}

channel strength

δ_r

openness of receiver

contagion strength γ_{sr} from a sender to a receiver

$$\gamma_{sr} = \varepsilon_s \alpha_{sr} \delta_r$$

Domain model
main concepts

Variables

- the level of a person *A*'s emotion q_A

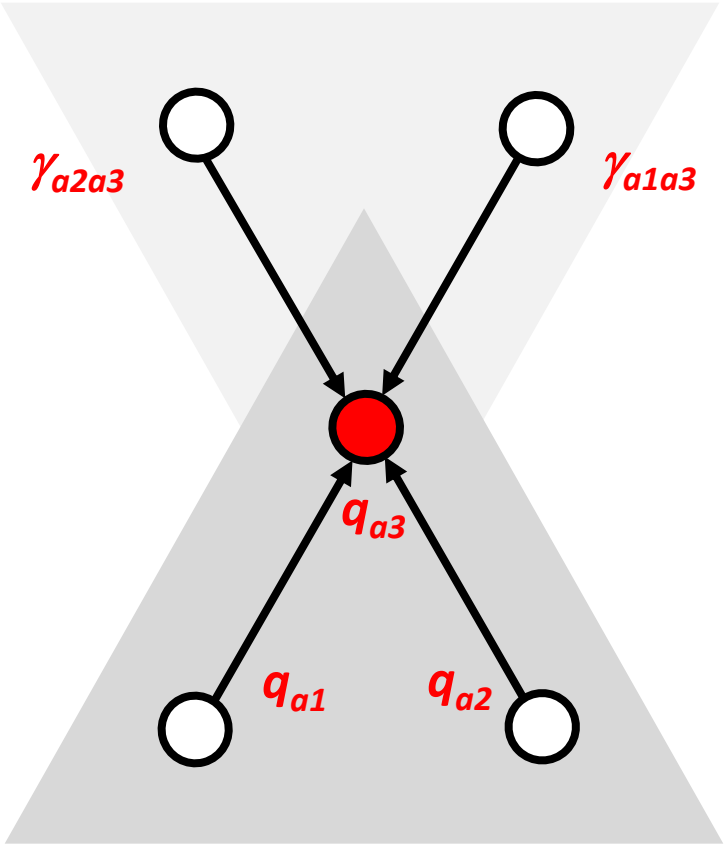
Parameters

- a person *A*'s emotion expressiveness ε_A
- a person *A*'s openness for received emotion δ_A
- the strength of the channel from sender *B* to receiver *A* α_{BA}
- the contagion strength from a sender *B* to a receiver *A* γ_{BA}

Note that some of the parameters can also be manipulated

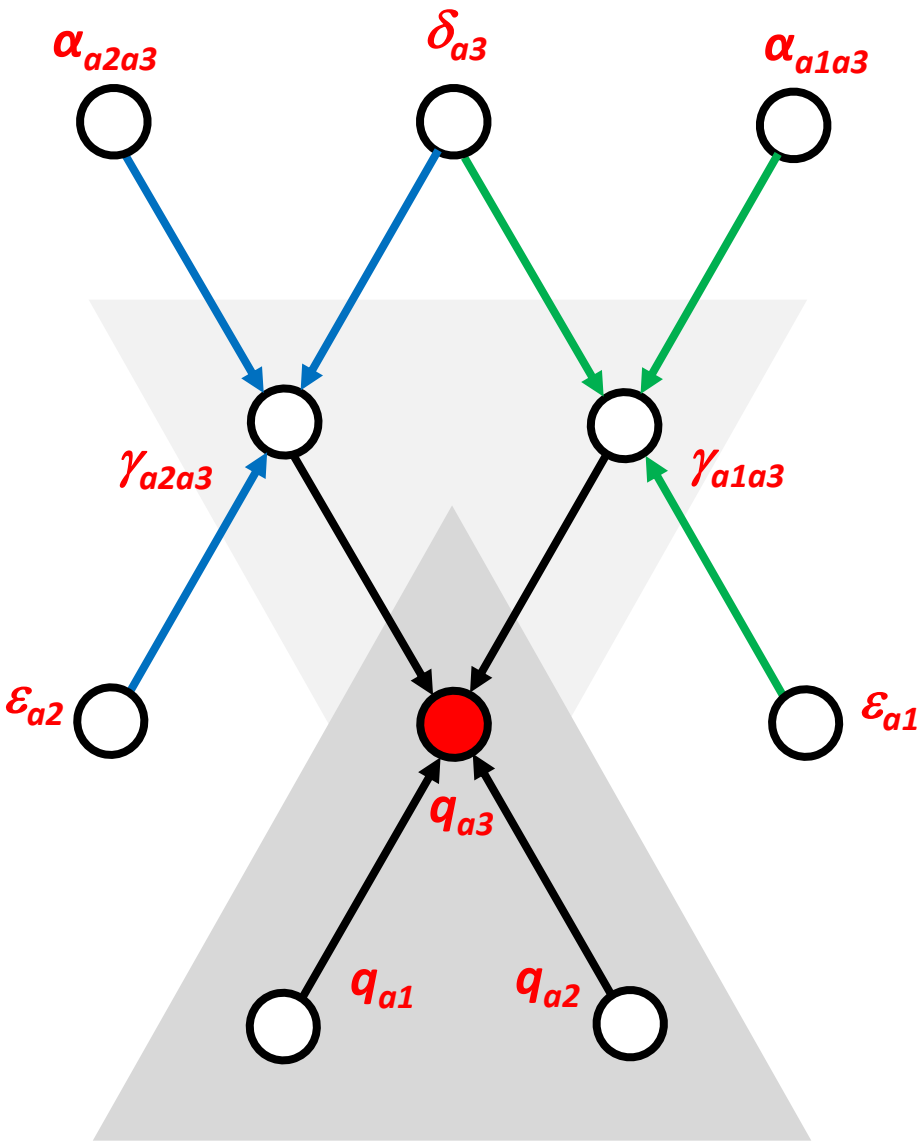
Domain model for 3 persons:

Relationships affection *a3*

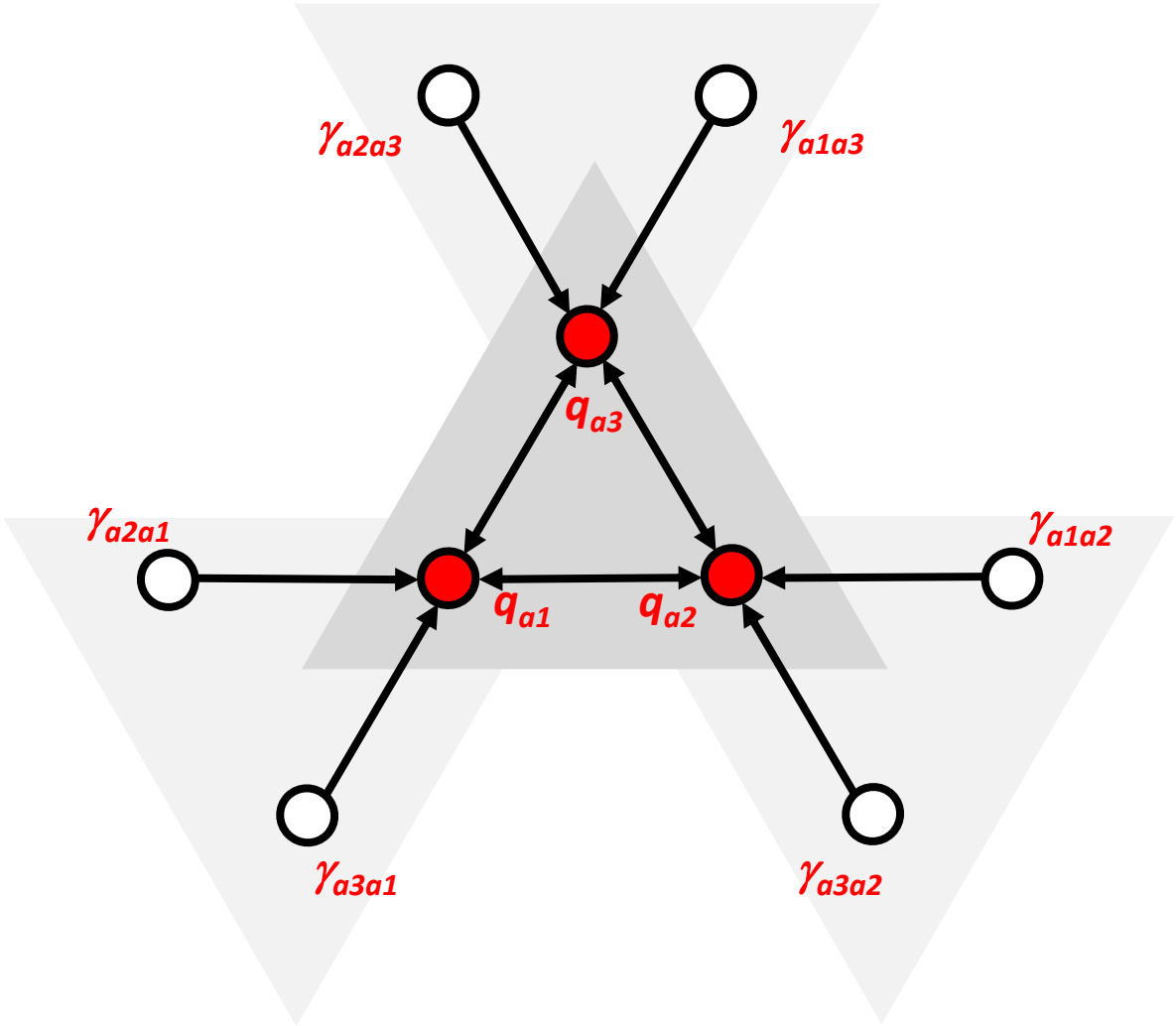


Domain model for 3 persons:

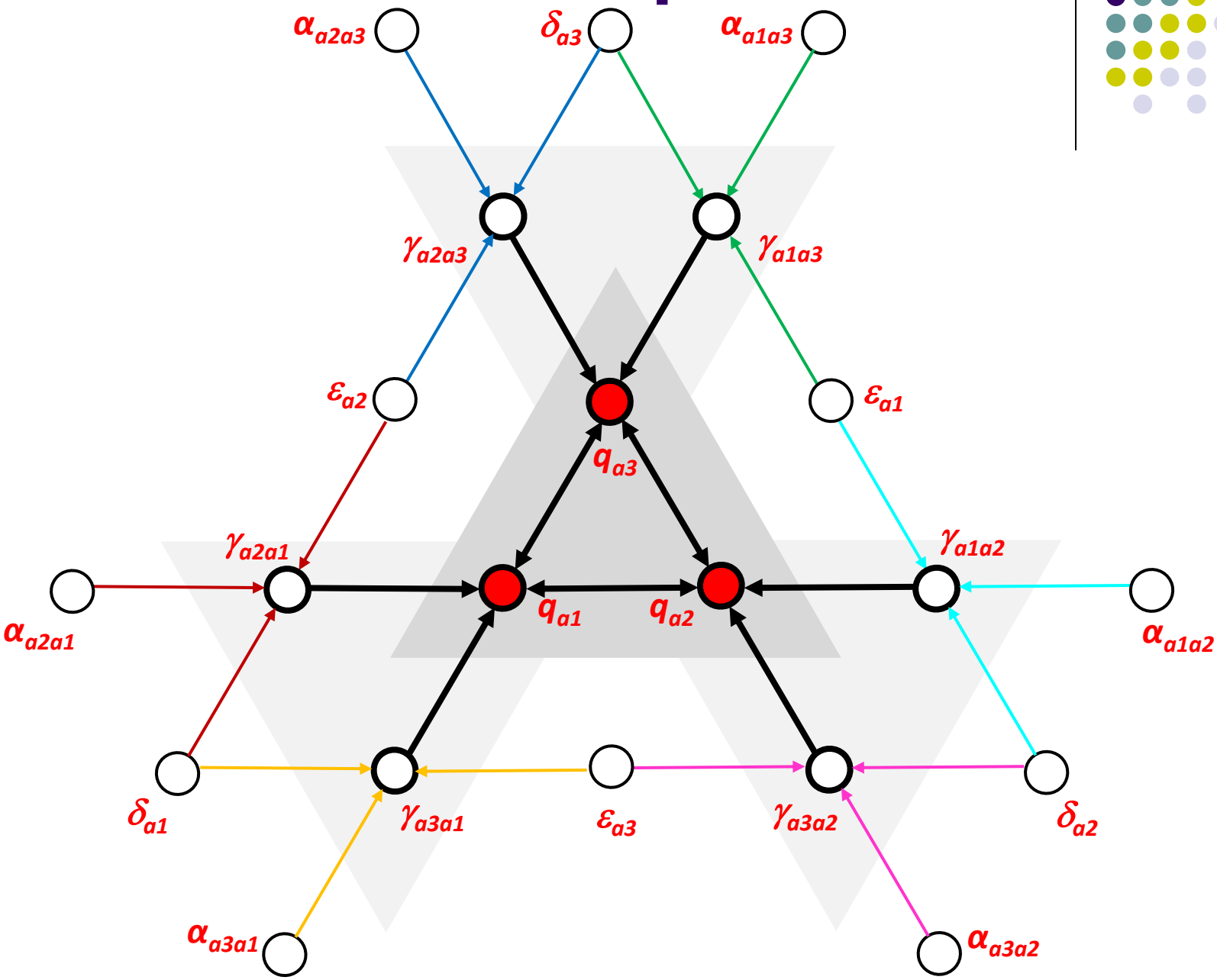
Relationships affection *a3*



Domain model for 3 persons



Domain model for 3 persons



Detailed domain model

General idea

At each time point, for each group member ***A***:

- Determine the **average emotion impact** q_A^* of the other group members on ***A***, taking into account that not all members have an equal impact on ***A***
- Determine the **difference** $q_A^* - q_A$ of this average q_A^* with the current emotion level q_A of person ***A***
- **Adjust** the emotion level q_A by a fraction γ of this difference, so that the new emotion level for ***A*** becomes $q_A + \gamma(q_A^* - q_A)$

- Alternative formulation:
  - that the new emotion level of ***A*** becomes a **weighted average** of the old emotion level q_A and the average value q_A^* of the other group members:
    $(1-\gamma)\, q_A + \gamma\, q_A^*$

In short:

The formalization of the model for emotion contagion is can be expressed as:

$$q_A(t+\Delta t) = q_A(t) + \gamma_A (q_A^* - q_A) \Delta t$$

for all group members ***A***.

Formalisation rules

- For agent A in a model of three agents:

ContagionStrengthBA(t+Δt) =

*ExpressivenessB(t) * ChannelStrengthAB(t) * OpennessA(t)*

…

EmotionlevelA(t+Δt) = EmotionlevelA(t) +

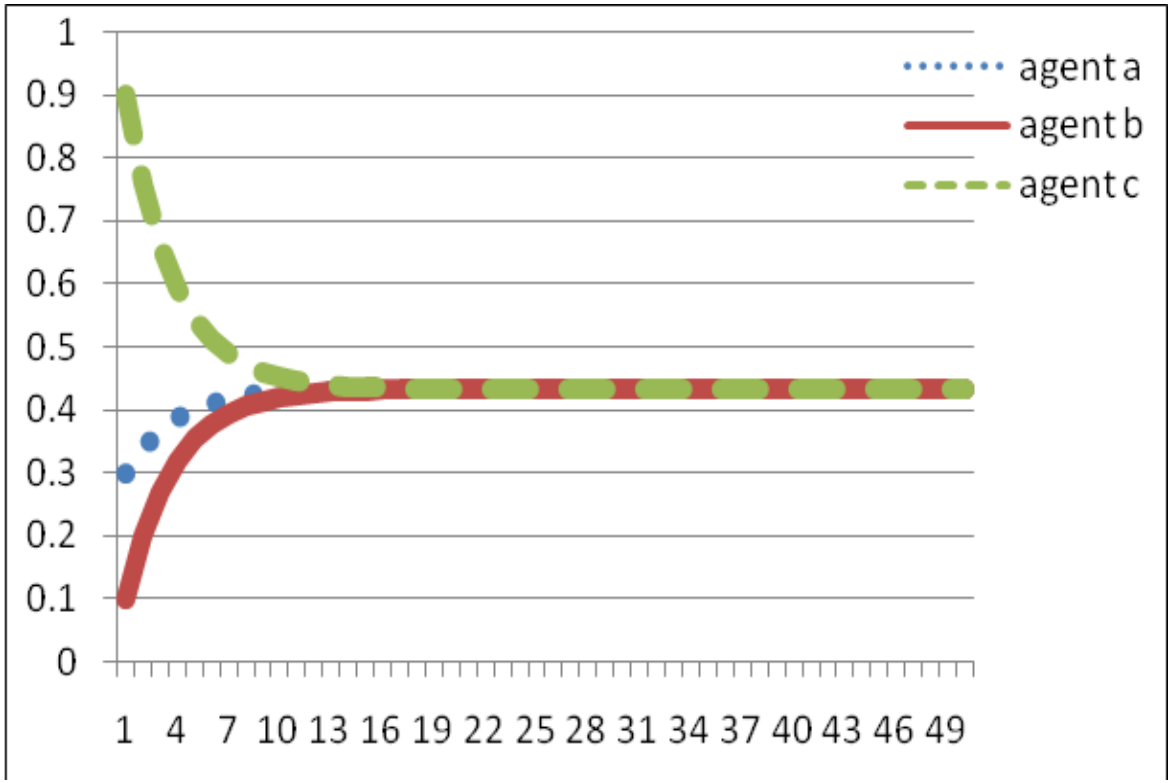
*(EmotionLevelB(t) - EmotionlevelA(t)) * ChannelStrengthAB(t) +*

*(EmotionLevelC(t) - EmotionlevelA(t)) * ChannelStrengthAC(t)*

Domain Model Simulation

Trace: General Pattern

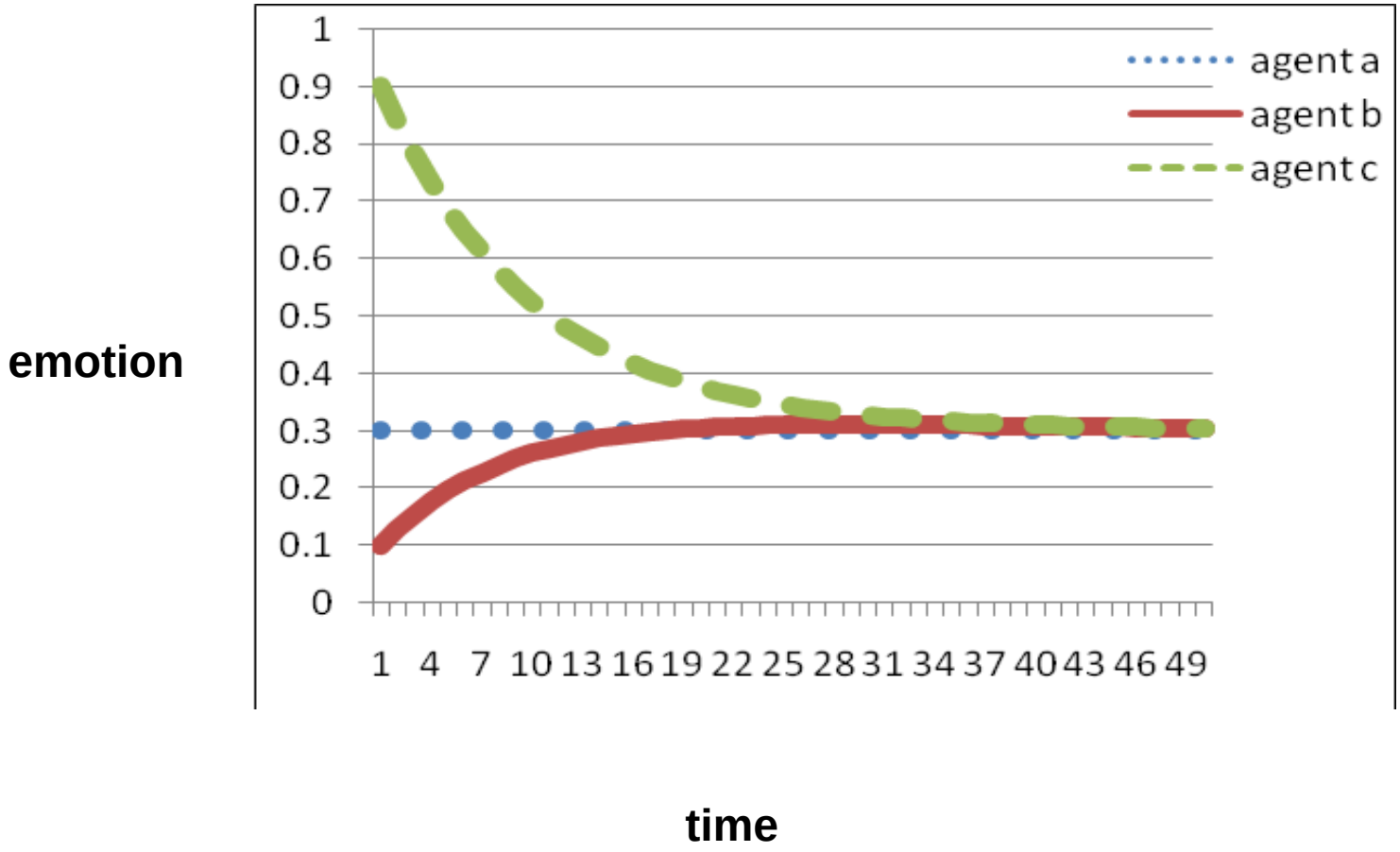
emotion



time

Domain Model Simulation

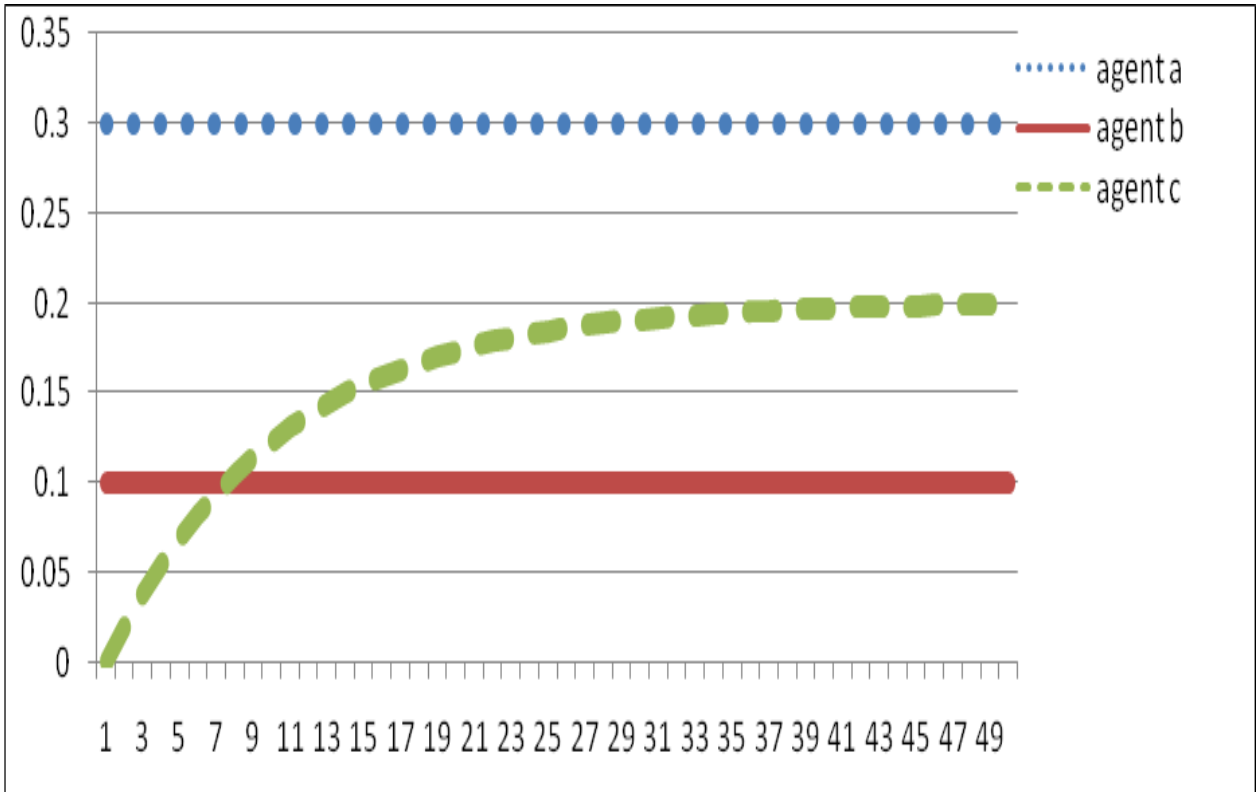
Trace: $\delta_a = 0$



Domain Model Simulation

Trace: $\delta_a = \delta_b = 0$

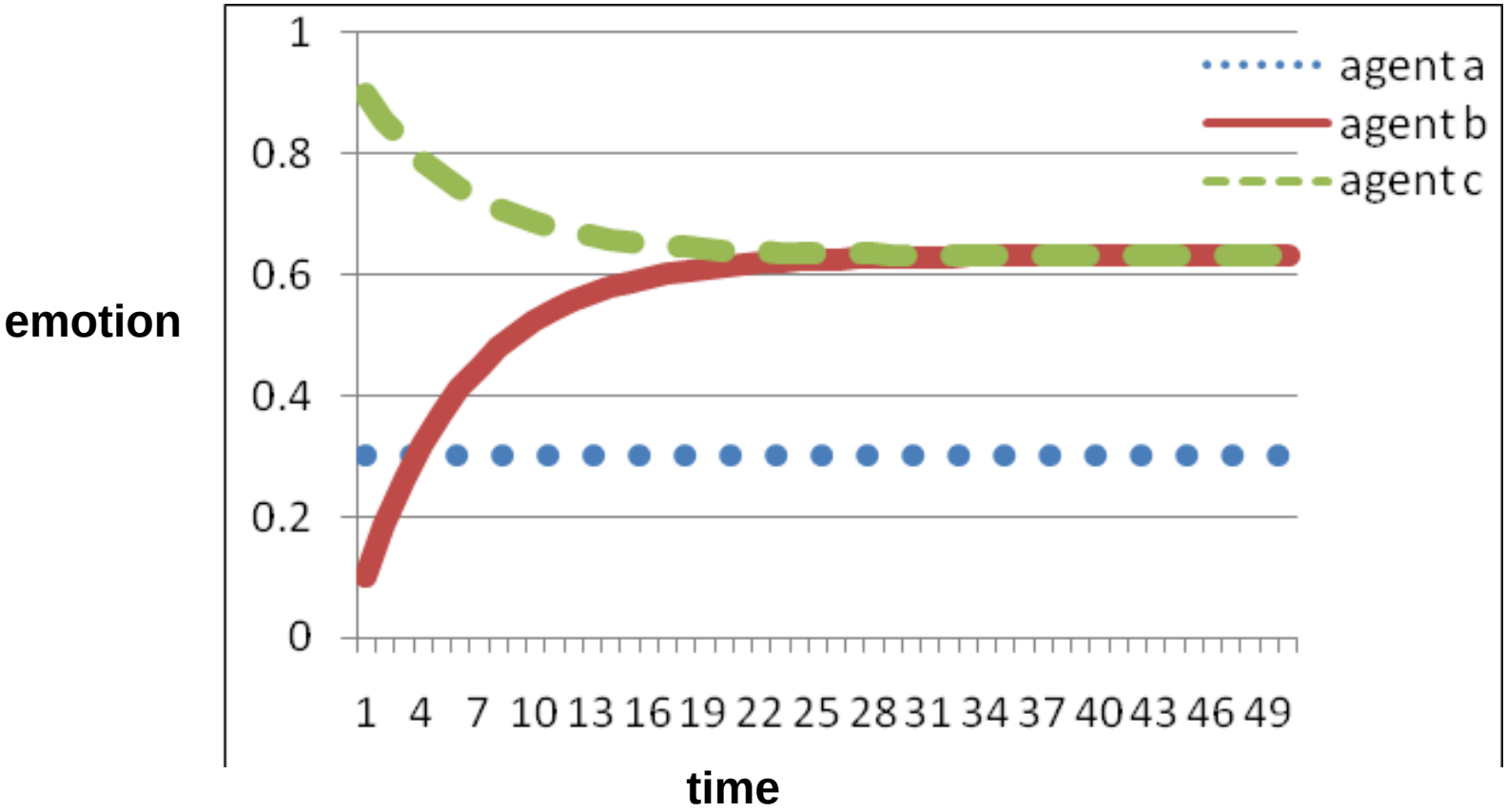
emotion



time

Domain Model Simulation

Trace: $\delta_a = \varepsilon_a = 0$



Conclusion

- Emotion contagion
  - Influencing others through induction of emotion states
  - Contagion strength: expressiveness, channel strength & openness
  - $qA(t+\Delta t) = qA(t) + \gamma A\,(qA^* - qA)\,\Delta t$
    for all group members A.

Summary modelling in Excel

- Core elements of a model:
  - Timeline
  - Concepts
    - with their possible values
  - Rules (relations)
    - for every time step, …
  - Simulation input
    - start values
    - external input
    - random variables
    - length of simulation

Timeline and Concepts

- **In Excel**: each row is a timestep and concepts are columns
  - no way to define what values concepts have (unspecified)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2
16	11	1	1	1	1	1	0	1	0	1	0	1		3,58	0,72	0,2
17	12	1	1	1	1	1	1	0	0	1	0	1		4,30	0,86	0,2
18	13	1	1	1	1	1	1	0	1	0	0	1		5,16	1,03	0,2
19	14	1	1	1	1	1	1	0	1	0	1	0		6,19	1,24	0,2
20	15	1	1	1	1	1	1	0	1	0	1	0		7,43	1,49	0,2
21	16	1	1	1	1	1	1	0	1	0	1	0		8,92	1,78	0,2
22	17	1	1	1	1	1	1	0	1	0	1	0		10,70	2,14	0,2
23	18	1	1	1	1	1	1	0	1	0	1	0		12,84	2,57	0,2

Rules

In Excel: formulae in cells

H11 =F10*(1-E10)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2

$$lacimpENZ(t+\Delta t) = lacIND(t)*(1 - glucIND(t))$$

Simulation input

- **In Excel**: first row: are start values; some columns: external input or random variables
  - length of simulation is number of rows

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2
16	11	1	1	1	1	1	0	1	0	1	0	1		3,58	0,72	0,2
17	12	1	1	1	1	1	1	0	0	1	0	1		4,30	0,86	0,2
18	13	1	1	1	1	1	1	0	1	0	0	1		5,16	1,03	0,2
19	14	1	1	1	1	1	1	0	1	0	1	0		6,19	1,24	0,2
20	15	1	1	1	1	1	1	0	1	0	1	0		7,43	1,49	0,2

Parameters

In Excel: yet another column

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2
16	11	1	1	1	1	1	0	1	0	1	0	1		3,58	0,72	0,2
17	12	1	1	1	1	1	1	0	0	1	0	1		4,30	0,86	0,2
18	13	1	1	1	1	1	1	0	1	0	0	1		5,16	1,03	0,2
19	14	1	1	1	1	1	1	0	1	0	1	0		6,19	1,24	0,2
20	15	1	1	1	1	1	1	0	1	0	1	0		7,43	1,49	0,2

What can be improved?

- Some aspects are not specified
  - i.e. values that concepts can have
- Different elements are represented in the same way
  - i.e. columns can be concepts, start-values, parameters, external factors
- Simulation input is scattered over different places
  - i.e. number of lines, columns with external inputs
- Not scalable with multiple agents

- Why problematic?
  - can cause errors; no automatic verification possible

What we need?

- New formalism with more explicit specification of model elements
  - Concepts and their possible values
  - Input for simulation: **scenario**
    - external input, length, start values
  - Parameters: variables
  - Rules: Python-code
  - Logical formalism that allows for parameterizing concepts
    - one rule for all agents, i.e. *contagionStrength(A,B,value)*

Modelling and Simulating in L2-Python

Dynamic Modelling for Human-centered Systems

Overview of Today's Lecture

- Modelling in L2-Python
  1. What is L2-Python?
  2. Programming overview
     - Concepts
     - Relations
     - Scenario's
     - Run and Visualization
- Debugging
- Assignment 3

Overview of Today's Lecture

- Modelling in L2-Python
  1. What is L2-Python?
  2. Programming overview
     - Concepts
     - Relations
     - Scenario's
     - Run and Visualization
- Debugging
- Assignment 3

What is L2-Python?

- Implementation of dynamic modelling formalism in Python
  - formalism originally named “LEADSTO” (Prolog)
  - then implementation in Matlab: L2-Matlab
  - new implementation is called L2-Python
- Hybrid formalism
  - logic formalism is basis
  - can cope with numerical values as well

What is L2-Python?

```python
def set_contagion_strength(states, t):
    """
        set the contagion strength between agents based on their expressiveness, and the other agent his/hers
        openness.
        First it is checked whether the required predicates are available. If not, this function will not
        be executed.
        Secondly, there is a for-loop to go over each has_expressiveness predicate. agent1 and his/hers
        expressiveness is retrieved from the predicate.
        Afterwards, there is a for-loop to go over each has_channel_strength, where agent1 is the first agent of
        the has_channel_strength predicate. agent2 is also retrieved from this predicate.
        Next, the openness of agent2 is retrieved from the has_openness predicate, and the contagion strength
        between the 2 agents is calculated.
        Finally, the new information for the predicate has_contagion_strength is appended to the
        new_predicates list, which is eventually added to the states predicates under the key has_contagion_strength

        has_expressiveness : [[agent], value]
        has_channel_strength : [[agent1, agent2], value]
        has_openness : [[agent], value]
        has_contagion_strength : [[agent1, agent2], value]
    """
    # obtain the name of the rule by using the inspect module
    rule_name = inspect.stack()[0][3]
    previous_state = states[t - 1]

    for has_expressiveness in previous_state.get_predicate("has_expressiveness"):
        # obtain the name of the agent and his/her expressiveness
        agent1 = has_expressiveness.get_name_agent(0)
        expressiveness = has_expressiveness.get_value()

        # again a for-loop, as agent1 can have multiple channel_strengths with different agents
        for has_channel_strength in previous_state.get_predicate_by_agent("has_channel_strength", agent1, 0):
            # obtain the name of the agent agent2 has a channel strength with, and his/her channel_strength
            agent2 = has_channel_strength.get_name_agent(1)
            channel_strength = has_channel_strength.get_value()
            # obtain the openness of agent2, by retrieving the has_openness predicate, using agent2 as the name of the
            # agent
            openness_agent2 = previous_state.get_predicate_by_agent("has_openness", agent2, 0)
            # as the get_predicate_by_agent() function returns a list (see the get_predicate_by_agent() function),
            # we have to take the first element (since every agent has only 1 openness)
            openness_agent2 = openness_agent2[0]
            openness = openness_agent2.get_value()
            # calculate the contagion_strength
            contagion_strength = round(expressiveness * channel_strength * openness, 3)
            # add the predicate to the next state
            new_predicate = Predicate("has_contagion_strength", [agent1, agent2], contagion_strength)
            states[t].add_predicate_to_state(new_predicate, rule_name)
```

Programming!

Why L2-Python?

Excel

VS

L2-Python

Why L2-Python?

- Why logical formalism?
  - Representation close to natural language
  - High abstraction level
    - define statements over variables
  - Automated tool support for logical analysis
- Why Python?
  - You are all familiar with it
  - Allows to easily embed models or use additional libraries

Core principle

- Concepts: predicates
  - predicates can have arguments and can be nested
    - has_connection_strength([arnie, bernie], 0.4)
- Relations: Python functions that define what values at next time step hold
- Simulation is list of **state objects**
  - numbered from 0 to last time step of simulation
  - every state (time point) consist of predicates
    - facts that hold at a specific time point and are used by the rules

Formalization of relations

- By means of **executable dynamic properties**
  - logical relations that specify truth value dependencies between **state properties** over time

Dynamic property

"LEADSTO" relation $\twoheadrightarrow$

If X holds at t

then Y will hold at t + Δt

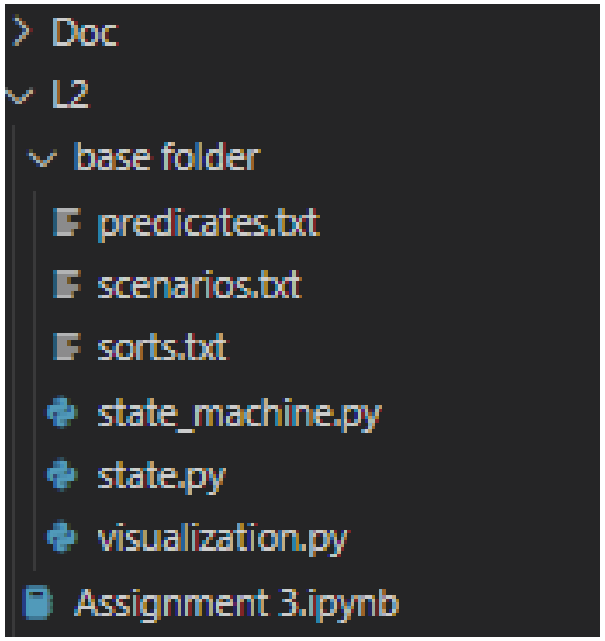
$X \twoheadrightarrow Y$

Overview of Today's Lecture

- Modelling in L2-Python
  1. What is L2-Python?
  2. Programming overview
     - Concepts
     - Relations
     - Scenario's
     - Run and Visualization
- Debugging
- Assignment 3

Programming

- All models consist of:



→ Documentation

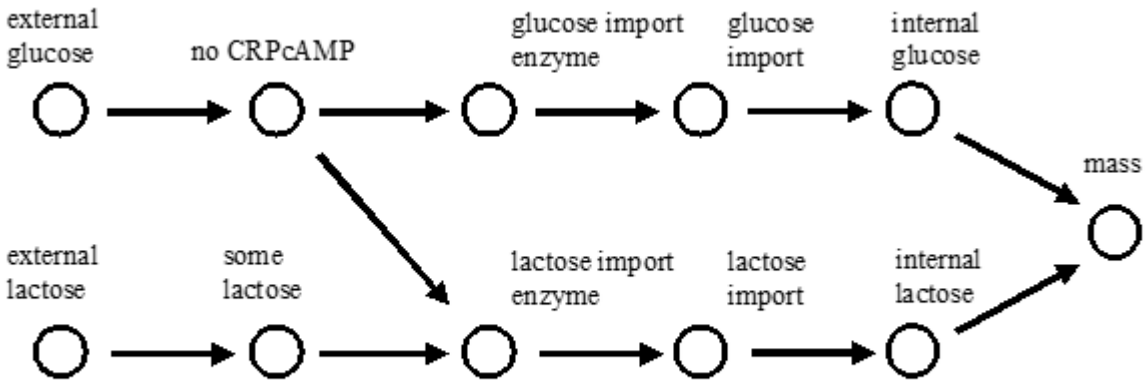
→ Concepts and simulation scenario

→ helper files

→ used to define which graphs are created

→ Relations + assignment

Predicates



- glucose externally present: *EXTgluc* [0/1]
- lactose externally present: *EXTlac* [0/1]
- glucose indicator (no CRPcAMP): *glucIND* [0/1]
- lactose indicator: *lacIND* [0/1]
- …

Predicates

In Excel: columns

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2
16	11	1	1	1	1	1	0	1	0	1	0	1		3,58	0,72	0,2
17	12	1	1	1	1	1	1	0	0	1	0	1		4,30	0,86	0,2
18	13	1	1	1	1	1	1	0	1	0	0	1		5,16	1,03	0,2
19	14	1	1	1	1	1	1	0	1	0	1	0		6,19	1,24	0,2
20	15	1	1	1	1	1	1	0	1	0	1	0		7,43	1,49	0,2
21	16	1	1	1	1	1	1	0	1	0	1	0		8,92	1,78	0,2
22	17	1	1	1	1	1	1	0	1	0	1	0		10,70	2,14	0,2
23	18	1	1	1	1	1	1	0	1	0	1	0		12,84	2,57	0,2

Predicates

In L2-Python: define predicates in the predicates.txt file

Are saved as dictonary with predicate name as key, (list of) agent and possible values per state (predicates.txt)

```
%%writefile predicates.txt
external_glucose;        {AGENT, BOOLEAN}
external_lactose;        {AGENT, BOOLEAN}
mass;                    {AGENT, REAL}
```

Sorts

In Excel: cells can contain any number

t	Δt	EG	EL	GI	LI	GIE	LIE	GIM	LIM	IG	IL	M
0	1	0	1	0	0	0	0	0	0	0	0	1,00
1	1	0	1	0	1	0	0	0	0	0	0	1,00
2	1	0	1	0	1	0	1	0	0	0	0	1,00
3	1	0	1	0	1	0	1	0	1	0	0	1,00
4	1	0	1	0	1	0	1	0	1	0	1	1,00

In L2-Python: sorts define which "type of values" are allowed at the positions of a predicate (sorts.txt)

Predefined: BOOLEAN & REAL

```
%%writefile predicates.txt
external_glucose;        {AGENT, BOOLEAN}
external_lactose;        {AGENT, BOOLEAN}
mass;                    {AGENT, REAL}
```

→

```
%%writefile sorts.txt
AGENT;      {arnie, bernie, charlie}
```

Scenarios

The set of (input) values for a particular simulation.

Scenarios

In Excel:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2
16	11	1	1	1	1	1	0	1	0	1	0	1		3,58	0,72	0,2
17	12	1	1	1	1	1	1	0	0	1	0	1		4,30	0,86	0,2
18	13	1	1	1	1	1	1	0	1	0	0	1		5,16	1,03	0,2
19	14	1	1	1	1	1	1	0	1	0	1	0		6,19	1,24	0,2
20	15	1	1	1	1	1	1	0	1	0	1	0		7,43	1,49	0,2

Scenarios

In L2-Python: scenarios.txt file

```
glucose_present{ecoli, 1}; [1:10]
glucose_present{ecoli, 1}; [10:20]
mass{ecoli, 1}; [1]
```

Programming

- Predicates, scenarios and sorts
  - If we want to create a rule that if an agent passes Dynamic Modelling, this agent is allowed to take Human-Computer Interaction
    - Predicate?
    - Sorts?
    - Scenario?

If passes(AGENT, DM)
$\twoheadrightarrow$ take_course(AGENT, HCI)

Dynamic property

If X holds at t

then Y will hold at t + Δt

"LEADSTO" relation $\twoheadrightarrow$

X $\twoheadrightarrow$ Y

Rules

Defines for each of the concepts what the truth values are at the next time step.

lacimpENZ(t+Δt) = lacIND(t)*(1 - glucIND(t))

Rules

In Excel:

H11 =F10*(1-E10)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2

$$lacimpENZ(t+\Delta t) = lacIND(t)*(1 - glucIND(t))$$

Rules

In L2-Python:

```python
def set_lactose_import_enzym(states, t):
    """
        set the lactose import enzym of an agent
    """
    # obtain the name of the rule by using the inspect module
    rule_name = inspect.stack()[0][3]
    # obtain the previous state
    previous_state = states[t - 1]

    for lactose_indicator_pred in previous_state.get_predicate("lactose_indicator"):
        # obtain the name of the agent and his/her lactose indicator
        agent = lactose_indicator_pred.get_name_agent(0)
        lc_ind_present = lactose_indicator_pred.get_value()

        # again a for-loop, use the name of the agent to get his/her glucose indicator
        for glucose_indicator_pred in previous_state.get_predicate_by_agent("glucose_indicator", agent, 0):
            # obtain glucose indicator
            gl_ind_present = glucose_indicator_pred.get_value()

            # perform the calculation
            lac_imp_enz = lc_ind_present * (1-gl_ind_present)

            # create the predicate and add it to the next state
            new_predicate = Predicate("lactose_import_enzym", [agent], lac_imp_enz)
            states[t].add_predicate_to_state(new_predicate, rule_name)
```

$$lacimpENZ(t+\Delta t) = lacIND(t)*(1 - glucIND(t))$$

Skeleton for rule file

```python
def sample_rule(states, t):
    # obtain the name of the rule by using the inspect module
    rule_name = inspect.stack()[0][3]
    # obtain the previous state
    previous_state = states[t - 1]

    for predicate in previous_state.get_predicate("name of the predicate"):
        # obtain the name of the agent
        agent = predicate.get_name_agent(0)
        # obtain the value of the predicate
        some_value = predicate.get_value()

        # create a new predicate
        new_predicate = Predicate("name of the new predicate", [agent], some_value)
        states[t].add_predicate_to_state(new_predicate, rule_name)
```

Rules

- Telling L2-Python which functions to execute as rules

```
################################
# any functions that actually need to be executed in the simulation must be put in this list
rules = [set_emotion_state, set_contagion_strength, set_emotion_level]
```

Parameters

Fixed variables (constants), which should be easily adaptable in the simulation.

Parameters

In Excel:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1			external	external	glucose	lactose	glucose	lactose	glucose	lactose	internal	internal			added	growth
2			glucose	lactose	indicator	indicator	enzyme	enzyme	import	import	glucose	lactose		mass	mass	efficiency
3																
4	*t*	*Δt*	*EG*	*EL*	*GI*	*LI*	*GIE*	*LIE*	*GIM*	*LIM*	*IG*	*IL*		*M*	*ΔM*	*γ*
5	0	1	0	1	0	0	0	0	0	0	0	0		1,00	0,00	0,2
6	1	1	0	1	0	1	0	0	0	0	0	0		1,00	0,00	0,2
7	2	1	0	1	0	1	0	1	0	0	0	0		1,00	0,00	0,2
8	3	1	0	1	0	1	0	1	0	1	0	0		1,00	0,00	0,2
9	4	1	0	1	0	1	0	1	0	1	0	1		1,00	0,20	0,2
10	5	1	0	1	0	1	0	1	0	1	0	1		1,20	0,24	0,2
11	6	1	0	1	0	1	0	1	0	1	0	1		1,44	0,29	0,2
12	7	1	0	1	0	1	0	1	0	1	0	1		1,73	0,35	0,2
13	8	1	0	1	0	1	0	1	0	1	0	1		2,07	0,41	0,2
14	9	1	0	1	0	1	0	1	0	1	0	1		2,49	0,50	0,2
15	10	1	1	1	0	1	0	1	0	1	0	1		2,99	0,60	0,2
16	11	1	1	1	1	1	0	1	0	1	0	1		3,58	0,72	0,2
17	12	1	1	1	1	1	1	0	0	1	0	1		4,30	0,86	0,2
18	13	1	1	1	1	1	1	0	1	0	0	1		5,16	1,03	0,2
19	14	1	1	1	1	1	1	0	1	0	1	0		6,19	1,24	0,2
20	15	1	1	1	1	1	1	0	1	0	1	0		7,43	1,49	0,2

Parameters

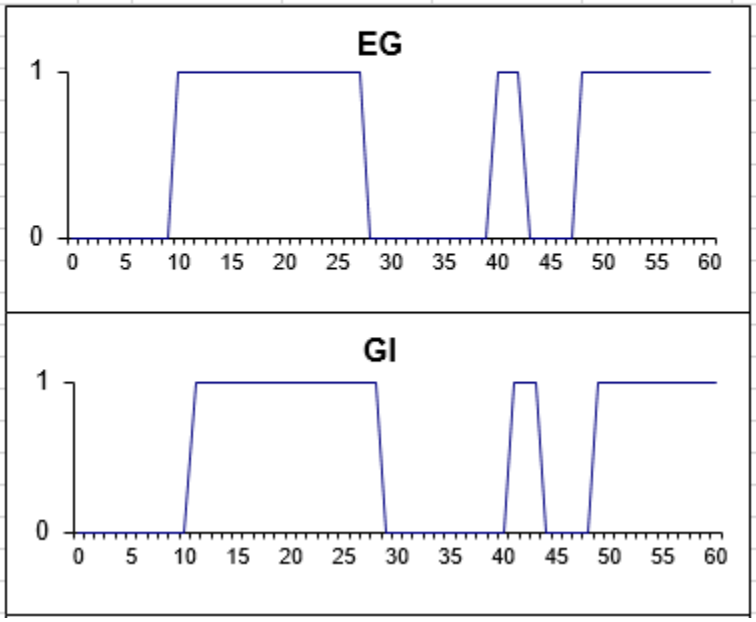
In L2-Python: make a separate code block in the Jupyter Notebook

```
##########################################
# any variables declared here will be treated as scenario parameters
growth_efficiency = 0.2
```

How to run?

In Excel: automatically

J	K	L	M	N	O	P
lactose import	internal glucose	internal lactose		mass	added mass	growth efficiency
LIM	*IG*	*IL*		*M*	*ΔM*	*γ*
0	0	0		1,00	0,00	0,2
0	0	0		1,00	0,00	0,2
0	0	0		1,00	0,00	0,2
1	0	0		1,00	0,00	0,2
1	0	1		1,00	0,20	0,2
1	0	1		1,20	0,24	0,2
1	0	1		1,44	0,29	0,2
1	0	1		1,73	0,35	0,2
1	0	1		2,07	0,41	0,2
1	0	1		2,49	0,50	0,2
1	0	1		2,99	0,60	0,2
1	0	1		3,58	0,72	0,2
1	0	1		4,30	0,86	0,2
0	0	1		5,16	1,03	0,2
0	1	0		6,19	1,24	0,2
0	1	0		7,43	1,49	0,2
0	1	0		8,92	1,78	0,2
0	1	0		10,70	2,14	0,2
0	1	0		12,84	2,57	0,2
0	1	0		15,41	3,08	0,2



In L2-Python

1. Initialize rules

```
rules = [set_contagion_strength, set_emotion_level]
```

2. Import the StateMachine and run_visualization

```
from state_machine import StateMachine
from visualization import run_visualization
```

3. Define the duration and name of the .txt files

```
MAX_T = 50
scenario = "scenarios.txt"
predicates = "predicates.txt"
sorts = "sorts.txt"
```

4. Initialize the StateMachine

```
StateMachine = StateMachine(predicates, sorts, scenario, MAX_T, rules)
```

5. Run!

```
StateMachine.run()
```

Assuming the

- predicates.txt
- sorts.txt
- scenarios.txt
- rules

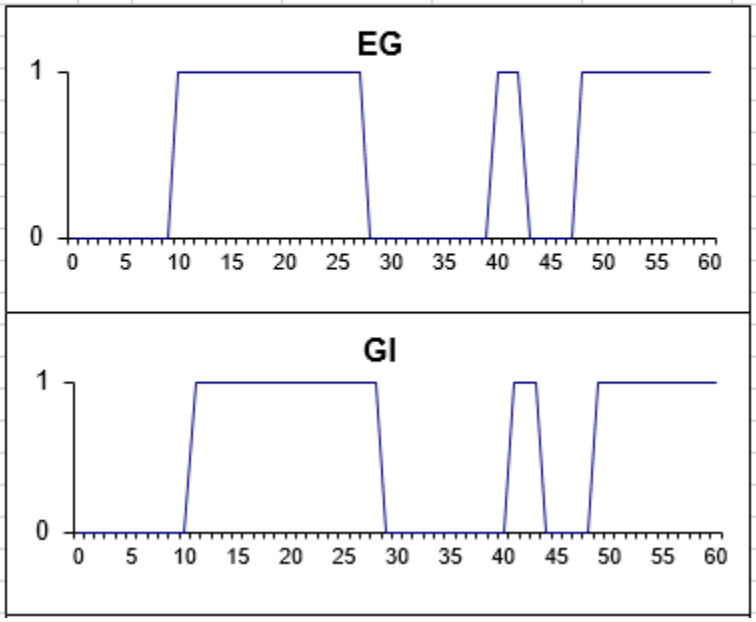
have already been defined correctly!

Please note that if you changed a rule AFTER initializing the StateMachine, you have to re-initialize the StateMachine

How to visualize?

In Excel: (almost) automatically

J	K	L	M	N	O	P
lactose import	internal glucose	internal lactose		mass	added mass	growth efficiency
LIM	*IG*	*IL*		*M*	*ΔM*	*γ*
0	0	0		1,00	0,00	0,2
0	0	0		1,00	0,00	0,2
0	0	0		1,00	0,00	0,2
1	0	0		1,00	0,00	0,2
1	0	1		1,00	0,20	0,2
1	0	1		1,20	0,24	0,2
1	0	1		1,44	0,29	0,2
1	0	1		1,73	0,35	0,2
1	0	1		2,07	0,41	0,2
1	0	1		2,49	0,50	0,2
1	0	1		2,99	0,60	0,2
1	0	1		3,58	0,72	0,2
1	0	1		4,30	0,86	0,2
0	0	1		5,16	1,03	0,2
0	1	0		6,19	1,24	0,2
0	1	0		7,43	1,49	0,2
0	1	0		8,92	1,78	0,2
0	1	0		10,70	2,14	0,2
0	1	0		12,84	2,57	0,2
0	1	0		15,41	3,08	0,2



In L2-Python

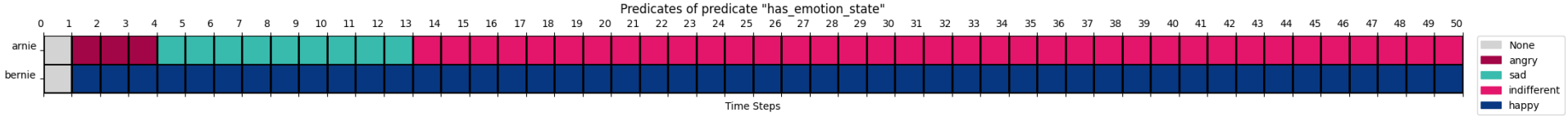
1. Execute the “run_visualization” function, which we imported
    1. The first argument is the StateMachine object
    2. The second argument is a **list** of predicates you do **<u>not</u>** want to visualize

```
run_visualization(StateMachine, ["has_expressiveness"])
```

```
The following predicates will be visualized:
has_openness
has_channel_strength
has_emotion_level
has_contagion_strength
```

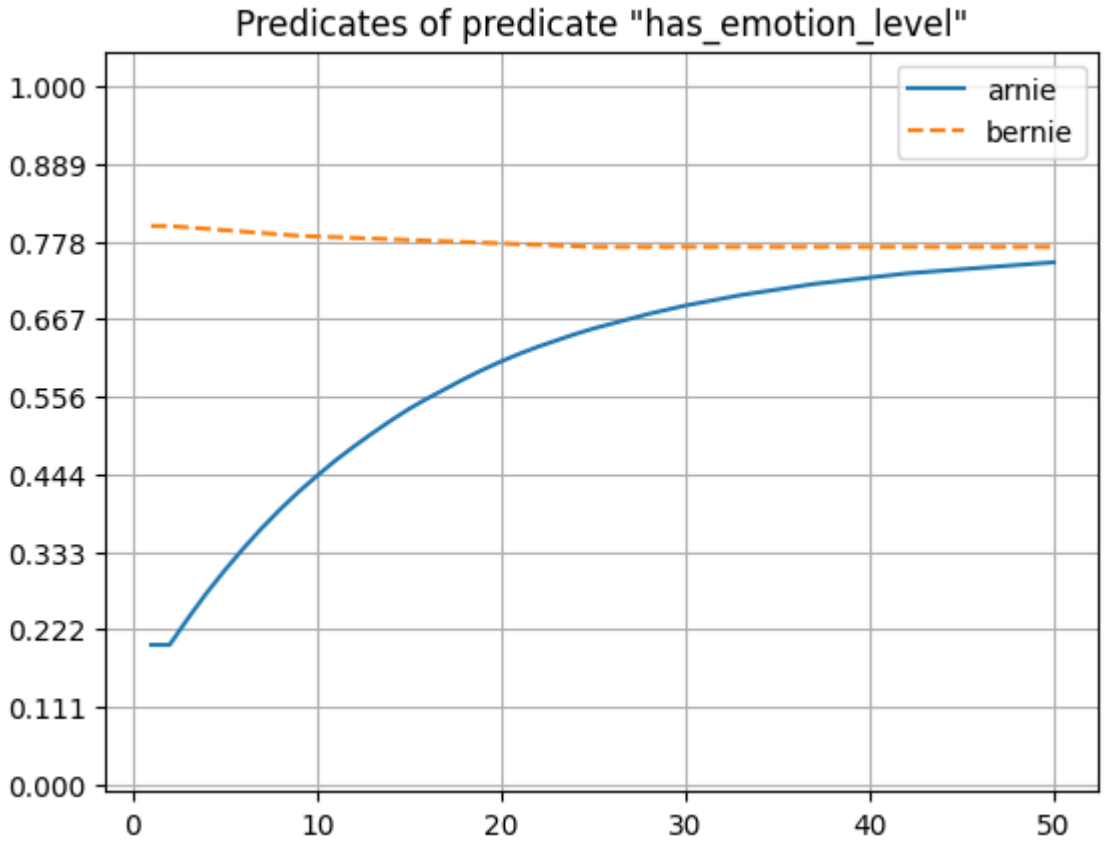
Visualization

In L2-Python (for truth values):



Visualization

In L2-Python (for numerical values):



Overview of Today's Lecture

- Modelling in L2-Python
  1. What is L2-Python?
  2. Programming overview
     - Concepts
     - Relations
     - Scenario's
     - Run and Visualization
- Debugging
- Assignment 3

In L2-Python

In order to debug: pass the argument **`debug_mode=True`** when running the StateMachine

```python
from state_machine import StateMachine
StateMachine = StateMachine(predicates, sorts, scenario, MAX_T, rules)
# Run debug_mode
StateMachine.run(debug_mode=True)
```

```
##################Debug mode activated##################
The StateMachine contains the following defined sorts, besides BOOLEAN and REAL:
    AGENT, EMOTION_STATE

The StateMachine contains the predicates with their corresponding sorts:
    Predicate name: has_expressiveness, predicate sorts (index dependent): ['AGENT', 'REAL']
    Predicate name: has_openness, predicate sorts (index dependent): ['AGENT', 'REAL']
    Predicate name: has_channel_strength, predicate sorts (index dependent): ['AGENT', 'AGENT', 'REAL']
    Predicate name: has_contagion_strength, predicate sorts (index dependent): ['AGENT', 'AGENT', 'REAL']
    Predicate name: has_emotion_level, predicate sorts (index dependent): ['AGENT', 'REAL']
    Predicate name: has_emotion_state, predicate sorts (index dependent): ['AGENT', 'EMOTION_STATE']
##################Debug report per state##################

This state, state 1, contains the following predicates:
    The predicate has_expressiveness, with agents: arnie, and value 0.2
    The predicate has_expressiveness, with agents: bernie, and value 0.8
    The predicate has_openness, with agents: arnie, and value 0.8
    The predicate has_openness, with agents: bernie, and value 0.2
    The predicate has_channel_strength, with agents: arnie, bernie, and value 1.0
    The predicate has_channel_strength, with agents: bernie, arnie, and value 1.0
    The predicate has_emotion_level with agents: arnie, and value 0.2
    The predicate has_emotion_level with agents: bernie, and value 0.8
```

Note: if predicates are undefined in the StateMachine (not in the predicates.txt), the StateMachine will also give you a warning when running it.

Overview of Today's Lecture

- Modelling in L2-Python
  1. What is L2-Python?
  2. Programming overview
     - Concepts
     - Relations
     - Scenario's
     - Visualization
- Debugging
- Assignment 3

Assignment 3

- L2-Python tutorial – emotion contagion, but now in hybrid formalism
- Individual
- Pass / Fail
- Checked by TA

START THIS WEEK WITH INSTALLING AND READING

All of the models we will work with will be created in L2-Python. Make sure **YOU** understand how it works, not just your partner.

Formalisation of concepts

- Predicates

Concept	Formalisation
person B has emotion expressiveness E	`has_expressiveness(B:AGENT, E:REAL)`
person A has openness for received emotion D	`has_openness(A:AGENT, D:REAL)`
channel from sender B to receiver A has strength C	`has_channel_strength(` `B:AGENT, A:AGENT, C:REAL)`
contagion strength from B to A is CS	`has_contagion_strength(` `B:AGENT, A:AGENT, CS:REAL)`
overall contagion strength to receiver A is CS	`has_overall_contagion_strength(` `A:AGENT, CS:REAL)`
person A has relevance factor R	`has_relevance(A:AGENT, R:REAL)`
person A has emotion level V	`has_emotion_level(A:AGENT, V:REAL)`
the group has emotion level V	`group_emotion_level(V:REAL)`

Formalisation of concepts

- Sorts

Sort	Description of use	Elements
AGENT	group members	`a1, a2, a3, a, b, c, arnie, bernie, charlie`
REAL	group member emotion level, group emotion level, expressiveness factor, channel strength, openness, contagion strength, relevance factor	`real numbers`

Assignment 3 – Tips

- Really do understand what you are doing
- Warnings are there to **warn** you. Warnings about predicates not being available at the beginning of the simulation **can** be expected
- **Do** use the debug mode when debugging
- **Do not** touch the base folder
- **Check** rule by rule. For example, if you need to implement 3 rules: implement rule 1 and check whether it is working by visualizing/debug

Questions?