In [1]:

```
%load_ext autoreload
%autoreload 2
```

Lab 03 - Fuzzy rules

- Professor: Carlos Peña (carlos.pena@heig-vd.ch (mailto:carlos.pena@heig-vd.ch))
- Assistant 2018: Gary Marigliano (gary.marigliano@heig-vd.ch (mailto:gary.marigliano@heig-vd.ch))
- Assistant 2019: Diogo Leite (diogo.leite@heig-vd.ch (mailto:diogo.leite@heig-vd.ch))

Date: Winter 2019

Instructions:

- Read this notebook
- Do/Answer where **TODO student** is specified
- The folder structure is like this:

```
fuzzy_systems
├── core
└── view
```

- `core` contains core classes like membership_functions, fuzzy_rules,...
- `view` contains classes used to display what the core classes do.
- Please keep this structure when you will do the exercises.

TODO student Read and explore the code provided both in this folder.

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

In [3]:

```
# basic fuzzy operators
OR_max = (np.max, "OR_max")
AND_min = (np.min, "AND_min")
MIN = (np.min, "MIN")
```

In [4]:

```
from fuzzy_systems.core.linguistic_variables.linguistic_variable import Linguistic
Variable
from fuzzy_systems.core.membership_functions.lin_piece_wise_mf import LinPWMF
from fuzzy_systems.core.rules.fuzzy_rule import FuzzyRule, Antecedent, Consequent
```

In [5]:

```
!pygmentize fuzzy_systems/core/rules/fuzzy_rule.py
```

```
from collections import defaultdict
from copy import deepcopy
from typing import Dict, List, Callable, Tuple

from fuzzy_systems.core.membership_functions.free_shape_mf import Fre
eShapeMF
from fuzzy_systems.core.rules.fuzzy_rule_element import Antecedent, C
onsequent


class FuzzyRule:
    def __init__(self,
                 ants: List[Antecedent],
                 ant_act_func: Tuple[Callable, str],
                 cons: List[Consequent],
                 impl_func: Tuple[Callable, str]):
        """
        Define a fuzzy rule

        Assumptions:
        * the antecedent's activation function is the same for all co
nsequents
        * multiple antecedents and consequents can be used for a sing
le rule

        :param ants: a list of Antecedent

        :param ant_act_func: A Tuple[Callable, str] where the callabl
e is
        either a t-norm or a t-conorm operator and where the string i
s used
        for visualization purposes. Generally, FIS.AND_min or FIS.OR_
max is used

        :param cons:a list of Consequent

        :param impl_func: Implication function i.e. the function f(a,
b) where
        a is a scalar, the result value of the antecedents activation
of this
        rule, and where b represents the membership function(s) of th
e
        consequent(s) used in the rule. This function will return an
implicated
        membership function. Generally, min or product are used.
        """
        self._ants = ants
        self._ant_act_func = ant_act_func
        self._cons = cons
        self._impl_func = impl_func

    @property
    def antecedents(self):
        return self._ants

    @property
    def consequents(self):
```

```
        return self._cons

    def fuzzify(self, crisp_inputs: Dict[str, float]) -> List[float]:
        """
        This function will fuzzify crisp input values on each rule's
antecedents

        :param crisp_inputs: the rule's antecedents crisps inputs val
ues i.e. a
        user's/dataset sample input. Example crisp_inputs = {"tempera
ture": 18,
        "sunshine": 55}

        :return: a list of fuzzified inputs (same size as the number
of
        antecedents) for this particular rule
        """

        fuzzified_inputs_for_rule = []

        for antecedent in self.antecedents:
            crisp_input = crisp_inputs[antecedent.lv_name.name]

            fuzzified_input_for_rule = antecedent.lv_name[antecedent.
lv_value].fuzzify(crisp_input)

            if antecedent.is_not:
                fuzzified_input_for_rule = 1 - fuzzified_input_for_ru
le


            fuzzified_inputs_for_rule.append(fuzzified_input_for_rul
e)

        return fuzzified_inputs_for_rule

    def activate(self, fuzzified_inputs):
        """
        Compute and return the antecedents activation for this rule
        :param fuzzified_inputs:
        :return: a scalar that represents the antecedents activation
        """
        ant_val = fuzzified_inputs[0]

        # apply the rule antecedent function using a sliding window o
f size 2
        for i in range(1, len(fuzzified_inputs)):
            ant_val = self._ant_act_func[0]([ant_val, fuzzified_input
s[i]])

        return ant_val

    def implicate(self, antecedents_activation):
        """
        Compute and return the rule's implication for all the consequ
ents for
        this particular rule.
        A rule's implication is computed as follow:
        RI_for_consequent_C =  implication_func(antecedents_activatio
```

```
n, C)

        :param antecedents_activation: the rule's antecedents activat
ion value.
        So the scalar value returned by self.activate()
        :return: a list (in the same order as the consequents were gi
ven in
        the constructor) of FreeShapeMF objects that represents the r
ule's
        consequents (i.e. output variables) after applying the implic
ation
        operation
        """

        impl_func = self._impl_func[0]
        implicated_consequents = defaultdict(list)

        for con in self._cons:
            # get the output variable's MF used by this specific cons
equent
            # in this rule. For example the MF of "warm" in the case
of
            # the linguistic variable "temperature".
            ling_value = con.lv_name[con.lv_value]

            in_values = deepcopy(ling_value.in_values)  # FIXME deepc
opy needed?
            mf_values = [impl_func([val, antecedents_activation]) for
                         val in ling_value.mf_values]

            # lv_name.name is the name of the linguistic variable, e.
g.
            # "temperature"
            implicated_consequents[con.lv_name.name].append(
                FreeShapeMF(in_values, mf_values))

        return implicated_consequents

    def get_output_variable_names(self):
        return [con.lv_value.name for con in self.consequents]

    def __repr__(self):
        text = "IF ({}), THEN ({})"

        ants_text = " {} ".format(self._ant_act_func[1]).join(
            ["{} is {}".format(a.lv_name.name, a.lv_value) for a in
                self.antecedents])

        cons_text = " {} ".format(",").join(
            ["{} is {}".format(c.lv_name.name, c.lv_value) for c in
                self.consequents])

        return text.format(ants_text, cons_text)
```

TODO student

- Explore the code in `fuzzy_system.core.rules` module
- Implement the parts where **TODO student** is mentionned
  - In `fuzzy_rule.py` complete the implementation of `fuzzify()`. You must take care of the NOT antecedents (reminder: the NOT is simply $1 - \mu_{antecedent_i}(x)$). You can then verify your implementation by running the small unit tests below.

Small unit tests

In [6]:

```
lv_quality = LinguisticVariable(name="quality", ling_values_dict={
     "poor": LinPWMF([0, 1], [5, 0]),
     "average": LinPWMF([0, 0], [5, 1], [10, 0]),
     "good": LinPWMF([5, 0], [10, 1])
})

lv_service = LinguisticVariable(name="service", ling_values_dict={
    "poor": LinPWMF([0, 1], [5, 0]),
    "average": LinPWMF([0, 0], [5, 1], [10, 0]),
    "good": LinPWMF([5, 0], [10, 1])
})

lv_tip = LinguisticVariable(name="tip", ling_values_dict={
    "low": LinPWMF([0, 1], [13, 0]),
    "medium": LinPWMF([0, 0], [13, 1], [25, 0]),
    "high": LinPWMF([13, 0], [25, 1])
})

r1 = FuzzyRule(
    ants=[
        Antecedent(lv_quality, "poor"),
        Antecedent(lv_service, "average", is_not=True)
    ],
    ant_act_func=OR_max,
    cons=[
        Consequent(lv_tip, "low"),
    ],
    impl_func=MIN
)

crisp_inputs_list = [
    {"quality": 3, "service" : 6},
    {"quality": 8, "service" : 3},
    {"quality": -10, "service" : 6},
    {"quality": 9, "service" : 7}
]

expected_outputs = [
    [0.4, 0.2],
    [0.0, 0.4],
    [1.0, 0.2],
    [0.0, 0.4]
]

outputs = []

for crisp_inputs in crisp_inputs_list:
    out = r1.fuzzify(crisp_inputs)
    outputs.append(out)

assert np.allclose(expected_outputs, outputs)
```

Exercice - please answer below

To submit

- Please make a zip called `lfa_labXX_YY.zip` where `XX` is the lab number and `YY` is your familly name. For example: `lfa_lab02_smith.zip` .
- The mail's subject is `[LFA] rendu labXX` where `XX` is the lab number

The zip must contain all *needed* the files to run this notebook. That is, don't send your virtualenv (only the requirements.txt)**. If any additional steps are required to run your notebook(s)/code, please add a README.md where you indicate all the needed steps to reproduce your work.**

Note: Your notebooks must run completely even after the Jupyter kernel has been restarted. To ensure it will be the case when your lab will be reviewed, please select in the top menu "Kernel -> Restart and Run all" and check that the output of each cell is the desired output you want to submit.