

Example 1: Leadsto tutorial

Consider an experimental setting, involving two positions (say, p1 and p2), an animal, a piece of food, and a transparent screen (e.g., a window). Suppose the animal is placed at position p1, the food is placed at p2, and the screen is placed in between, separating the animal from the food. Multiple trials are performed in which, at some variable moment, the screen is raised, and the animal is free to go to any position. After a number of trials, it turns out that a regularity can be observed in the behaviour of the animal. This regularity can be expressed informally by the following property:

*Every time that the agent observes that there is food at p2,
and no screen is present, it will go to p2.*

In semi-formal form, this property can be written as follows:

LP1 (Local Property 1)

at any point in time,
if the agent observes that food is present at position p2
and it observes that no screen is present,
then it will go to position p2

To implement this model, we create the following files.

sorts.l2

The only sort we will use is BOOLEAN, which is defined by default. Therefore, there is no need for a sorts.l2 file.

predicates.l2

```
goes_to_p2;           BOOLEAN
observes_food_at_p2;  BOOLEAN
observes_no_screen;   BOOLEAN
```

These are the three predicates used in LP1, each of them using a boolean value.

scenarios.l2

```
default (
  observes_no_screen{true};    3
  observes_food_at_p2{true};   [1:3]
)

no_food (
  observes_no_screen{true};    3
)
```

Here we define our scenarios, in this case two. In the first scenario the animal observes food at time points 1, 2 and 3 and no screen at time point 3. In the second scenario, the animal observes no screen on time point 3, but does not observe any food.

parameters.l2

This example does not make use of any parameters.

rules.m

```
function [ fncs ] = rules()
    %DO NOT EDIT
    fncs = l2.getRules();
    for i=1:length(fncs)
        fncs{i} = str2func(fncs{i});
    end
end

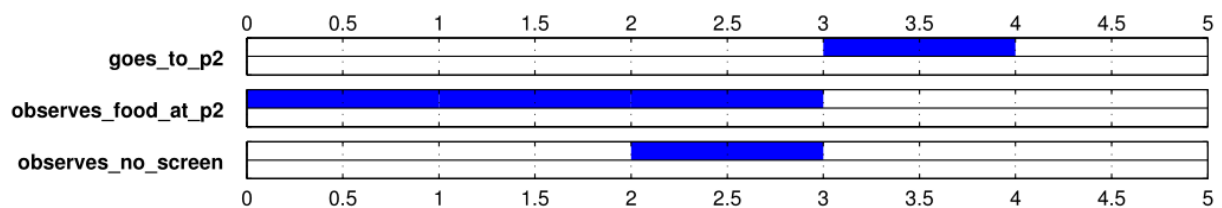
%ADD RULES BELOW
function result = ddr1( trace, parameters, t )
    move = trace(t).observes_food_at_p2 & trace(t).observes_no_screen;
    result = {t+1, predicate('goes_to_p2', move)};
end
```

Our rules.m file only contains one rule, namely lp1. Here, we check whether both observes_food_at_p2 and observes_no_screen are true at time point t in the current trace. The comparison tells us whether the animal goes to p2 at the next time step, thus this is our result for goes_to_p2 at time point t+1.

Running the example in Matlab

```
clear all
close all
model = l2('01_leadsto_tutorial')
model.simulate(5)
model.plot()
```

This code clears all variables and closes all windows, after which it loads, runs and plots the model. The resulting plot looks as follows.



Example 2: Emotion regulation

Emotion regulation refers to ‘all of the conscious and nonconscious strategies we use to increase, maintain, or decrease one or more components of an emotional response’ (Gross, 2001). This ability to regulate our own emotional state is one of the properties that distinguish humans from other higher animals, providing us a high behavioral flexibility, which may result in a higher sense of well-being and mental health. A specific strategy for emotion regulation that is widely studied is reappraisal. Gross (2001) defines reappraisal as a process where ‘the individual reappraises or cognitively re-evaluates a potentially emotion-eliciting situation in terms that decrease its emotional impact’.

A simple simulation model of reappraisal, which is based on the model presented in (Bosse, Pontier, and Treur, 2010) consists of the following two difference equations:

- 1)
$$\text{new_ERL} = \text{ERL} + (1-\beta) * (((1-w) * v_1 + w * v_2) - \text{ERL}) \Delta t$$
- 2)
$$\text{new_}v_2 = v_2 - (\alpha_2 * d / d_{\max}) \Delta t$$
 (where $d = \text{ERL} - \text{ERL}_{\text{norm}}$)

To implement this model, we create the following files.

sorts.l2

The only sort we will use is REAL, which is defined by default. Therefore, there is no need for a sorts.l2 file.

predicates.l2

```
erl;      REAL
d;        REAL
v2;       REAL
```

These are the three values calculated in the description above. Note, we could also calculate d ‘on the fly’ in the rule for v_2 , thereby eliminating the need for this predicate.

scenarios.l2

```
erl{0};   1
d{0};     1
v2{0};    1
```

We only define one scenario, with all the values at time point 1 being 0.

parameters.l2

```
default(
    beta;          0
    w;             0
    v1;            0
    a2;            0
    delta_t;       0
    d_max;         1
    erl_norm;      0
)

reappraisal(
    beta;          0.5257
    w;             0.5773
    v1;            0.4170
    a2;            0.0426
    delta_t;       0.0439
    d_max;         1
    erl_norm;      0
)
```

This example uses multiple parameters. We define two sets, one default set containing meaningless values and one set tuned to resemble a reappraisal pattern.

rules.m

```
function [ fncs ] = rules()...

%ADD RULES BELOW
function result = ddr1( trace, params, t )
    erl_new = trace(t).erl + (1-params.beta) * (( (1-params.w) * ...
        params.v1 + params.w * trace(t).v2 ) - trace(t).erl) * ...
        params.delta_t;
    result = {t+1, 'erl', erl_new};
end

function result = ddr2( trace, params, t )
    v2_new = trace(t).v2 - ( params.a2 * trace(t).d / params.d_max ) * ...
        params.delta_t;
    result = {t+1, 'v2', v2_new};
end

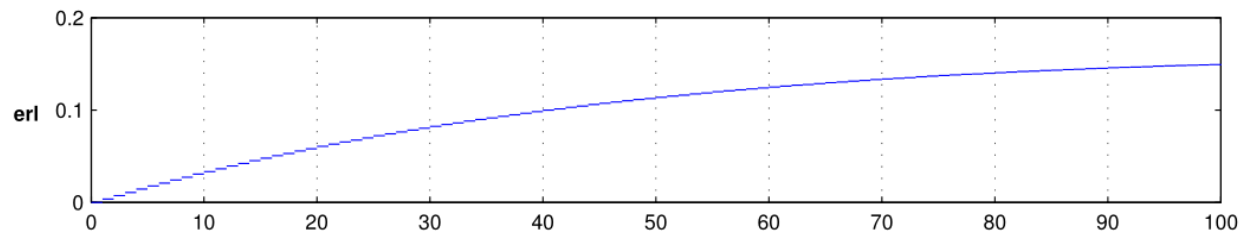
function result = ddr3( trace, params, t )
    d_new = trace(t).erl - params.erl_norm;
    result = {t+1, 'd', d_new};
end
```

Our rules.m file only contains the two formulas given in the text, as well as a separate function for d.

Running the example in Matlab

```
clear all
close all
model = l2('02_emotion_regulation')
model.simulate(100, 'default', 'reappraisal')
model.plot({'erl'})
```

This code clears all variables and closes all windows, after which it loads, runs and plots the model. For this simulation run, the default (and only) scenario is used, but the parameter set 'reappraisal' is used instead of the set of default values. For the plot, only 'erl' is of our interest.



Example 3: IAPS pictures

'The International Affective Picture System (IAPS) is [...] a set of normative emotional stimuli for experimental investigations of emotion and attention.' (Lang et al., 1999) Each of these pictures has a rating for both valence and arousal. Consider an agent viewing a number of these pictures. The valence and arousal experienced by that agent will adapt to those of the pictures viewed. The speed of this process depends on the speed parameters w_1 and w_2 respectively.

DDR1 (Domain Dynamic Relation 1)

```
at any point in time, for each agent X
if   agent X has a valence V and arousal A
and  an agent X observes a picture P from the IAPS picture set
and  picture P has a valence rating V1 and arousal rating A1
then agent X will get a valence of  $V+w_1(V1-V)$  and arousal of  $A+w_2(A1-A)$ 
```

For the implementation of this model, the following files are created.

sorts.l2

```
AGENT;      {arnie, bernie, charlie}
PICTURE;    {p1, p2, p3}
```

We consider here both agent and pictures as sort, with 3 agents and 4 pictures possible.

predicates.l2

```
% picture(PICTURE, VALENCE, AROUSAL)
picture;    {PICTURE, REAL, REAL}

% emotion(AGENT, VALENCE, AROUSAL)
emotion;    {AGENT, REAL, REAL}

observes:   {AGENT. PICTURE}
```

For this model, three predicates are necessary. The first, picture, describes the valence and arousal of each picture. The second, emotion, describes the current valence and arousal value of an agent. The last predicate, observes, is used to denote if an agent views a particular picture.

parameters.l2

```
w1;         0.1
w2;         0.1
```

To keep it simple, we only consider one possible set for the parameters given in the model description.

scenarios.l2

```
% IAPS picture valence/arousal
picture{p1, 1, 9}; [1:Inf]
picture{p2, 9, 9}; [1:Inf]

% Starting emotion of the agents
emotion{arnie, 4.5, 4.5}; 1
emotion{bernie, 4.5, 4.5}; 1

% Observing of pictures
observes{arnie, p1}; [1:25]
observes{bernie, p1}; [1:25]
observes{arnie, p2}; [26:50]
```

One default scenario is defined, using two pictures and two agents. The first lines describe the static valence and arousal values for the two pictures; one very arousing sad picture (p1) and another very arousing happy picture (p2). The next lines describe the starting emotion values of arnie and bernie. The last lines describe that both agents view the sad picture after which only arnie looks at the happy picture.

rules.m

```
%ADD RULES BELOW
function result = ddrl( trace, params, t )

    result = {};

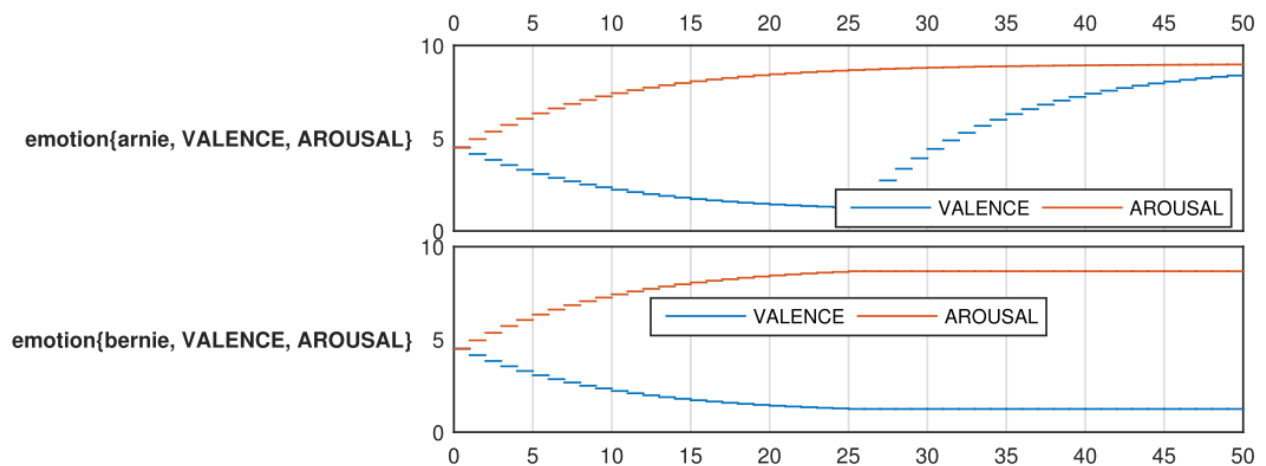
    %go through each agent's emotion
    for emotion = trace(t).emotion
        x = emotion.arg{1}; %agent name
        v = emotion.arg{2}; %valence
        a = emotion.arg{3}; %arousal
        % go through each picture that agent observes
        for observes = l2.getall(trace, t, 'observes', {x, NaN})
            p = observes.arg{2}; %picture name
            %get the valence and arousal of that picture
            [~, picture] = l2.exists(trace, t, 'picture', {p, NaN, NaN});
            v1 = picture.arg{2}; %picture valence
            a1 = picture.arg{3}; %picture arousal
            %adjust v and a accordingly
            v = v + params.w1 * (v1 - v);
            a = a + params.w2 * (a1 - a);
        end
        %add the new emotion level to the results
        result = { result{:} {t+1, 'emotion', {x, v, a}} };
    end
end
```

One, relatively large rule is needed to implement this model. The majority of the rule is part of a for loop going through the emotion of each agent. For that agent, each picture it observes is retrieved. For those pictures, the valence and arousal value is looked at and using those values the emotion level of the agent is changed. When each picture an agent looked at is processed, the new emotion level is added to the results.

Running the example in Matlab

```
clear all
close all
model = l2('03_IAPS_pictures')
model.simulate(50);
%Plot is normally too large for screen,
%but a custom plot_emotion.m has been written.
model.plot({'emotion'});
```

The plot is shown below, but as you might have noticed uses a custom plot_emotion.m that is residing in the model folder. More on how to write custom plot functions can be found in the manual.



Example 4: Theory of mind

One of the most important milestones in theory of mind development is gaining the ability to attribute *false belief*: that is, to recognize that others can have beliefs about the world that are diverging. In the ‘appearance-reality’, or ‘Smarties’ task, experimenters ask children what they believe to be the contents of a box that looks as though it holds a candy called smarties. After the child guesses (usually) smarties, it is shown that the box in fact contained pencils. The experimenter then re-closes the box and asks the child what she thinks another person, who has not been shown the true contents of the box, will think is inside. The child passes the task if he/she responds that another person will think that there are smarties in the box, but fails the task if she responds that another person will think that the box contains pencils. Gopnik & Astington (1988) found that children pass this test at age four or five years.

This experiment has been implemented as follows.

sorts.l2

```
AGENT;      {arnie, bernie, charlie}
INFO;       {smarties, pencils, <belief>}
```

As can be seen, we have three different agents, and some INFO element which holds information about the contents of the box (smarties or pencils). However, it is also stated that the INFO element can be <belief>, meaning that instead of either smarties or pencils a predicate ‘belief’ is also a valid INFO element. This predicate, as well as two others are defined below.

predicates.l2

```
age;        {AGENT, REAL}
belief;     {AGENT, INFO}
observe;    {AGENT, INFO}
```

Three predicates are used, one describing the age of the agent, one for the beliefs of the agent and a third predicate for the observations of an agent. Keep in mind, that the belief can be either smarties, pencils or a belief about some other agent having a particular belief.¹

parameters.l2

```
age_tom;    4
```

¹ Using this formalization, an agent can also have an observation about some particular belief. This however is not used or required for modeling this experiment.

One parameter is used for this model, setting the age from which an agent can apply theory of mind, thereby understanding that a different person, not having seen the contents of the box, will falsely believe that there are smarties instead of pencils in the box.

scenarios.l2

```
default (  
    age{arnie, 3};                                [1:5]  
    belief{arnie, smarties};                        [1]  
    belief{arnie, belief{bernie, smarties}};        [1]  
    observe{arnie, pencils};                        [2]  
)  
  
older (  
    age{arnie, 5};                                [1:5];  
    belief{arnie, smarties};                        [1];  
    belief{arnie, belief{bernie, smarties}};        [1];  
    observe{arnie, pencils};                        [2];  
)
```

We define two scenarios, one where Arnie is too young to apply theory of mind, and another where he is old enough. Furthermore, we initially set both the belief of arnie that there are smarties in the box and his belief that he believes that bernie thinks there are smarties in the box. Lastly, at timepoint 2 arnie observes there are pencils in the box.

On the next page, the rule to implement this model is shown. It cycles through all beliefs and for each beliefs considers all observations made by the agent that has that belief. For each observation, we first check whether the belief was a belief of the agent itself or a nested belief about some other agent's belief. This is done by checking if the observation is a structure, which it would be in the case of a nested belief. If it isn't a structure, we update the belief with the current observation. Otherwise, we need to check whether or not the agent is able to apply theory of mind. If not, the nested belief also gets updated to match the current observation.

Although this method works, it relies on the assumption that any nested belief does not contain another nested belief. For example, consider the following belief where arnie believes that bernie believes that charlie believes the box contains smarties.

```
belief{arnie, belief{bernie, belief{charlie, smarties}}}
```

In order to cope with any number of nested beliefs, the rule needs to be implemented differently using a form of recursive programming. This method will be explained next and is included in the example code as comments. For the given scenarios, results for both methods are similar.

rules.m

```
function result = ddr1( trace, params, t )
    result = {};
    %go through each belief
    for belief = trace(t).belief
        agent = belief.arg{1}; %agent name
        belief = belief.arg{2}; %the belief or a nested belief
        % go through each observation of that agent
        for observe = l2.getall(trace, t, 'observe', {agent, NaN})
            observation = observe.arg{2}; %the info / nested observation

            %belief is own belief, not a nested belief
            if ~ispredicate(belief)
                belief = observation;

            %belief is a (nested) belief of someone else
            else
                %get age to check old enough to apply theory of mind
                [~, age] = l2.exists(trace, t, 'age', {agent, NaN});
                tom = age.arg{2} > params.age_tom;

                %if no theory of mind, simply update nested belief
                if ~tom
                    belief = predicate('belief', {belief.arg{1} observation} );
                end
            end
        end
        result = { result{:} {t+1, 'belief', {agent, belief}} };
    end
end
```

On the following page, the rule to implement this rule recursively is shown. The first part of this rule is similar as different from before. It cycles through all the beliefs and updates them for the relevant observations. However, beliefs are now updated by a nested function `updateBelief`, such that it is possible to cope with any amount of nested beliefs.

Consider the function `updateBelief` and firstly notice that it is part of the function `ddr1`. It takes as input the current belief, the observation and whether or not theory of mind is applicable. If the belief is not a predicate (`~ispredicate(belief)`), the belief becomes equal to the observation. Otherwise, if the belief is a nested belief, that belief is only updated if no theory of mind is applied. The new belief is a new structure, named belief (the predicate name). The value for the agent is equal to the agent name in the previous belief. To update the belief in this nested belief, the `updateBelief` function is called again with the belief value from the nested belief, the current observation and whether or not theory of mind is applied. Using this recursive structure, it does not matter how many nested beliefs there are, they always get updated as expected.

rules.m

```
%ADD RULES BELOW
function result = ddr1( trace, params, t )
    result = {};
    %go through each belief
    for belief = trace(t).belief
        agent = belief.arg{1}; %agent name
        belief = belief.arg{2}; %the belief or a nested belief
        % go through each picture that agent observes
        for observe = l2.getall(trace, t, 'observe', {agent, NaN})
            observation = observe.arg{2}; %the info or a nested observation
            %get age to check if name is old enough to apply theory of mind
            [~, age] = l2.exists(trace, t, 'age', {agent, NaN});
            tom = age.arg{2} > params.age_tom;
            belief = updateBelief(belief, observation, tom);
        end
        result = { result{:} {t+1, 'belief', {agent, belief}} };
    end

    function belief = updateBelief(belief, observation, tom)
        %belief is a info element, not a nested belief
        if ~ispredicate(belief)
            belief = observation;

            %belief is a (nested) belief of someone else
        else
            %if no theory of mind, simply update nested belief
            if ~tom
                belief = predicate('belief', {belief.arg{1} ...
                    updateBelief(belief.arg{2}, observation, tom)} );
            end
            %otherwise, do nothing with the belief
        end
    end
end
end
```

Running the example in Matlab

```
clear all
close all
model = l2('04_theory_of_mind')
% model = l2('04_theory_of_mind', 'rules', 'recursive.m')
model.simulate(3);
model.plot();
set(gcf, 'name', 'Default scenario', 'numbertitle', 'off')
model.simulate(3, 'older');
model.plot();
set(gcf, 'name', 'Older scenario', 'numbertitle', 'off')
```

To run this example, the model is simulated for both scenarios. The resulting graphs are shown below as well. Note the difference between the age of arnie and whether or not the nested belief of bernie gets updated based on the observation arnie makes at timepoint 2.

