TUTORIAL
Systems of fuzzy IF-THEN rules in NetLogo using the fuzzy logic library

1 Background and overall description of the model

This tutorial demonstrates how to create a system of fuzzy IF-THEN rules in NetLogo using the Fuzzy Logic library **fuzzy-logic.nls**. Systems of fuzzy IF-THEN rules are explained in section 2.6 of the paper. Here we use an example inspired by two of the researchers' visit to Segovia (Spain) during July 2014. The researchers engaged in a guided walking tour, visiting the most iconic places of the city and, at the end of the tour, participants received a satisfaction survey. In this example, for simplicity of presentation, we use only two factors that can contribute to a holiday experience: the weather conditions, expressed by ambient temperature, and the cost of the tour[1]. The following text provides an overview of the model.

The model runs in discrete time-steps. Figure 1 provides an informal sketch that illustrates the sequence of events within each time-step. Note that the sequence starts with the occurrence of a tour; in every time-step one (and only one) tour takes place.

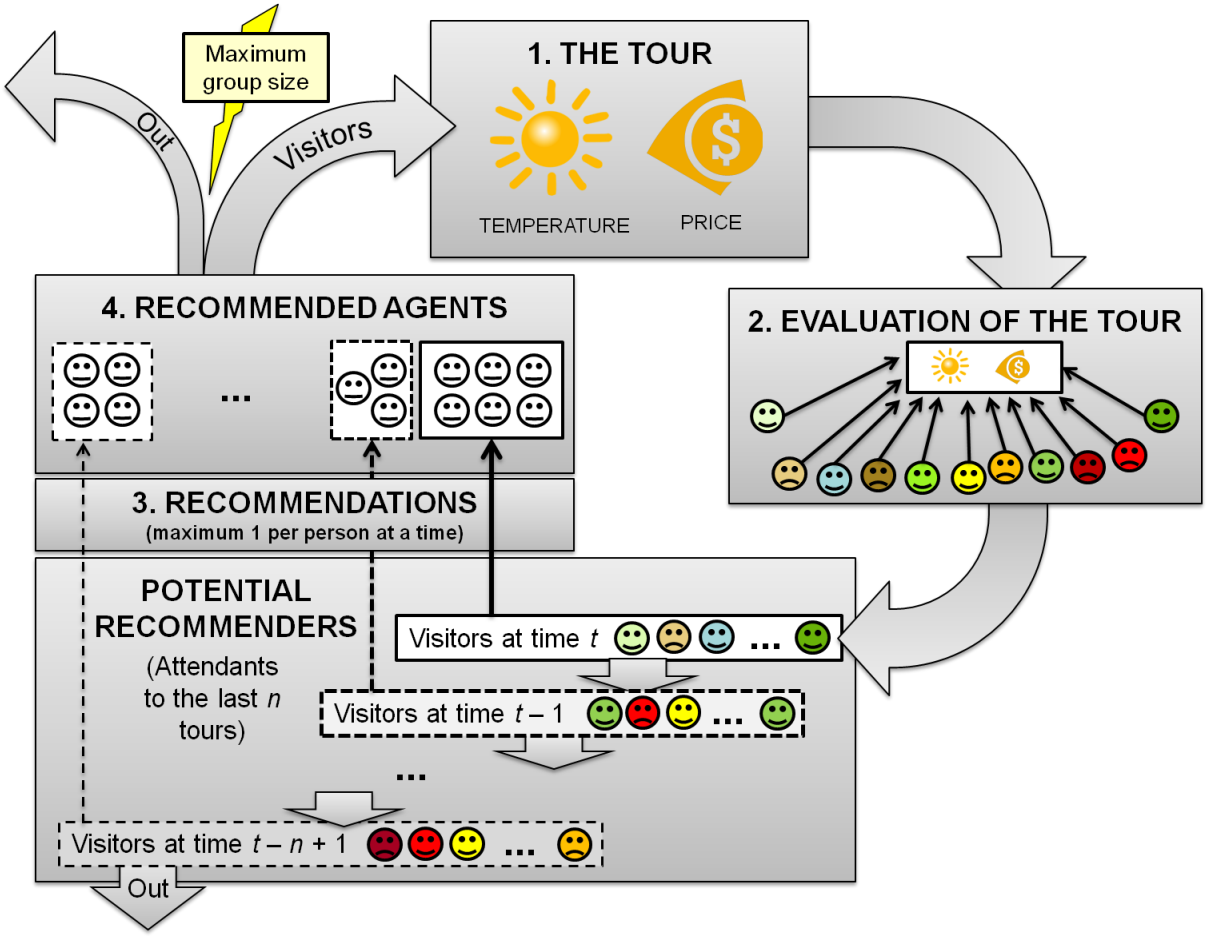

Fig. 1. Sketch of the sequence of events in the model

[1] Other versions of the model include the knowledge and enthusiasm of the tour guide, word-of-mouth recommendations and different types of tourists (i.e. optimistic and pessimistic). For simplicity of presentation, we have reduced the number of membership functions and IF-THEN fuzzy rules here. Upon request, the authors can provide additional information and the models.

A time-step comprises the following sequence of events (we use **red Arial font** for the parameters in the model):

1. The tour. A group of agents goes on the sightseeing tour. These agents are called the visitors of this time-step. The price of the tour is a parameter of the model named **price**, and the temperature is generated randomly (using a uniform distribution with range [-10, 50]) every time a tour takes place (i.e. every time-step). In the very first time-step, the number of visitors is determined by the user by setting the value of a parameter called **initial-num-of-visitors**. In the following time-steps, the number of visitors is calculated endogenously as explained below. Agents can only attend the tour once and the maximum number of visitors at any given tour is a parameter of the model named **group-size**.

2. Evaluation of the tour. Having experienced the tour, each individual visitor computes its own probability of recommending it. This probability is computed using the following two rules:

R1: IF (price is *inexpensive* AND temperature is *nice*),
THEN it is *likely* that I will recommend.

R2: IF (price is *expensive* OR temperature is *extreme*),
THEN it is *unlikely* that I will recommend.

In this model the rules are the same for every visitor, but the perception of what constitutes *nice* versus *extreme* temperature, *inexpensive* versus *expensive* ticket, and *likely* versus *unlikely* are particular to each agent. Specifically, as shown in the family of membership functions drawn in Fig. 2, each individual agent has its own concept (i.e. fuzzy set) of *nice temperature* (blue on left plot), *extreme temperature* (red on left plot), *inexpensive price* (blue on middle plot), *expensive price* (red on middle plot), *likely* (blue on right plot) and *unlikely* (red on right plot).

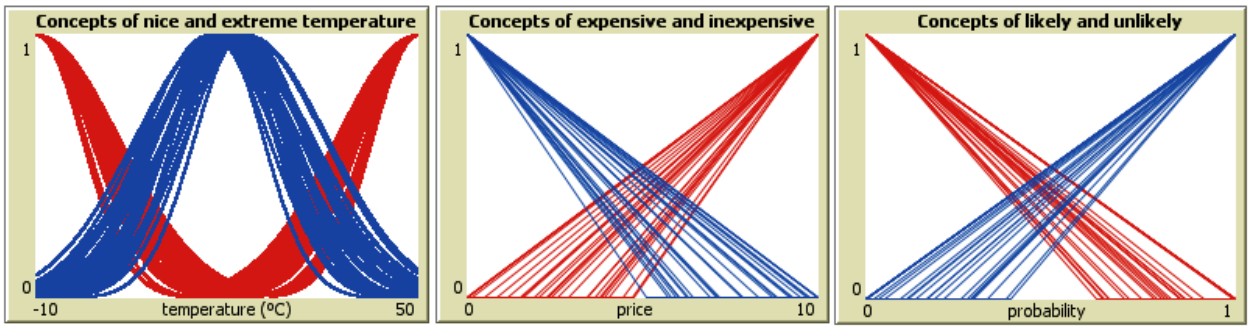

Fig. 2. Representation of various fuzzy sets held by different agents

The individual fuzzy sets are created following a template and adding some noise whose magnitude is controlled with the parameter **variability**. Further details on the creation of the individual fuzzy sets and on the computation of the probability are provided in the following sections of this appendix.

At the end of this second stage, every visitor has computed its own probability of recommending the tour, which will remain unchanged throughout the life of the visitor.

3. Recommendations. Every agent who has attended the tour recently is given one opportunity to recommend the tour to another (newly created) agent. The opportunity will materialise in an actual recommendation according to the agent's probability of recommending. The number of opportunities that each agent is given to recommend the tour is a parameter of the model named **recommend-opportunities**. Since agents are given one opportunity to recommend in each time-step,

and they are given **recommend-opportunities** opportunities, the set of agents who can recommend at any time-step is composed of all the agents who have attended one of the latest **recommend-opportunities** tours (see Fig. 1).

4. Identification of the visitors to the tour in the following time-step. At this point we have a set of new agents who have been recommended the tour. If this set contains fewer than **group-size** agents, all of them will attend the tour in the following time-step. Otherwise, a subset of **group-size** agents is selected randomly from the set of recommended agents to attend the tour in the following time-step. The non-selected recommended agents are disregarded.

In the following time period a new tour starts (with different temperature conditions), which will be assessed by the new set of visitors. These new attendees will compute their own probability to recommend the tour to their own social contacts. In this way, the iterative process can go on indefinitely.

2 Using the model

This tutorial assumes some familiarity with Netlogo. First, place the files **fuzzy-logic-library-model.nlogo** (i.e. the model) and **fuzzy-logic.nls** (i.e. the library) in the same folder. Open the model **fuzzy-logic-library-model.nlogo** with NetLogo[2]. Then, inspect the interface. The graphic user interface (GUI) includes several objects you may already recognise, such as buttons, sliders and plots (Fig. 3).

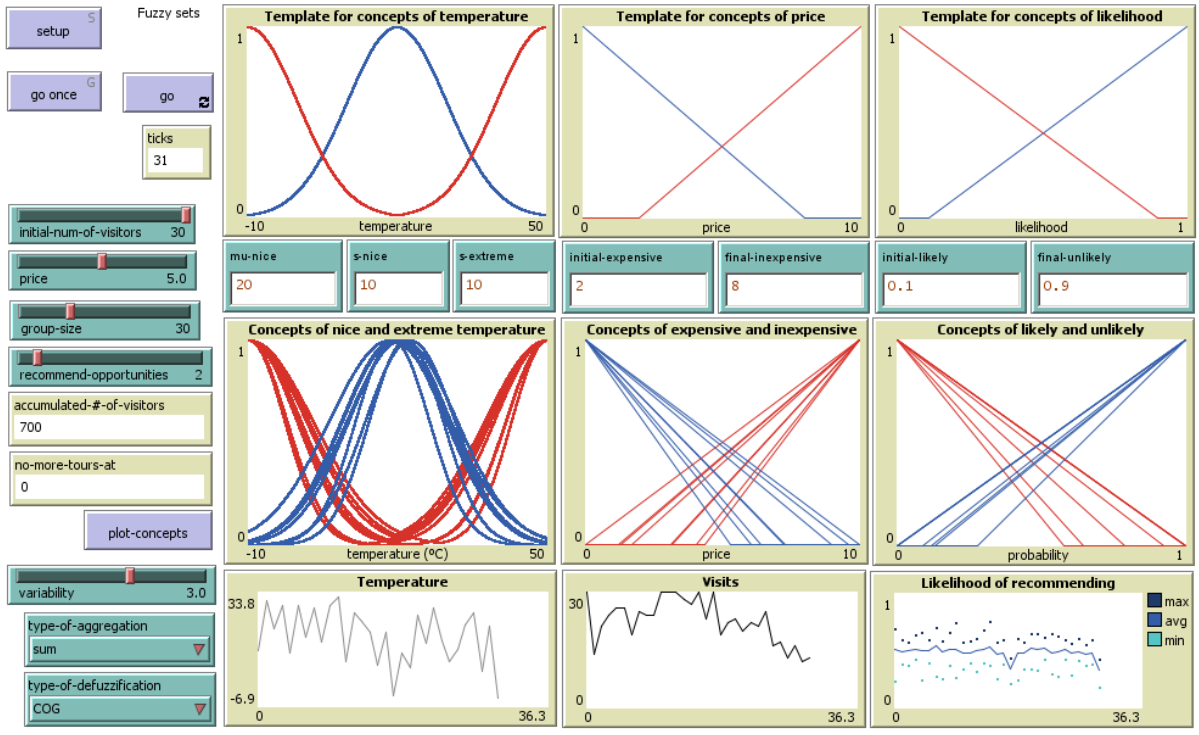

Fig. 3. GUI for the fuzzy-logic-library-model.nlogo model

Five sliders enable the user to interact with the model by setting the value of the parameters **initial-num-of-visitors**, **price**, **group-size**, **recommend-opportunities** and **variability** of the individual fuzzy sets. For demonstration purposes, we set the sliders arbitrarily with default values; for example, in the GUI presented in Fig. 3, the sliders for the **initial-num-of-visitors** and **group-size** have the value

[2] We encourage the reader to use the documentation of the Fuzzy Logic library in conjunction with the model and this tutorial.

30, the maximum number of recommendations **recommend-opportunities** is 2, the ticket **price** is 5 and the **variability** is 3. The value of all parameters can be changed at run time with immediate effect on the model.[3]

We use input boxes to set the parameters for the templates used to create the fuzzy sets: *nice temperature* (**mu-nice** and **s-nice**), *extreme temperature* (**s-extreme**), *expensive price* (**initial-expensive**), *inexpensive price* (**final-inexpensive**), *likely probability* (**initial-likely**) and *unlikely probability* (**final-unlikely**). The precise way in which these templates are created is detailed below:

- Template for *nice temperature*: Gaussian (bell-shaped) fuzzy set with mean **mu-nice**. Parameter **s-nice** plays a role similar to the standard deviation in the sense that it controls how much the function spreads out (see details in the documentation of the library).

- Template for *extreme temperature*: It is the maximum of two Gaussian (bell-shaped) fuzzy sets with means -10 and 50 respectively. Parameter **s-extreme** controls how much both of them spread out from their mean.

- Template for *expensive price*: Fuzzy set with piecewise linear membership function that joins the points [0 0], [**initial-expensive** 0] and [10 1].

- Template for *inexpensive price*: Fuzzy set with piecewise linear membership function that joins the points [0 1], [**final-inexpensive** 0] and [10 0].

- Template for *likely probability*: Fuzzy set with piecewise linear membership function that joins the points [0 0], [**initial-likely** 0] and [1 1].

- Template for *unlikely probability*: Fuzzy set with piecewise linear membership function that joins the points [0 1], [**final-unlikely** 0] and [1 0].

Individual fuzzy sets for the agents are created following the corresponding template and adding some random noise whose magnitude is controlled with the parameter **variability**. For details on this, see section 3 of this appendix.

At the bottom left of the GUI there are two chooser objects, which are used to select the **type-of-aggregation** and the **type-of-defuzzification** for the system of fuzzy IF-THEN rules. The default values are set to sum and centre of gravity (COG) respectively. For details on this, see section 3 of this appendix.

We use **blue Arial font** for the buttons in the model. The **setup** button creates the group of initial visitors, sets everything up and plots the templates used to create all individual fuzzy sets. Every agent has six individual fuzzy sets: two with temperature as base variable (*nice* and *extreme*), two with price as base variable (*expensive* and *inexpensive*) and two with probability as base variable (*likely* and *unlikely*). The templates used to create each of these fuzzy sets are shown on the three plots at the top of the interface. The **plot-concepts** button displays the individual membership functions of the fuzzy sets of the agents who will attend the tour in the following time-step (at the moment the button is pressed) in the three plots in the middle. The **go once** button carries out one time-step of the model and the **go** button makes the model run indefinitely.

[3] Except for parameter **initial-num-of-visitors**, naturally. Changing the value of this parameter will not have an effect once the setup button is pressed.

The GUI also includes three plots at the bottom that display the time series of temperature, number of visitors and (minimum, maximum and average) probabilities of recommendation of the visitors to the tour.

Press the **setup** button. Then press the **plot-concepts** button and observe the different fuzzy sets of the newly created agents. Then try the **go** button for a few seconds. You will obtain something similar to the interface shown in Fig. 4, where all the plots are populated. If you wait for long enough, chances are that a window saying that "There are no visitors" pops up.

Next, change the inputs to another set of values, change the aggregation and defuzzification rules (e.g., from sum to max and from COG to first of maxima FOM) and press the **go** button again. Try running the model several times, with different input values, and note the changes in the output.

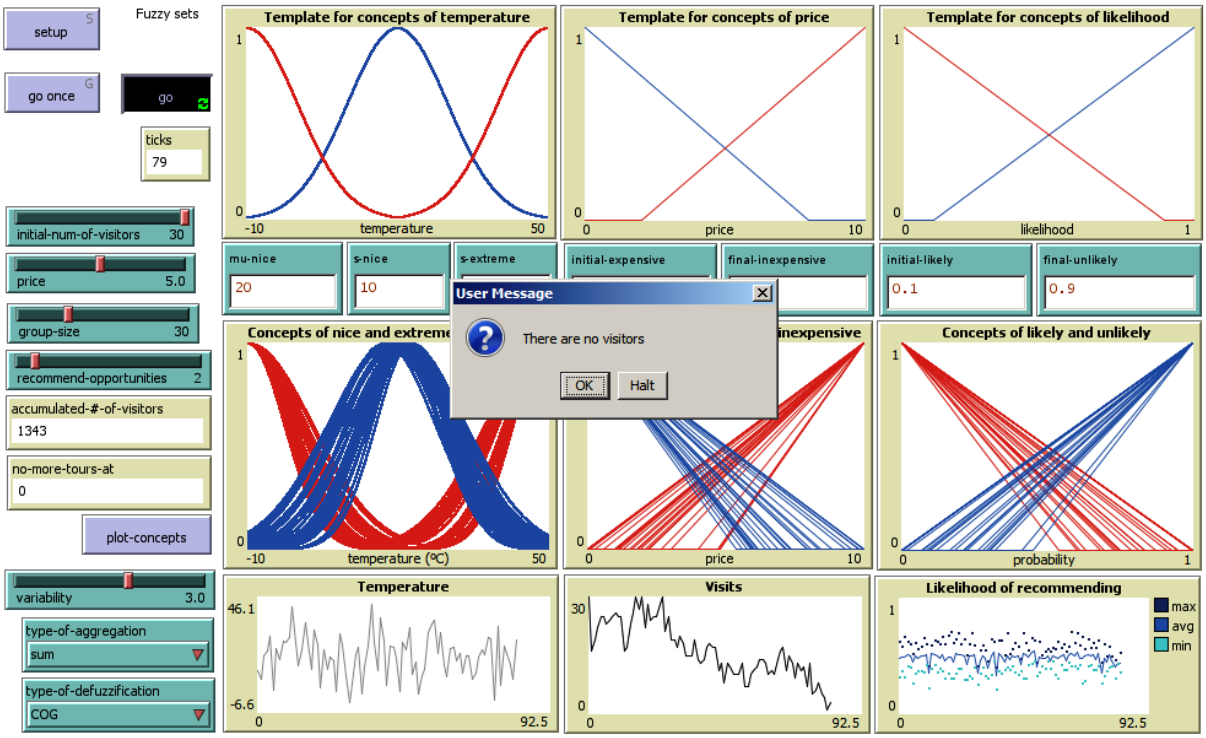

Fig. 4. GUI with results

Then, change from the Interface tab to the Code tab. If you click on "Includes" and select **fuzzy-logic.nls** from the drop-down list (Fig. 5), you will see the code for the functions in the **fuzzy-logic.nls** library (these elements are discussed at length in the documentation of the library). We do not recommend that inexperienced users attempt to modify any of this code. Figure 6 shows the functions in the library.

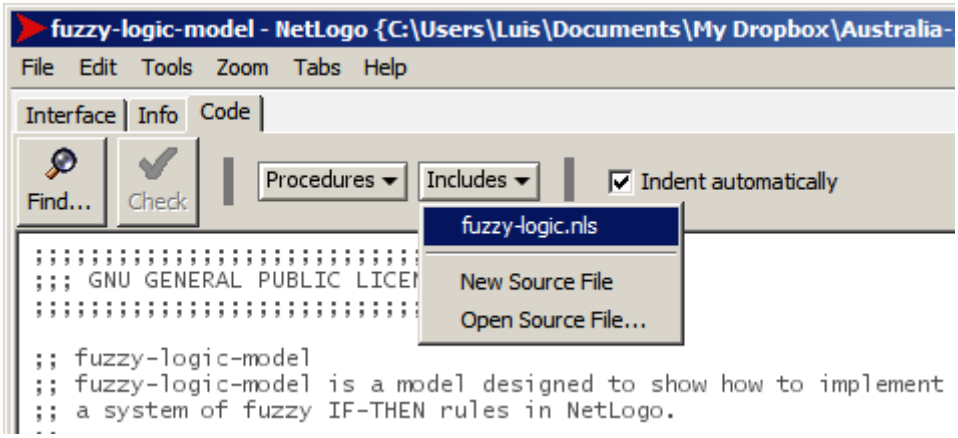

Fig. 5. Include the library file fuzzy-logic.nls

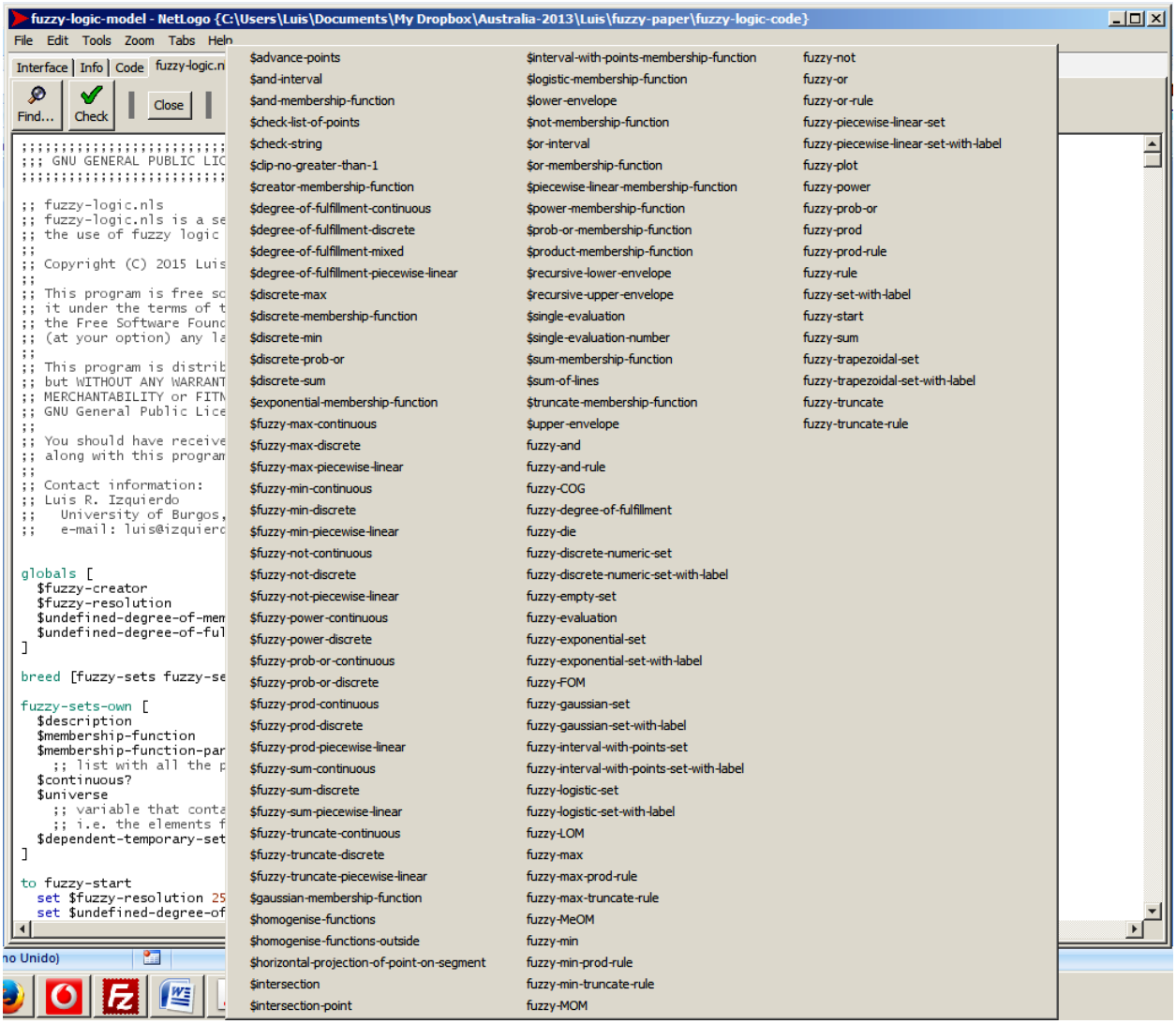

Fig. 6. Contents of the fuzzy-logic.nls library

3 Implementation of systems of fuzzy IF-THEN rules

In this section we explain how to implement the interpolation method for systems of fuzzy IF-THEN rules. In our particular case, the system of fuzzy IF-THEN rules contains the following two rules:

R1: IF (price is *inexpensive* AND temperature is *nice*),
THEN it is *likely* that I will recommend.

R2: IF (price is *expensive* OR temperature is *extreme*),
THEN it is *unlikely* that I will recommend.

It is clear that the first thing we must do is to create the fuzzy sets involved in the rules. In our model, each individual agent creates its own fuzzy sets by running the procedure "create-my-fuzzy-sets". As an example, let us see how to create the fuzzy set that represents an agent's concept of expensive (see Fig. 7). We want the membership function of that fuzzy set to be the piecewise linear function that joins the points [0 0], [*x* 0] and [10 1], where *x* is equal to the parameter **initial-expensive** plus a random number in between –**variability** and +**variability**.

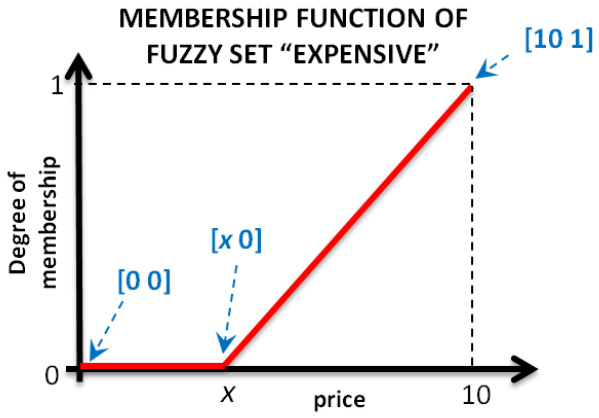

Fig. 7. Membership function of fuzzy set "expensive".

Introducing some stochasticity in the value of *x* ensures that all agents will have different concepts of expensive. The following code shows the excerpt of the procedure "create-my-fuzzy-sets" where the agent creates the described fuzzy set and stores it in its turtle variable called my-concept-of-expensive.

```
      ;; In the following line of code we just compute a value for x,
      ;; making sure that it is always in the range [0 10].
      ;; clip and noise are reporters defined in the next section of code.
let x ( clip [0 10] (initial-expensive + noise variability) )
      ;; In the following line we create the fuzzy set.
set my-concept-of-expensive fuzzy-piecewise-linear-set
                                    (list [0 0] (list x 0) [10 1])
```

Note that **fuzzy-piecewise-linear-set** is a procedure defined in the Fuzzy Logic library which takes a list of points $[[x_1\ y_1]\ [x_2\ y_2]\ \ldots\ [x_n\ y_n]]$ as input and reports a new fuzzy set whose membership function is the piecewise linear function that joins the input points; **initial-expensive** and **variability** are parameters; and **clip** and **noise** are reporters defined elsewhere in the code, as follows:

```
to-report clip [i v]   ;; reports the value v clipped within the interval i
  let f first i
  let l last i
  if v < f [set v f]
  if v > l [set v l]
  report v
end

to-report noise [v]    ;; reports a random number in the interval [-v v]
  report (v - random-float (2 * v))
end
```

Once you have understood the code above, we recommend you read the whole code within the procedure "create-my-fuzzy-sets". Let us now turn to the implementation of the interpolation method for systems of fuzzy IF-THEN rules.

As explained in section 2.6 of the paper, the interpolation method generally comprises five steps which together define a specific function that maps crisp input values into a crisp output value. In our particular case, the function will take numerical inputs *price* and *temperature*, and will produce a numerical *probability* as its output, i.e. *probability* = *f*(*price*, *temperature*). The actual name of the procedure that implements this function in our model is "compute-probability", a procedure to be run by each individual agent and which does not take any inputs because agents can directly access the value of *price* and *temperature* (since they are both global variables). The procedure does not actually report anything, but it does set the value of *my-prob-of-recommending*, which is a variable owned by the agent. Here we explain how to implement such a function in NetLogo.

1. Fuzzification of inputs (optional)

In our model we have not fuzzified the inputs *price* and *temperature*.

2. Computation of reshaped consequents for each rule.3F[4]

The first rule reads:

R1: IF (price is *inexpensive* AND temperature is *nice*), THEN it is *likely* that I will recommend.

In the model we use the function min for the logical operator AND and the function truncate as reshaping method. Thus, we select the procedure **fuzzy-min-truncate-rule** to implement our first rule R1. (Since this is the most common selection of operators for AND rules, the function **fuzzy-min-truncate-rule** is also called **fuzzy-and-rule**.)

```
let R1 fuzzy-and-rule (list
        (list price my-concept-of-inexpensive)
        (list temperature my-concept-of-nice)  )     my-concept-of-likely
```

[4] Note that this step corresponds to stages 2 (Computation of degrees of consistency between inputs and antecedents) and 3 (Reshaping of consequents) in the paper.

Note that R1 is now the reshaped consequent.
The second rule reads:

R2: IF (price is *expensive* OR
temperature is *extreme*), THEN it is *unlikely* that I will recommend.

In the model we use the function max for the logical operator OR and the function truncate as reshaping method. Thus, we select the procedure **fuzzy-max-truncate-rule** to implement our second rule R2. (Since this is the most common selection of operators for OR rules, the function **fuzzy-max-truncate-rule** is also called **fuzzy-or-rule**.)

```
let R2 fuzzy-or-rule (list
        (list price my-concept-of-expensive)
        (list temperature my-concept-of-extreme)  )  my-concept-of-unlikely
```

Note that R2 is now the reshaped consequent.

3. Aggregation of all the reshaped consequents

In this step all the reshaped consequents are combined together to form a single fuzzy set for the output variable. In the example below we use the function max as aggregation operator. Thus, we select the procedure **fuzzy-max** to aggregate all the reshaped consequents.

```
let my-prob-of-recommending-fuzzy-set fuzzy-max (list R1 R2)
```

Note that *my-prob-of-recommending-fuzzy-set* is now the aggregated fuzzy set. The actual code for the aggregation in the model is more general and allows the user to change the type of aggregation at runtime by modifying the value of the parameter **type-of-aggregation** (which can be set to "max", "prob-or" or "sum").

```
let my-prob-of-recommending-fuzzy-set
        (runresult (word "fuzzy-" type-of-aggregation " (list R1 R2)"))
```

4. Defuzzification of the aggregated fuzzy set (optional)

In the example below we use the function Centre of Gravity (COG) as defuzzification operator. Thus, we select the procedure **fuzzy-COG** to defuzzify the aggregated set.

```
set my-prob-of-recommending [fuzzy-COG] of my-prob-of-recommending-fuzzy-set
```

Note that *my-prob-of-recommending* is now a number in between 0 and 1. The actual code for the defuzzification in the model is more general and allows the user to change the type of defuzzification at runtime by modifying the value of the parameter **type-of-defuzzification** (which can be set to "COG", "FOM", "LOM", "MOM" or "MeOM").

```
set my-prob-of-recommending
        [(runresult (word "fuzzy-" type-of-defuzzification))]
                                        of my-prob-of-recommending-fuzzy-set
```

5. Finesse the use of memory

Finally, if you do not want to overuse memory, kill all the fuzzy sets that have been created in this procedure and you are not going to use anymore.

```
ask (turtle-set R1 R2 my-prob-of-recommending-fuzzy-set) [fuzzy-die]
```

So the final code for the implementation of the interpolation method is the following:

```
to compute-probability

;; COMPUTATION OF RESHAPED CONSEQUENTS FOR EACH RULE
  let R1 fuzzy-and-rule (list
        (list price my-concept-of-inexpensive)
        (list temperature my-concept-of-nice)  )     my-concept-of-likely
  let R2 fuzzy-or-rule (list
        (list price my-concept-of-expensive)
        (list temperature my-concept-of-extreme)  )  my-concept-of-unlikely

;; AGGREGATION OF ALL THE RESHAPED CONSEQUENTS
  let my-prob-of-recommending-fuzzy-set
        (runresult (word "fuzzy-" type-of-aggregation " (list R1 R2)"))

;; DEFUZZIFICATION OF THE AGGREGATED FUZZY SET
  set my-prob-of-recommending
        [(runresult (word "fuzzy-" type-of-defuzzification))]
                                        of my-prob-of-recommending-fuzzy-set

;; MEMORY FINESSE
  ask (turtle-set R1 R2 my-prob-of-recommending-fuzzy-set) [fuzzy-die]

end
```

4 Final comments

We present a simplified model with the objective of highlighting the benefits of fuzzy logic in an environment in which different agents interact with one another. As illustrated, the NetLogo environment allows for each agent to have its own linguistic terms (i.e. fuzzy sets) and/or rules of decision. Here, we present a case in which all the agents share the same fuzzy rules (adopting the view that these rules represent the "general wisdom") but they interpret them in different ways, i.e. the meaning of the linguistic terms (defined by their membership functions *nice*, *extreme*, *inexpensive*, *expensive*, *likely*, *unlikely*) is internal to each agent and reflects the individual heterogeneity of interpretations. Therefore, the fuzziness of the system resides in the way people feel, understand or treat those words in any particular situation: whereas 28 Celsius may be quite hot for someone living in a cold climate, it may be perceived as cold for a tropical island resident; the same price may be considered cheap or expensive depending on the person's socioeconomic status.

This treatment does not need to be the case. For example, the researcher may want to test the reverse case, with universal membership functions and distinct, "personalised" rules for the agents, including hedges and various combinations of unions and intersections of conditions. The Fuzzy Logic library allows for this flexibility. Also, note that this model uses only two rules for the sake of clarity in the exposition. However, adding more rules is straightforward following the steps in section 3 of this appendix.

A final word of caution: as explained in section 2.7 of the paper, the interpolation method is not logical deductive inference. Thus, caution is required in interpreting the output of a system of fuzzy IF-THEN rules.