Fuzzy controllers

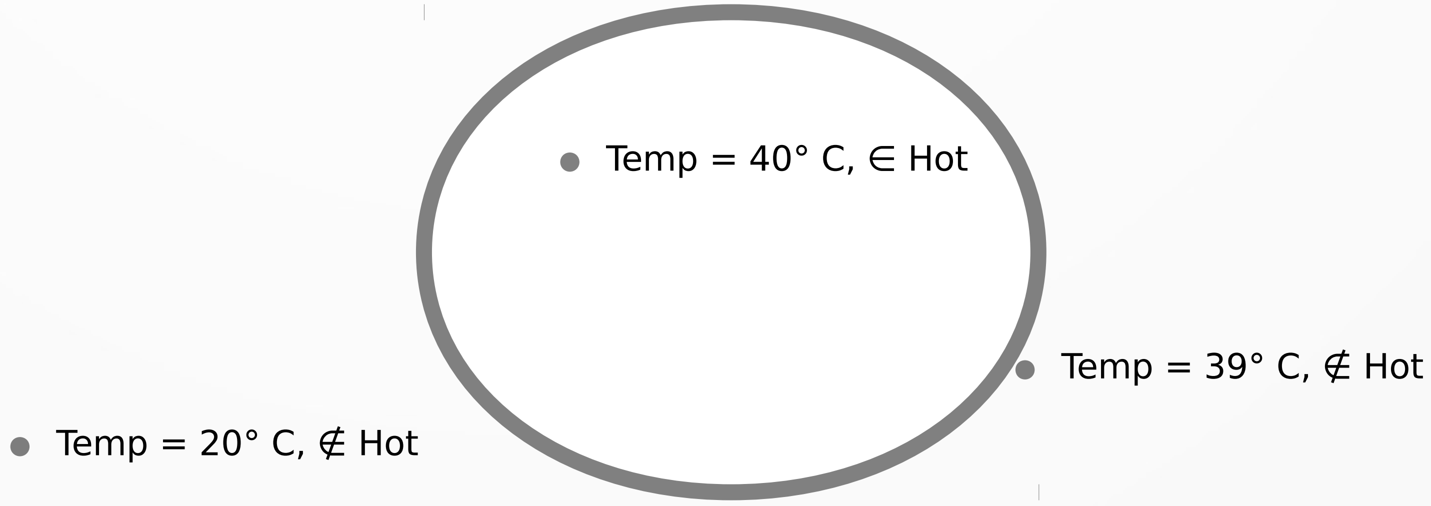
How does it work?

Motivation and basics

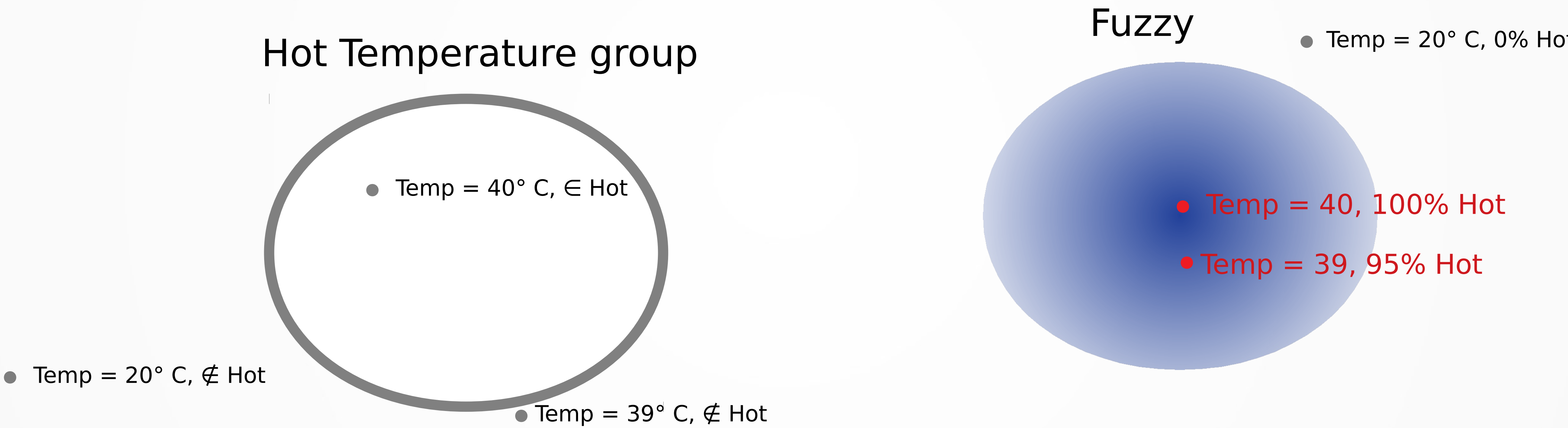
Conventional Logic

Conventional logic and group theory says that either an element is inside or outside a group, no in between.

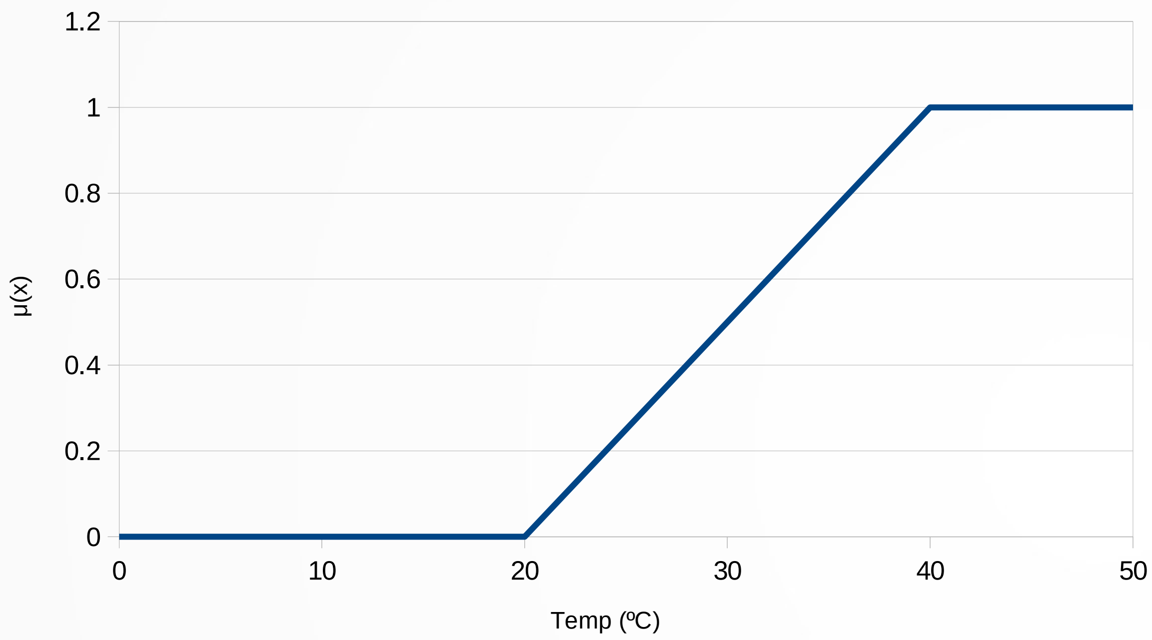
Hot Temperature group



- In fuzzy groups, elements can be somewhat inside a group
- Boundaries are “clouds” rather than “walls”



- Defining a membership function to “Hot” group as Ramp (20, 40)

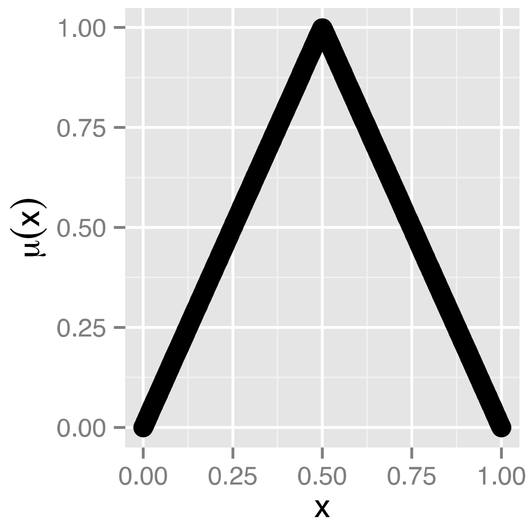


- Temp = 20, μ(x) = 0
- Temp = 25, μ(x) = 0.25
- Temp = 35, μ(x) = 0.75
- Temp = 40, μ(x) = 1

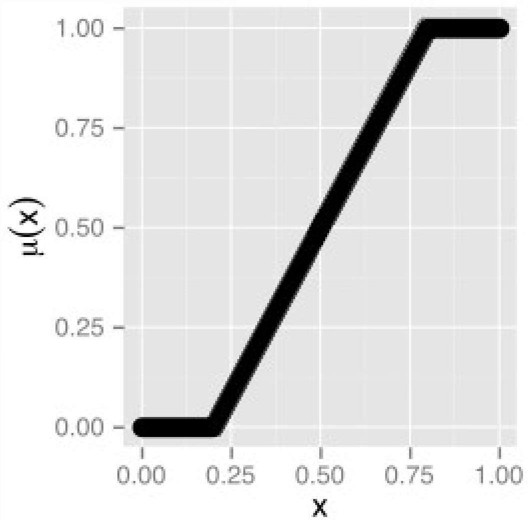
- An element’s degree of membership to a group is according to a membership function $\mu(x) \rightarrow [0,1]$

- Some of the most used membership functions are:

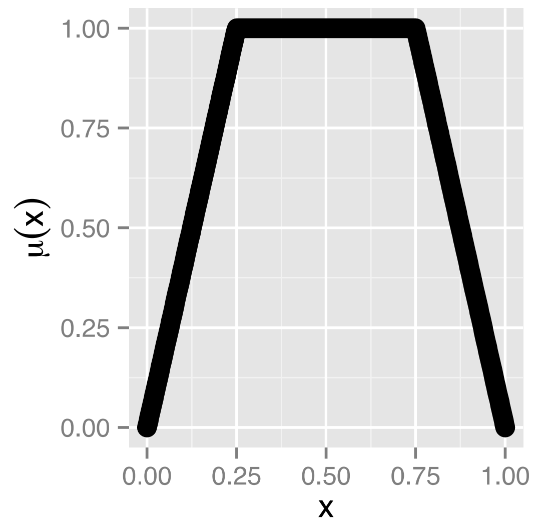
Triangular



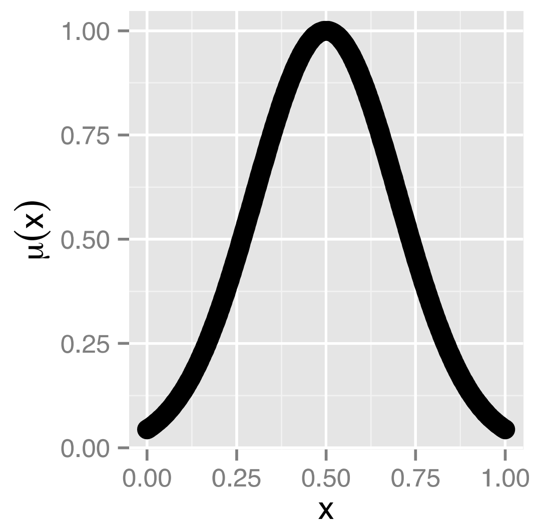
Ramp



Trapezoid

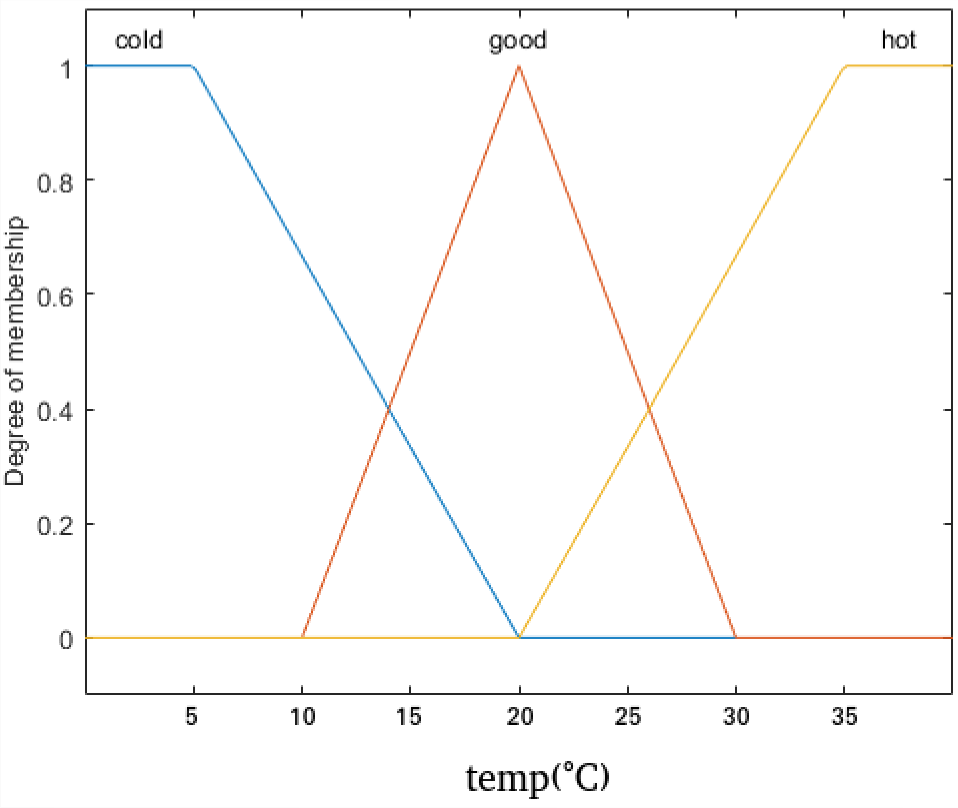


Gaussian



Source: fuzzylite

- We can even add more fuzzy groups, such as Cold, Good and Hot

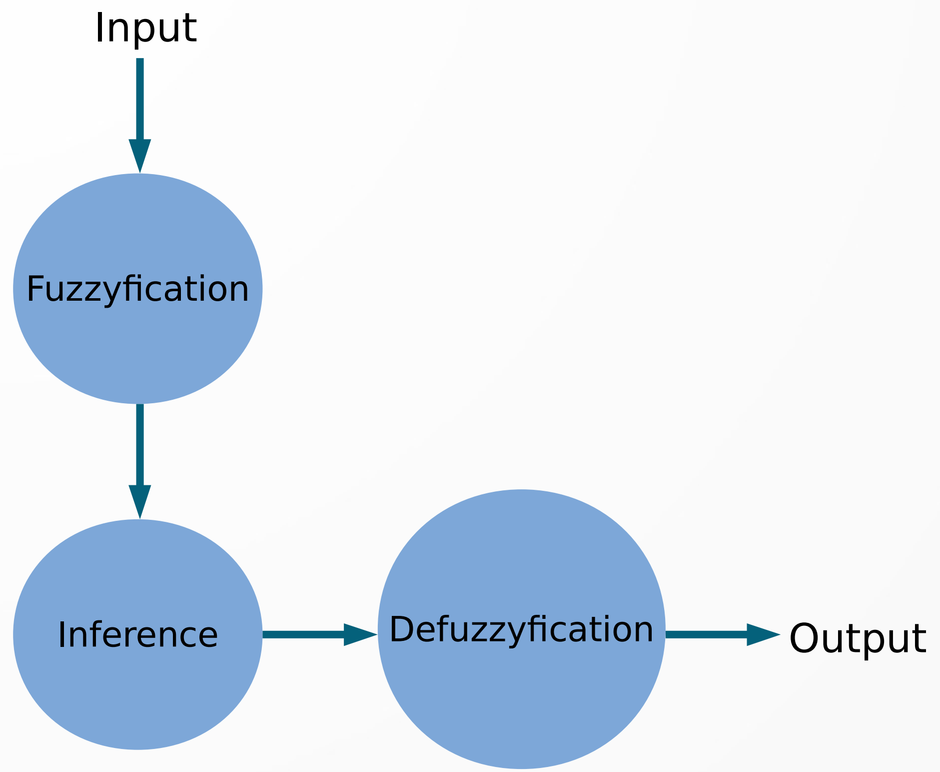


- temp = 15 → μ = (0.5, 0.333, 0)

- This way we have a bridge from Real numbers to Fuzzy groups.

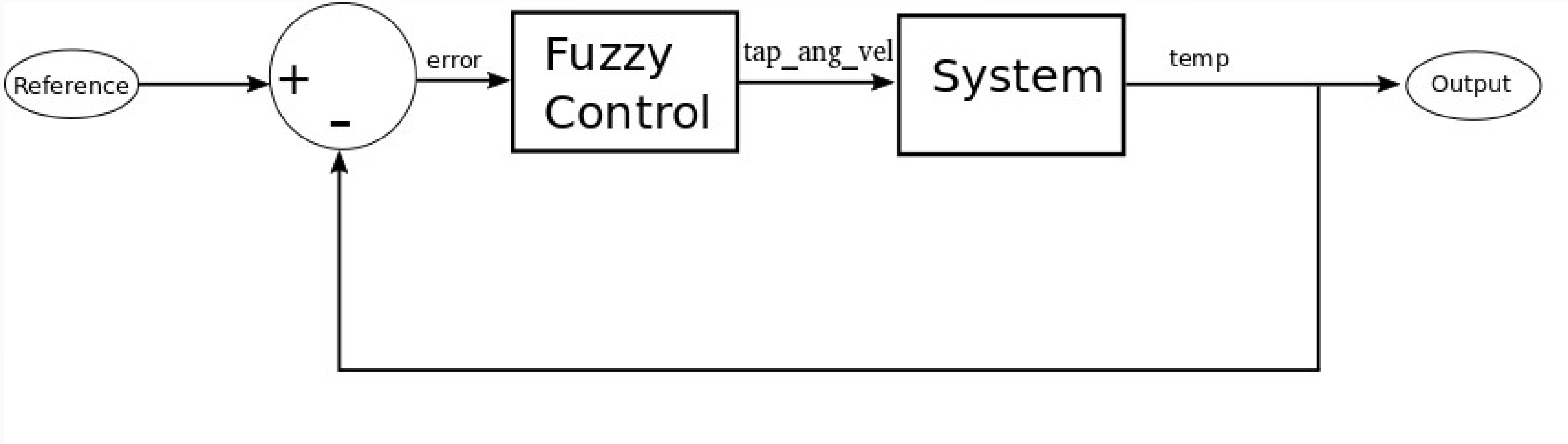
Fuzzy Controllers

- Fuzzy controllers have this structure:



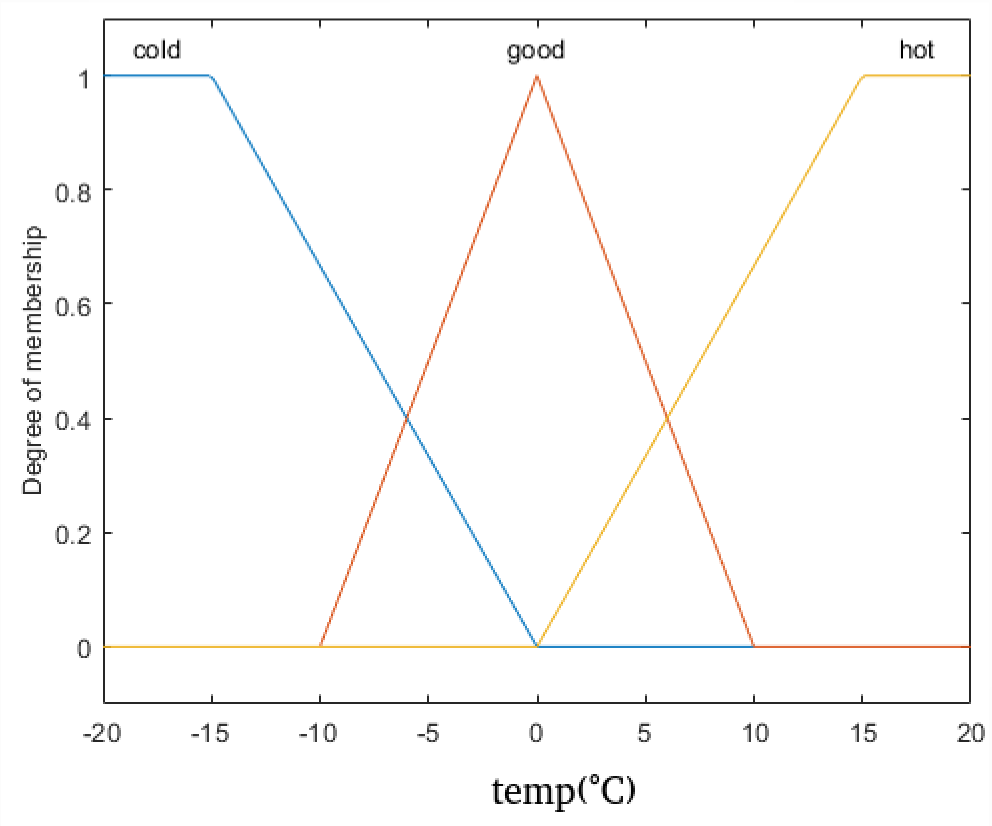
- The process of fuzzyfication is what shown previously, applying the membership functions

- Example system: Controlling the temperature of a tap
- We can use the same classical structure



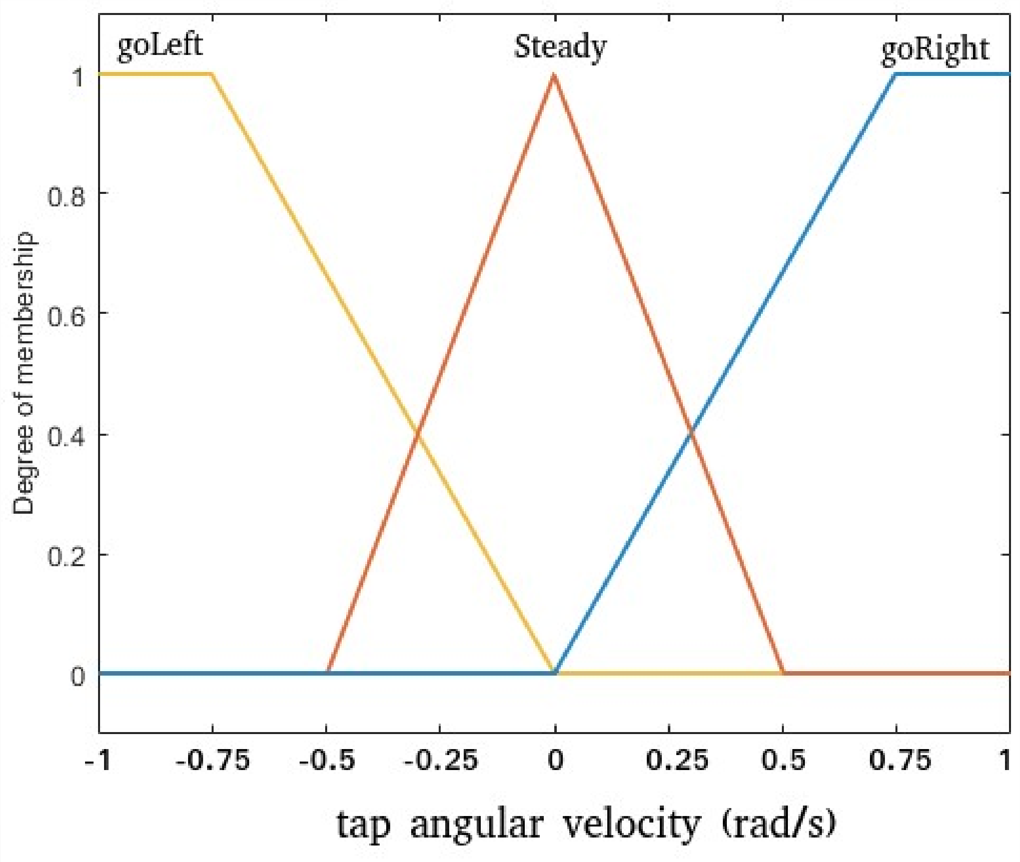
- Fuzzy control takes the place of a classic PID
- Just like classical control, it is smart to make the Fuzzy Controller receive the error instead of the reference

- Firstly, we have to define membership functions to the controller input
- Remember that the input is the error between temperature and reference



• Defining membership functions to the output

• The output represents how to change tap angle, moving it to the right (colder) or left (hotter)



- Fuzzy Inference
  - Fuzzy inference defines how the inputs affect the outputs
  - This definition uses Fuzzy Rules, an implication and aggregation operator

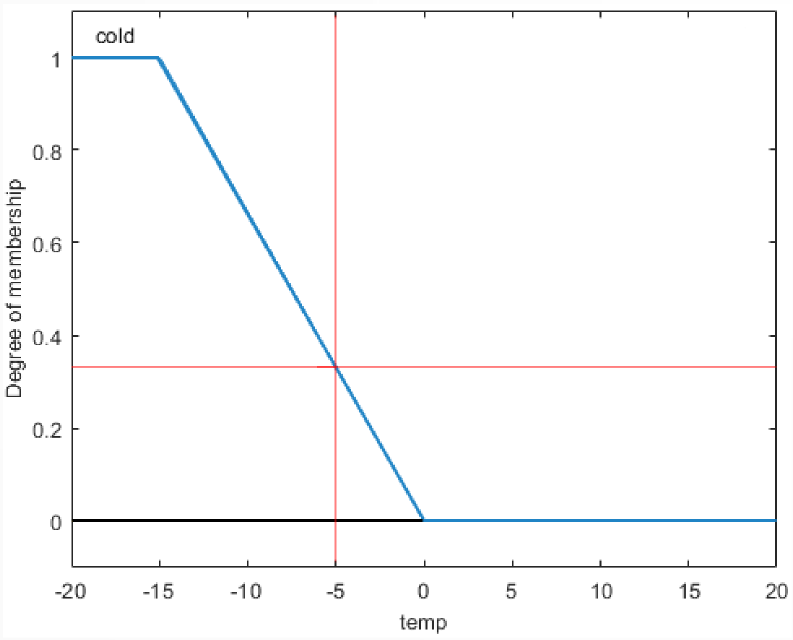
Fuzzy Rule
If “x” is “A” then “y” is “B” “x” is input “y” is output “A” and “B” are fuzzy groups.

  - In our example:

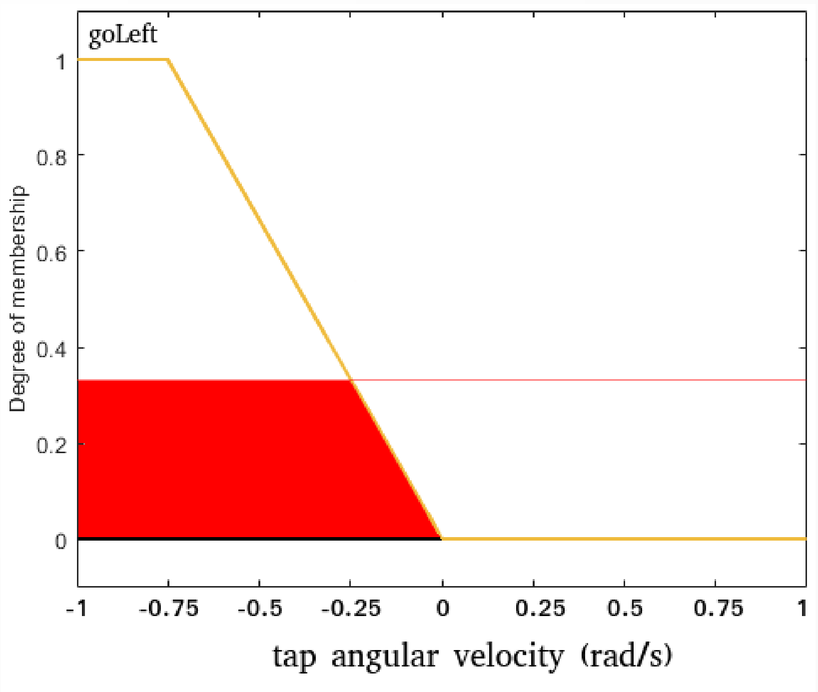
if "temp” is “cold” then “tap_angle_vel” is “goLeft”

- Fuzzy rule activation
  - After defining the rules, we have to define how it activates
  - For this, we need an implication method, usually **Minimum** between the input membership value and the output membership function
  - Ex: for rule: if “temp” is “cold” then “tap_angle_vel” is “goLeft”

temp = -5 $\rightarrow \mu_{cold}(-5) = 0.333$



Min(0.333, $\mu_{Left}(y)$)



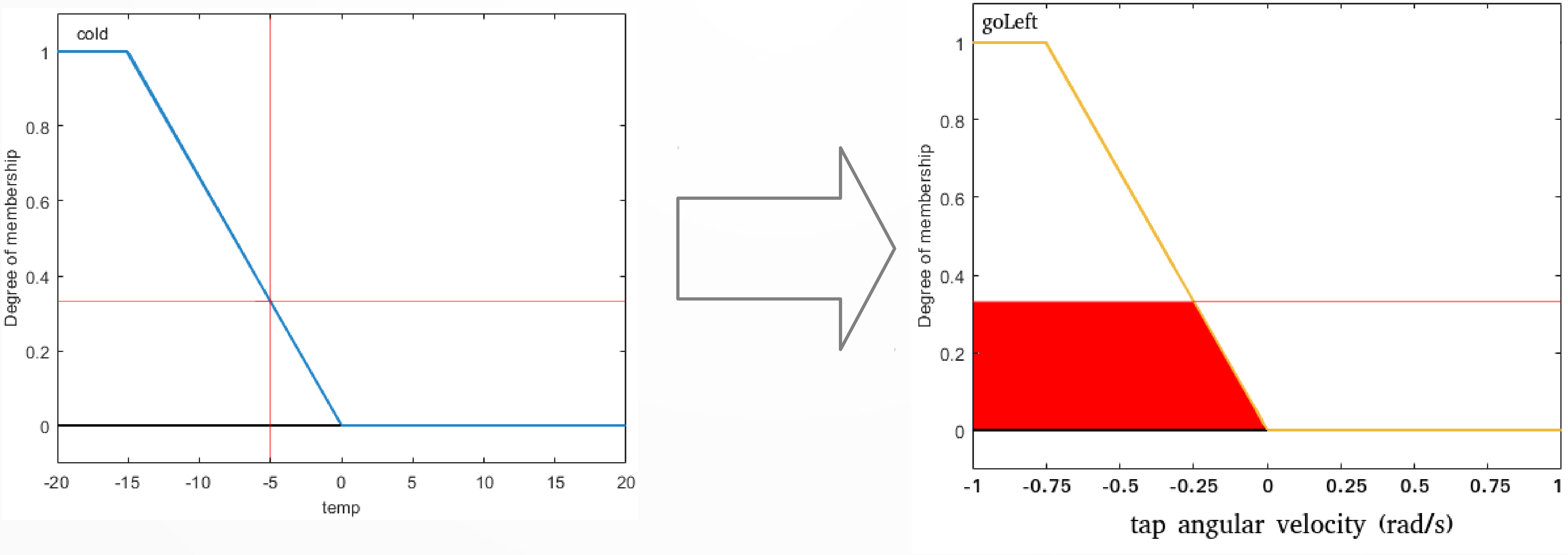
Aggregation

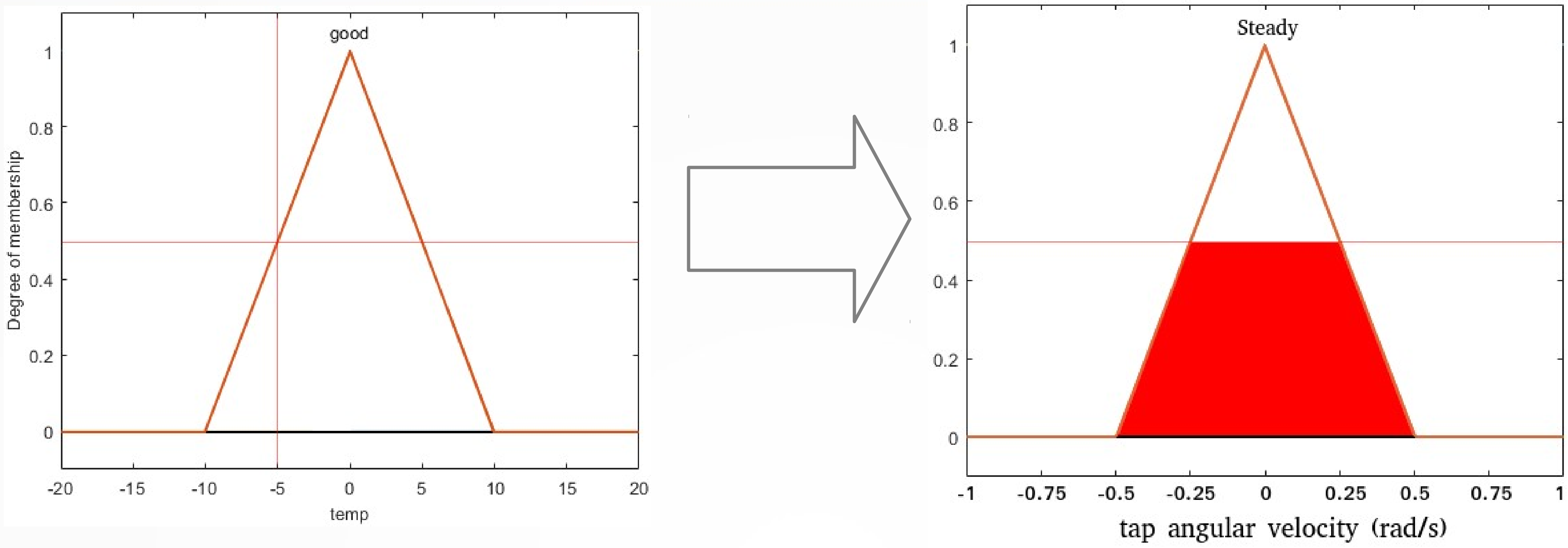
- The aggregation method calculates the fuzzy output, based on the rules set
- Usually, this is the **Maximum** operator for each fuzzy group
- Ex: Aggregating two rules

if “temp” is “cold” then “tap_angle_vel” is “goLeft”

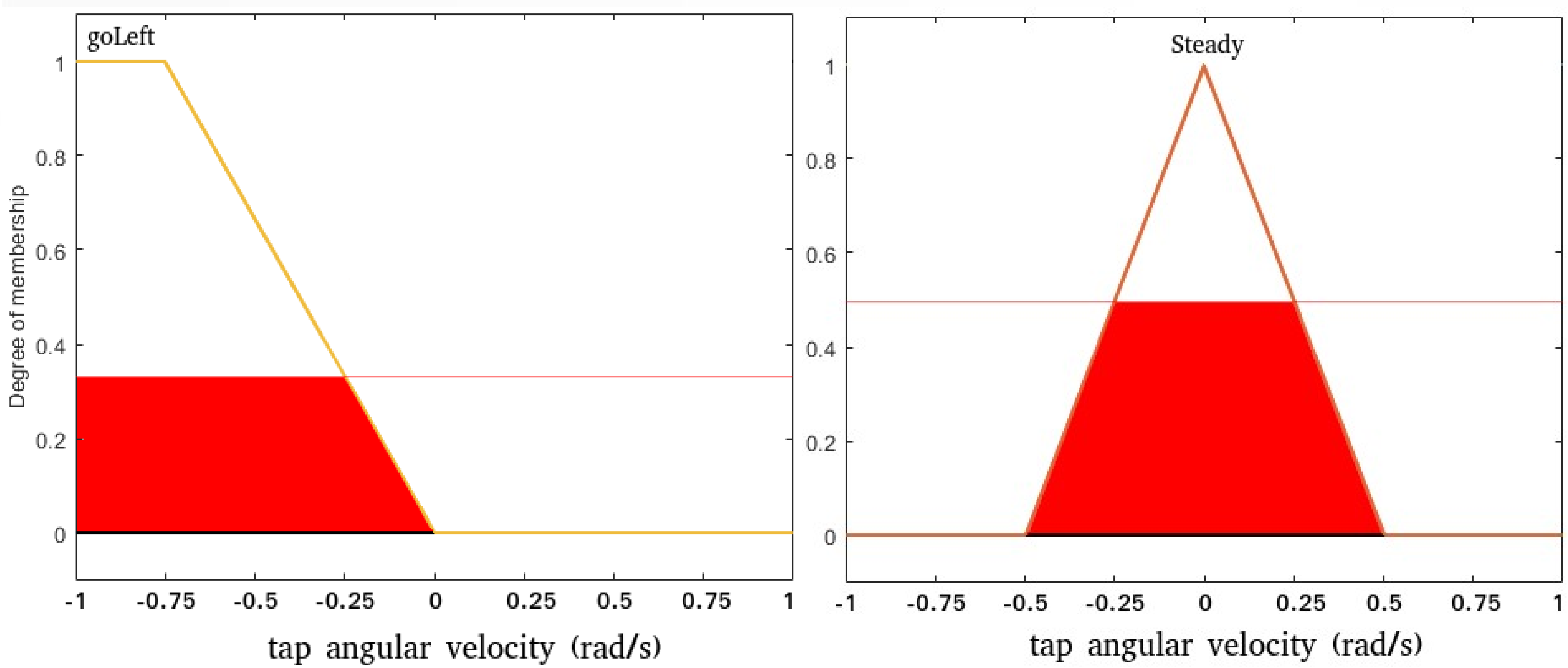
if “temp” is “good” then “tap_angle_vel” is “Steady”

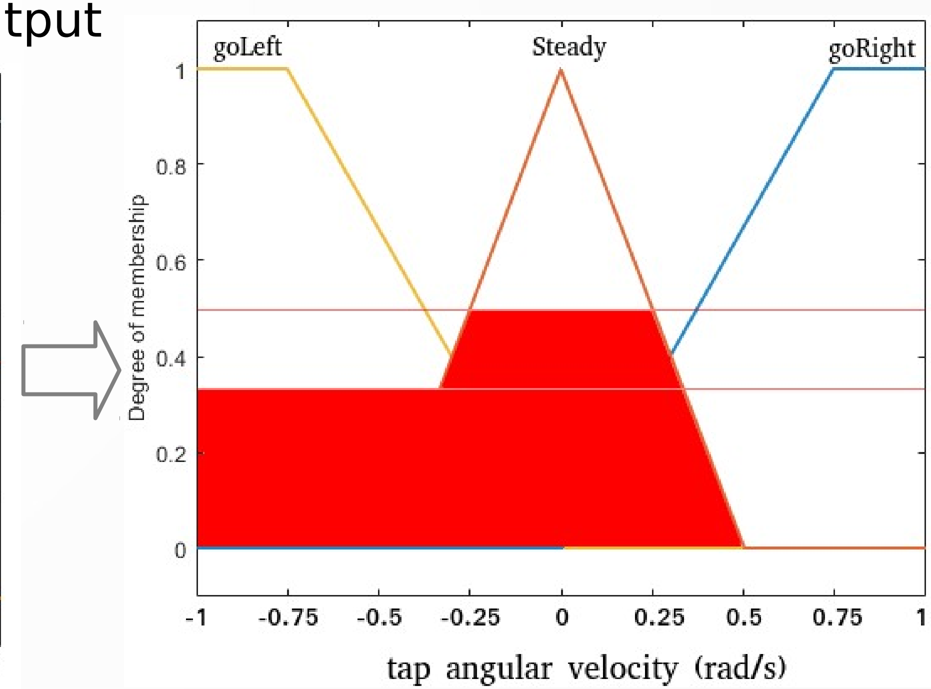
- Temp = -5 → $\mu_{cold}(-5) = 0.333$





- And using the maximum operator to get the final fuzzy output

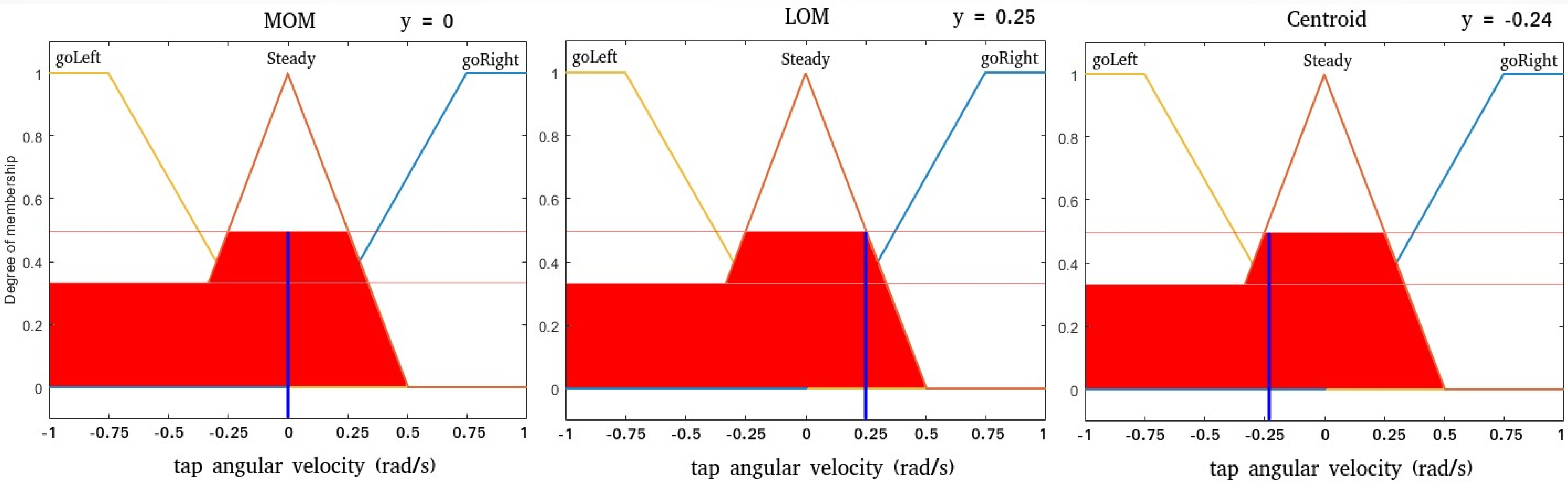




➢ Defuzzyfication

- The last step is to convert the final fuzzy output into a usable number

- This operator is left as a choice, such as: Largest Of Maxima (LOM), Mean of Maxima (MOM), Center Of Area (Centroid), and others



- The defuzzification method is up to the designer and usually is results oriented

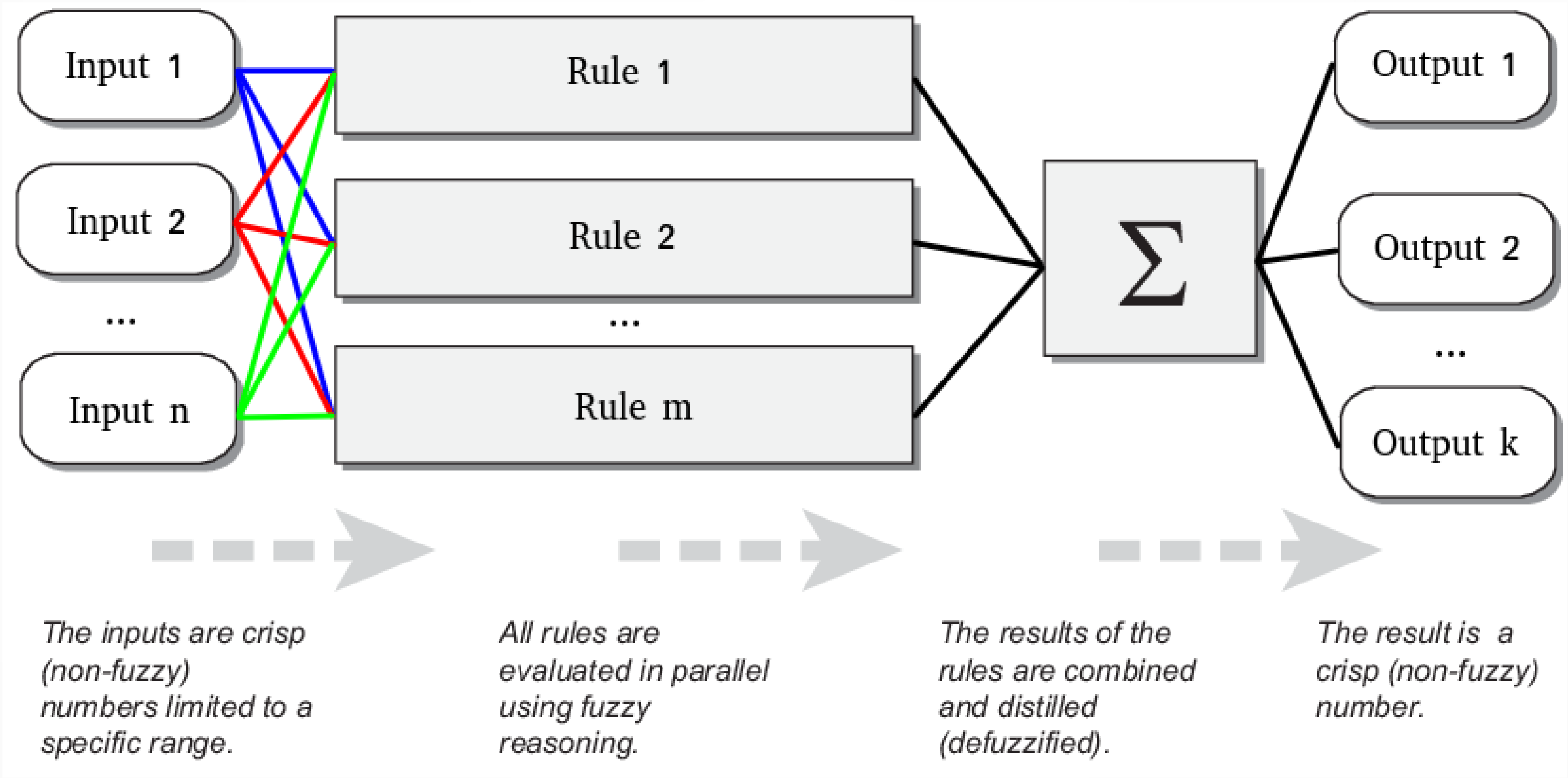
- Summary

1) The fuzzy controller receives inputs (numbers)

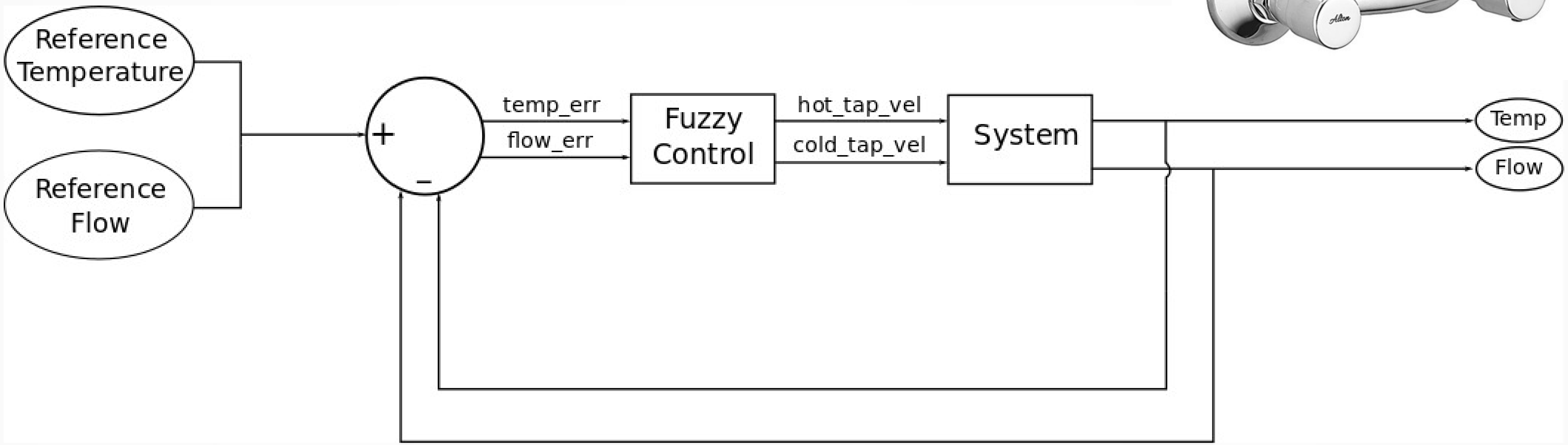
2) Each rule gets activated, one by one, using the And, Or and Implication operators, to get parcels of each output

3) Uses the aggregation operator to form the final fuzzy output

4)Defuzzify each output using the defuzzyfication operator

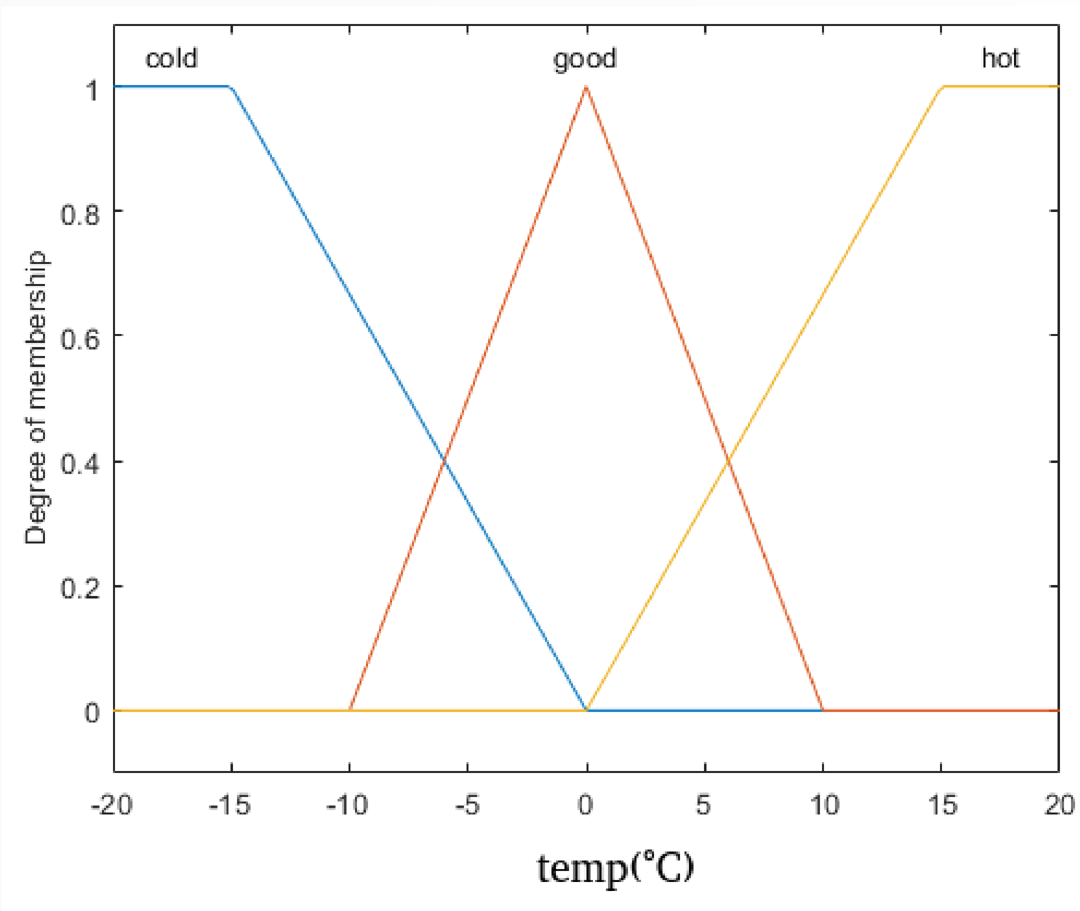


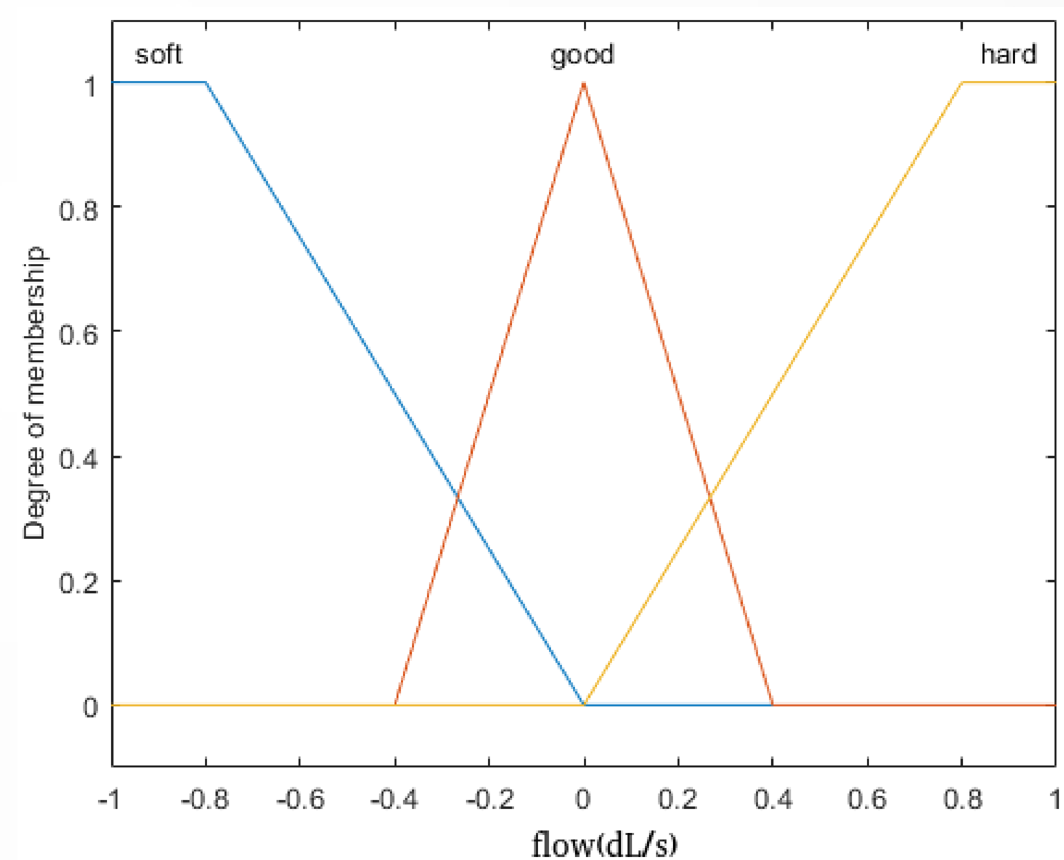
- MIMO system: Controlling the temperature and flow of a tap
- We can use the same classical structure



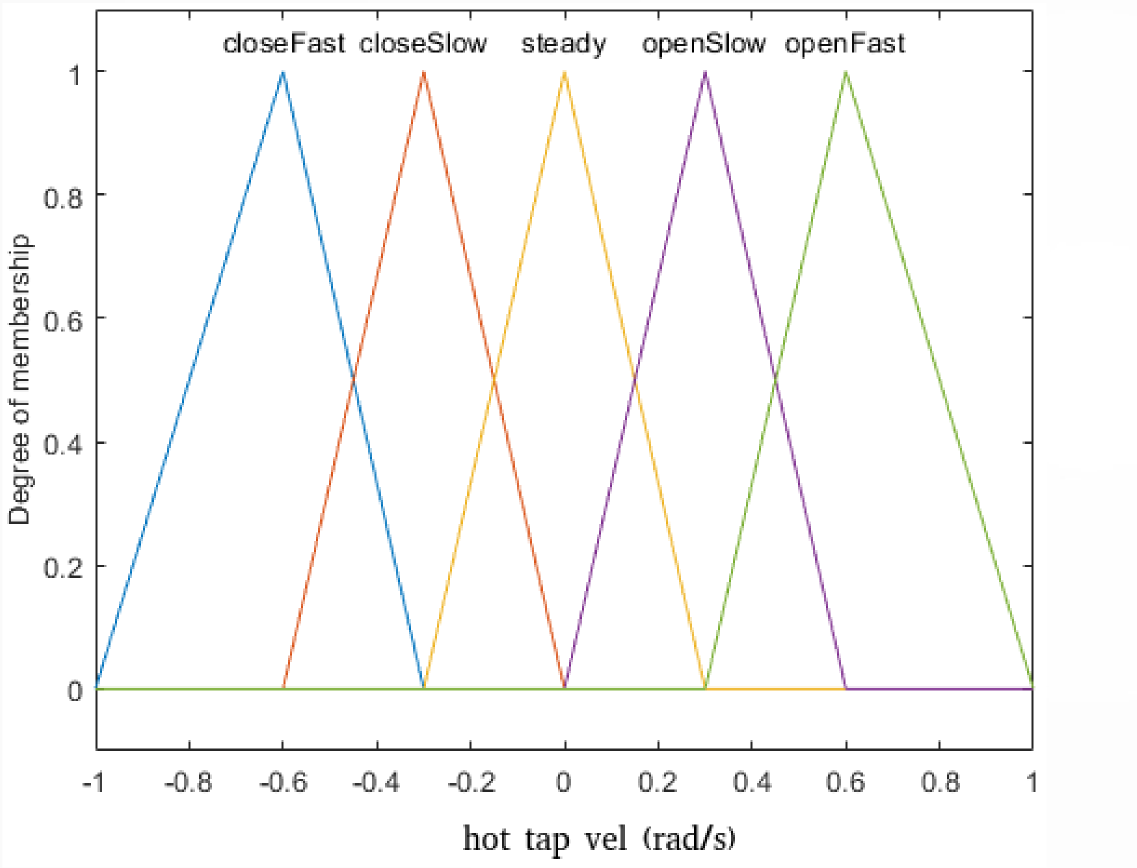
- Just like classical control, it is smart to make the Fuzzy Controller receive the error instead of the reference

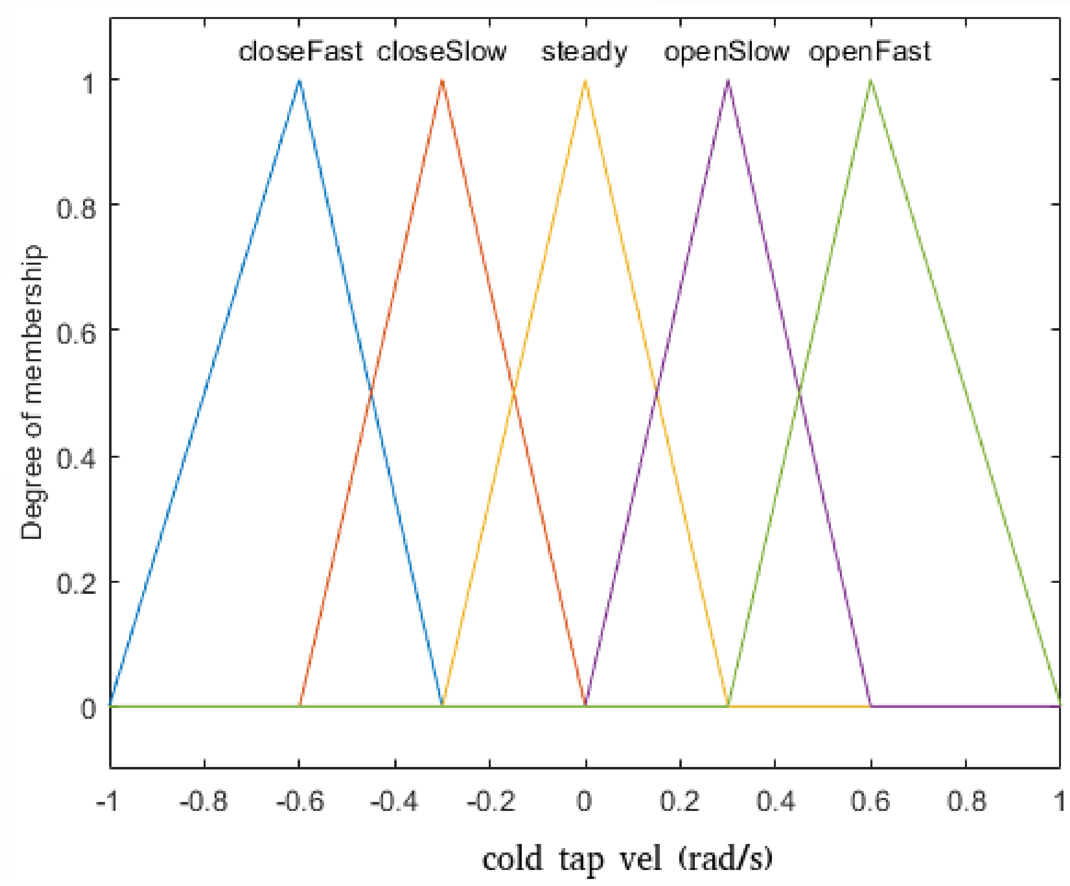
- Firstly, we have to define membership functions to controller inputs
- Remember that the input is the error between temperature and reference





• Defining membership functions to output



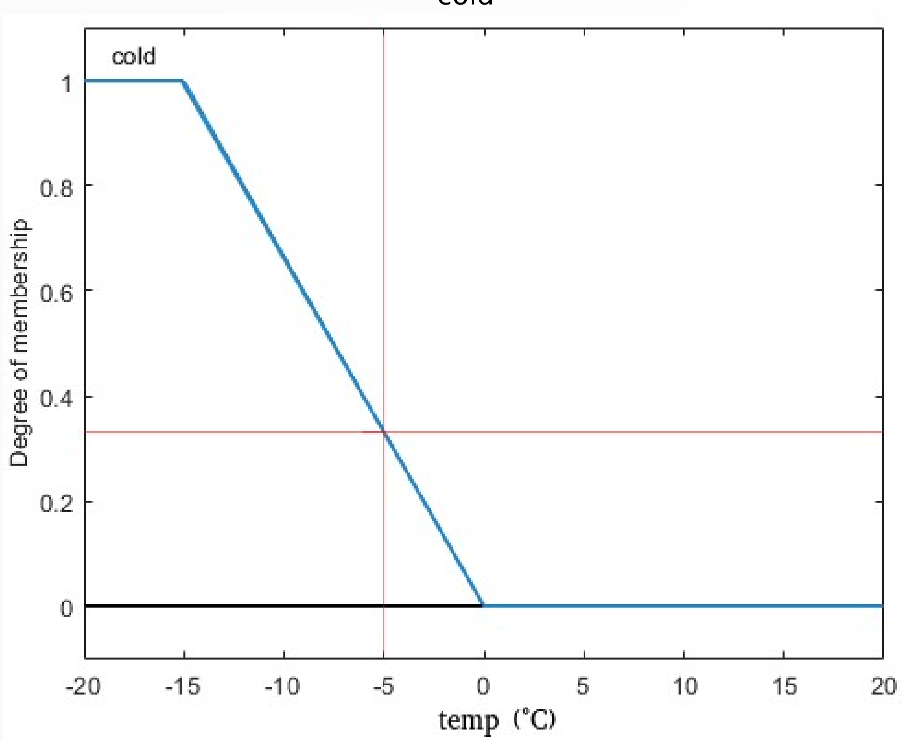


➢ MIMO Fuzzy rule activation

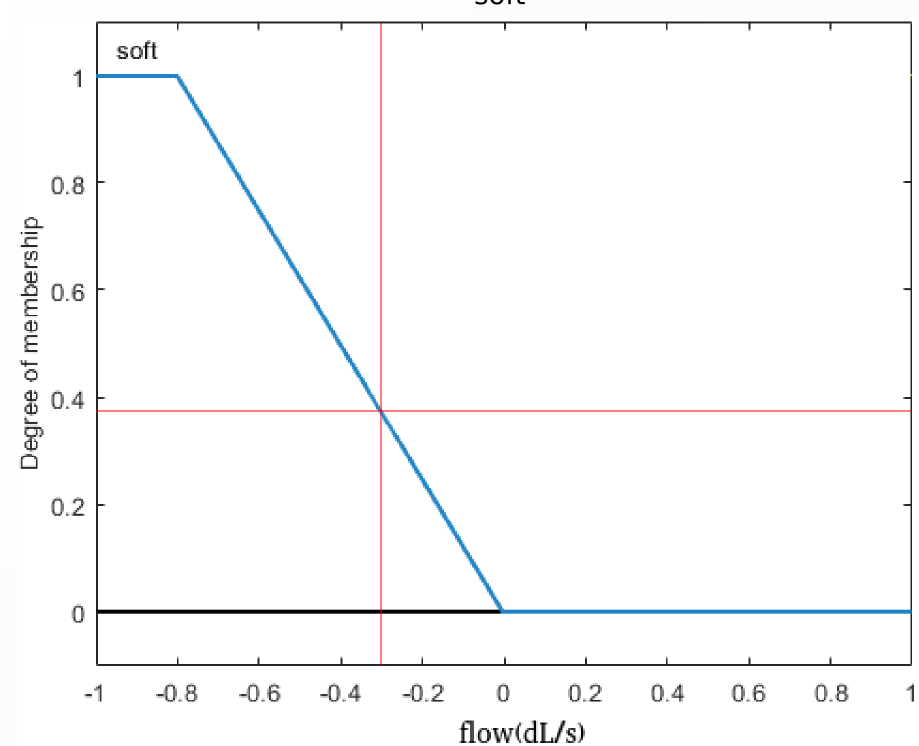
- The **Minimum** rule is still applied, but we have to define **and** / **or** operators

- if "temp" is "cold" **and** "flow" is "soft" **then**
  "hot_tap_vel" is "openFast" and "cold_tap_vel" is "openSlow"

temp = -5 →μ_{cold}(-5) = 0.333



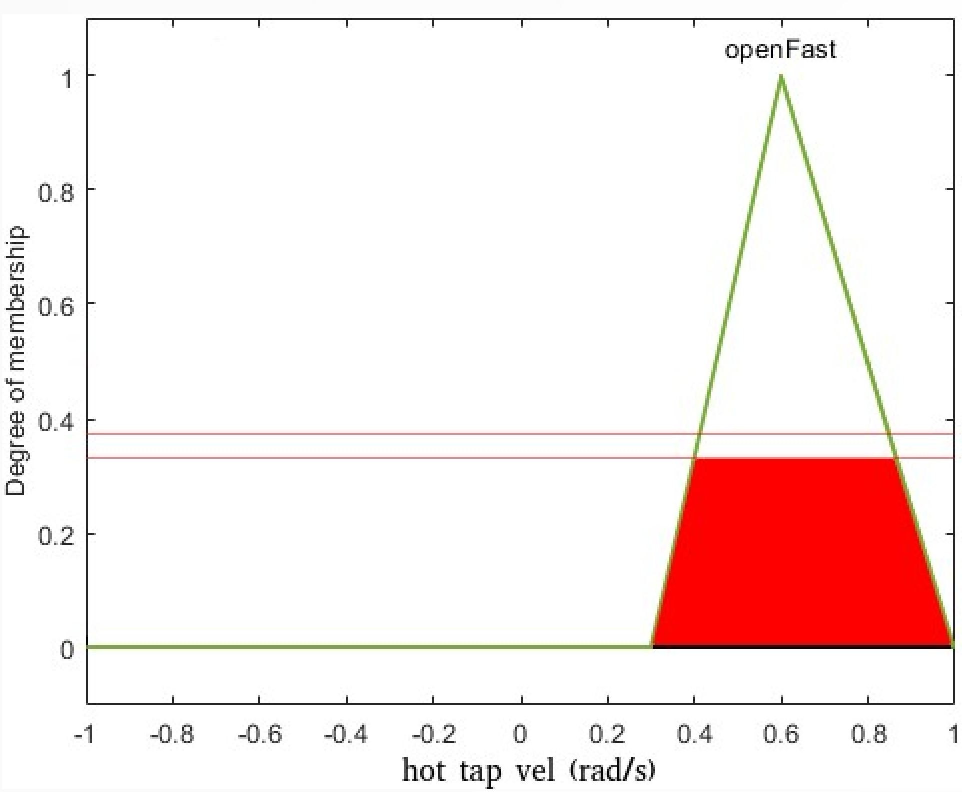
flow = -0.3 →μ_{soft}(-0.3) = 0.375

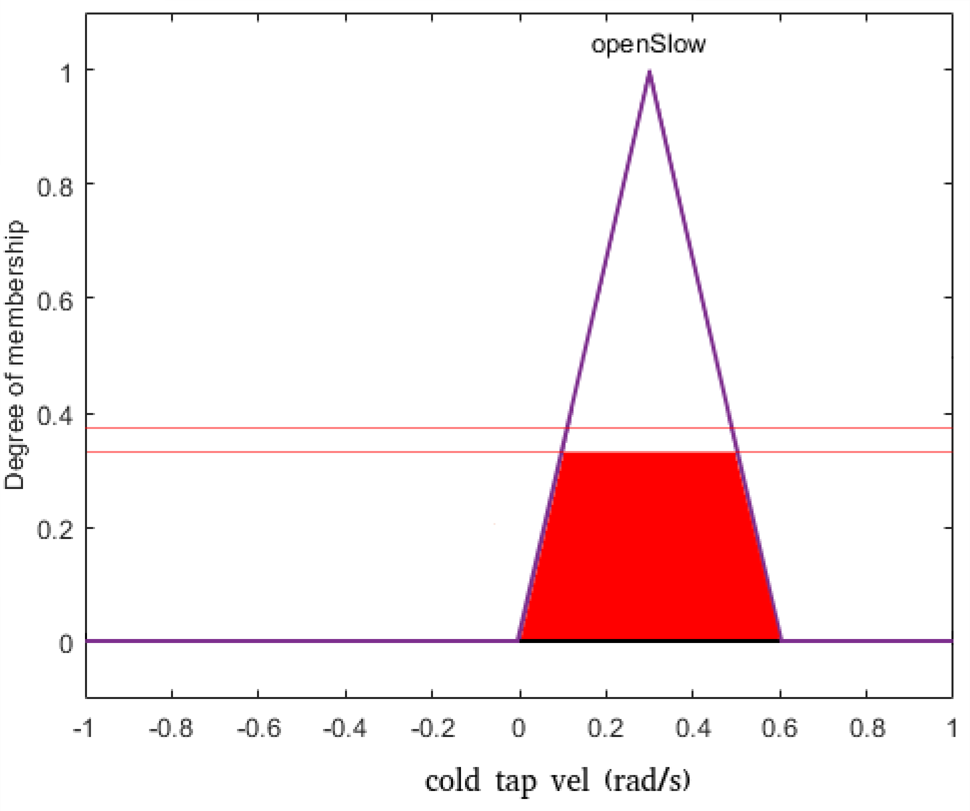


- We can define **and** as the Minimum between the 2, this way "A" **and** "B" = $\mathrm{Min}(\mu_A(x_0), \mu_B(x_1))$

- Then, we proceed just like the SISO system

if "temp" is "cold" **and** "flow" is "soft" **then**
"hot_tap_vel" is "openFast" and "cold_tap_vel" is "openSlow"





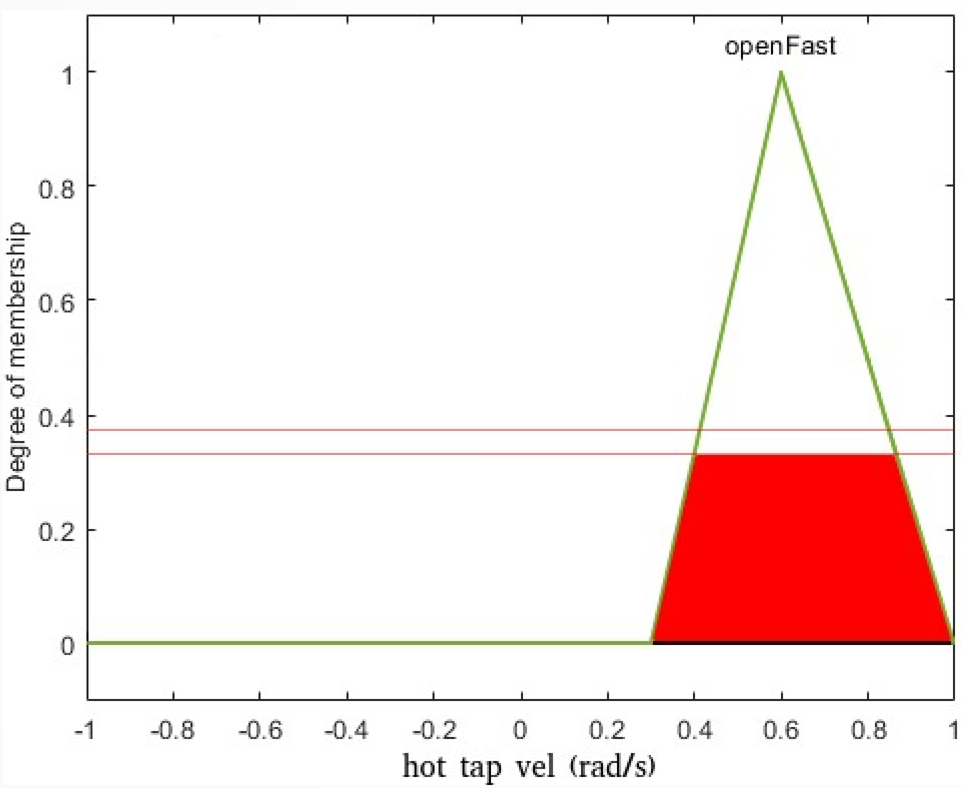
In a same manner, we can define **or** as Maximum

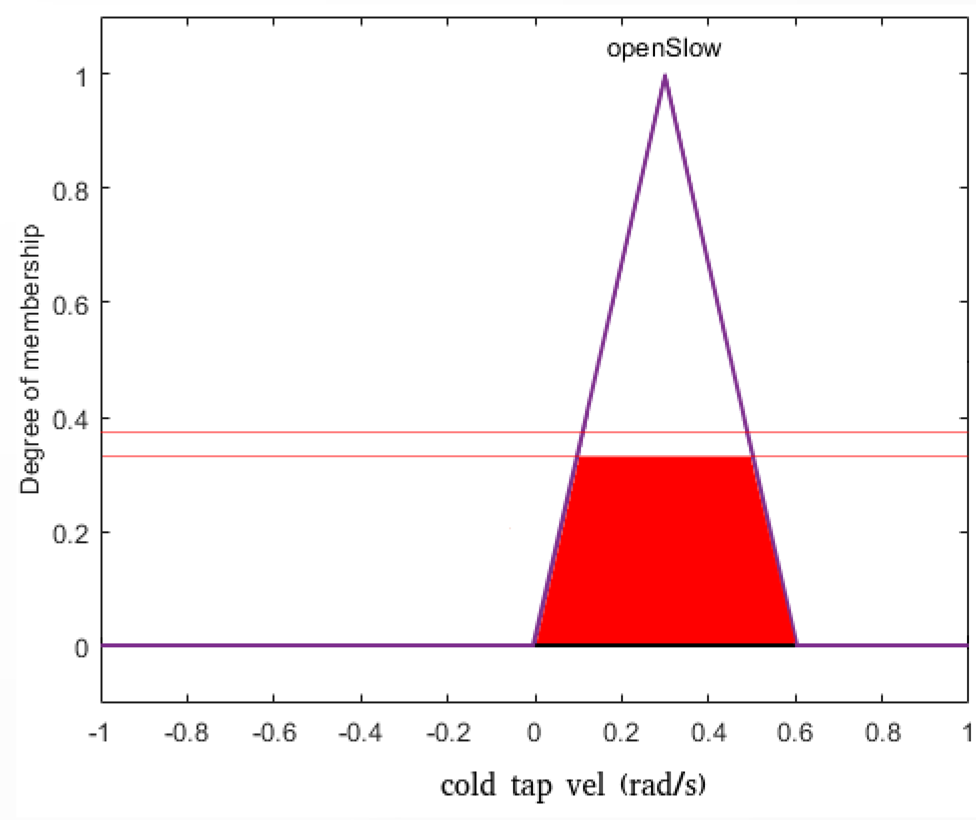
- As for the rules aggregation, it works the same way

if “temp” is “cold” **and** “flow” is “soft” **then**
“hot_tap_vel” is “openFast” and “cold_tap_vel” is “openSlow”

if “temp” is “good” **and** “flow” is “good” **then**
“hot_tap_vel” is “steady” and “cold_tap_vel” is “steady”

- The first rule is activated the same way as previously for temp = -5 and flow = -0.3

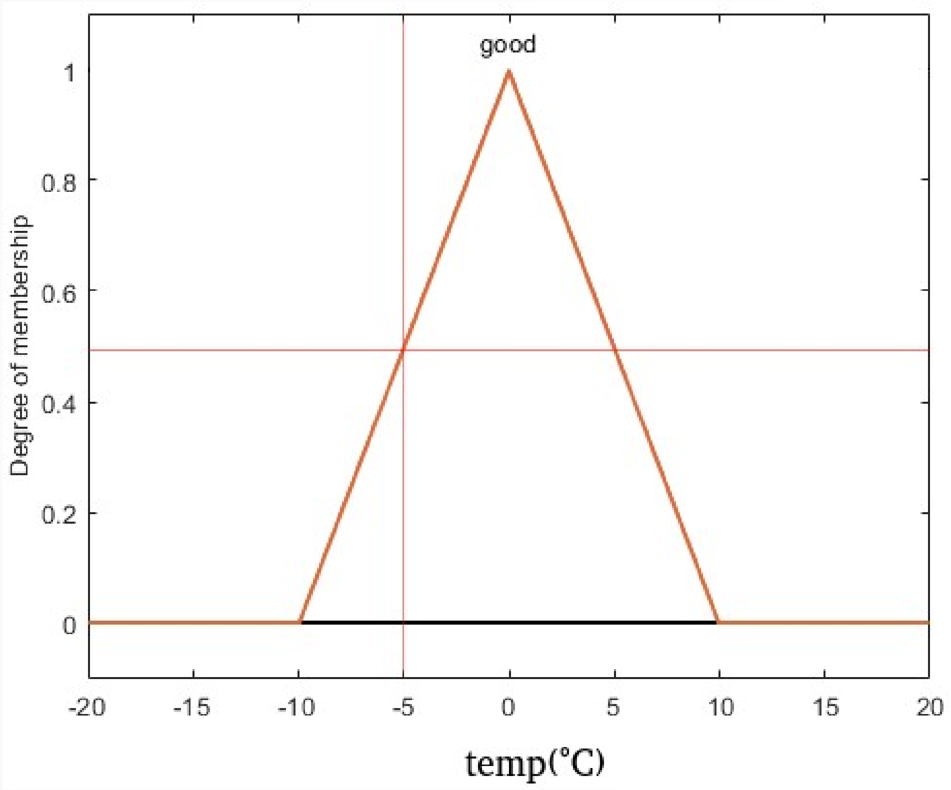




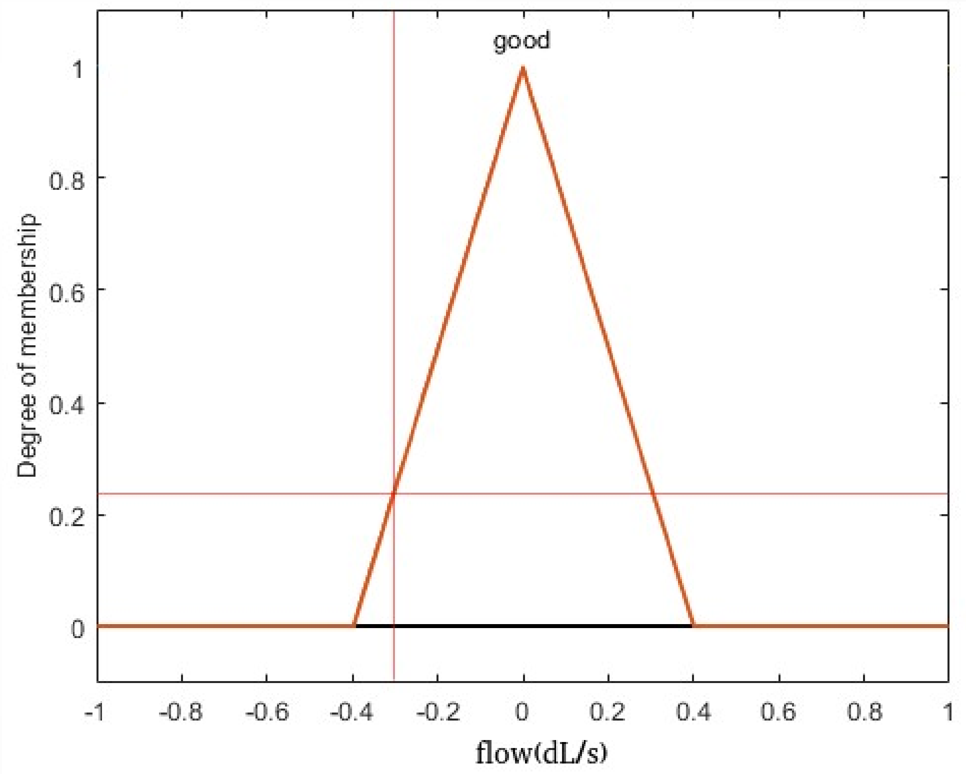
- And for the second rule

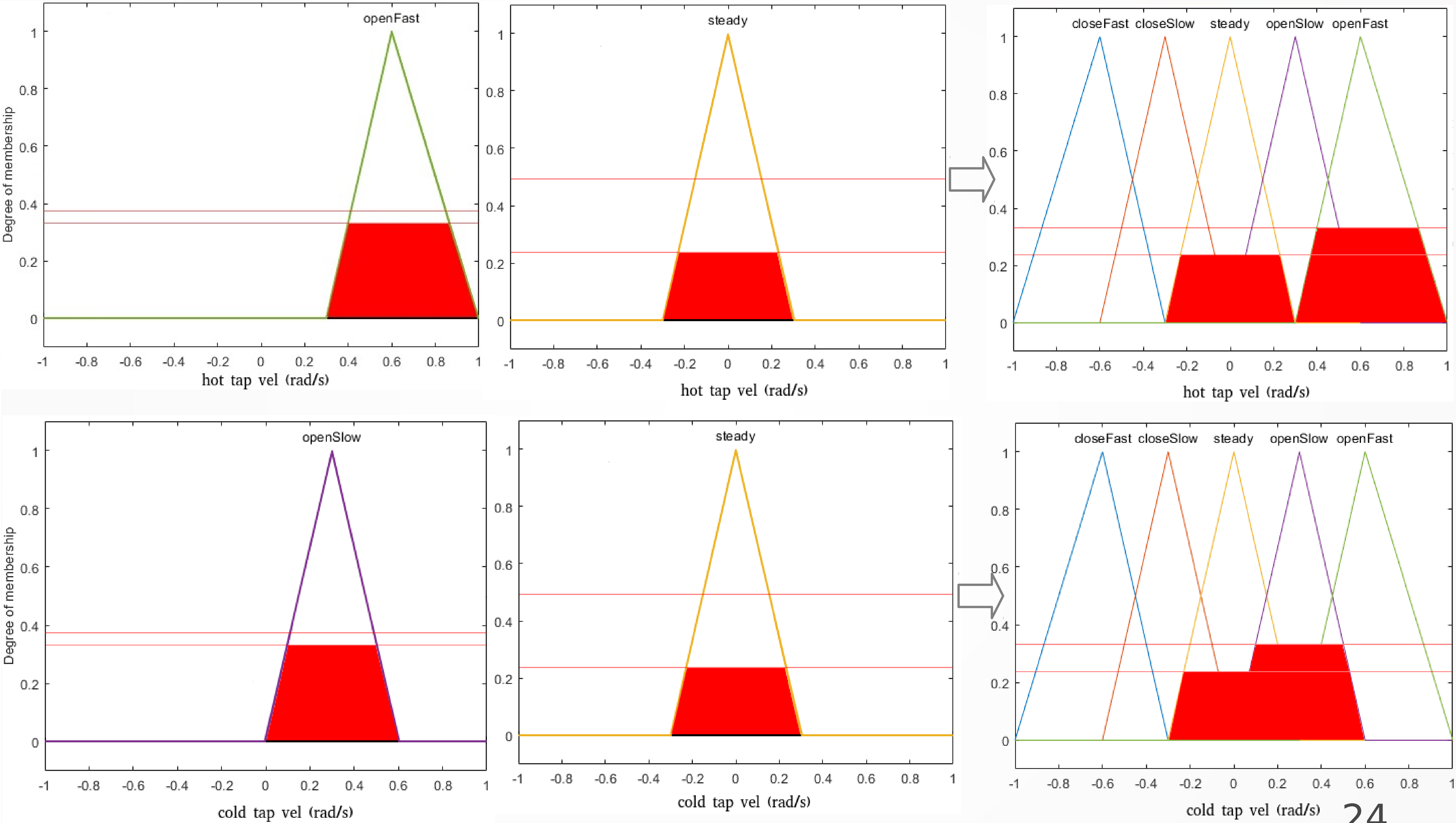
if “temp” is “good” **and** “flow” is “good” **then**
“hot_tap_vel” is “steady” and “cold_tap_vel” is “steady”

temp = -5 →$\mu_{good}(-5) = 0.5$



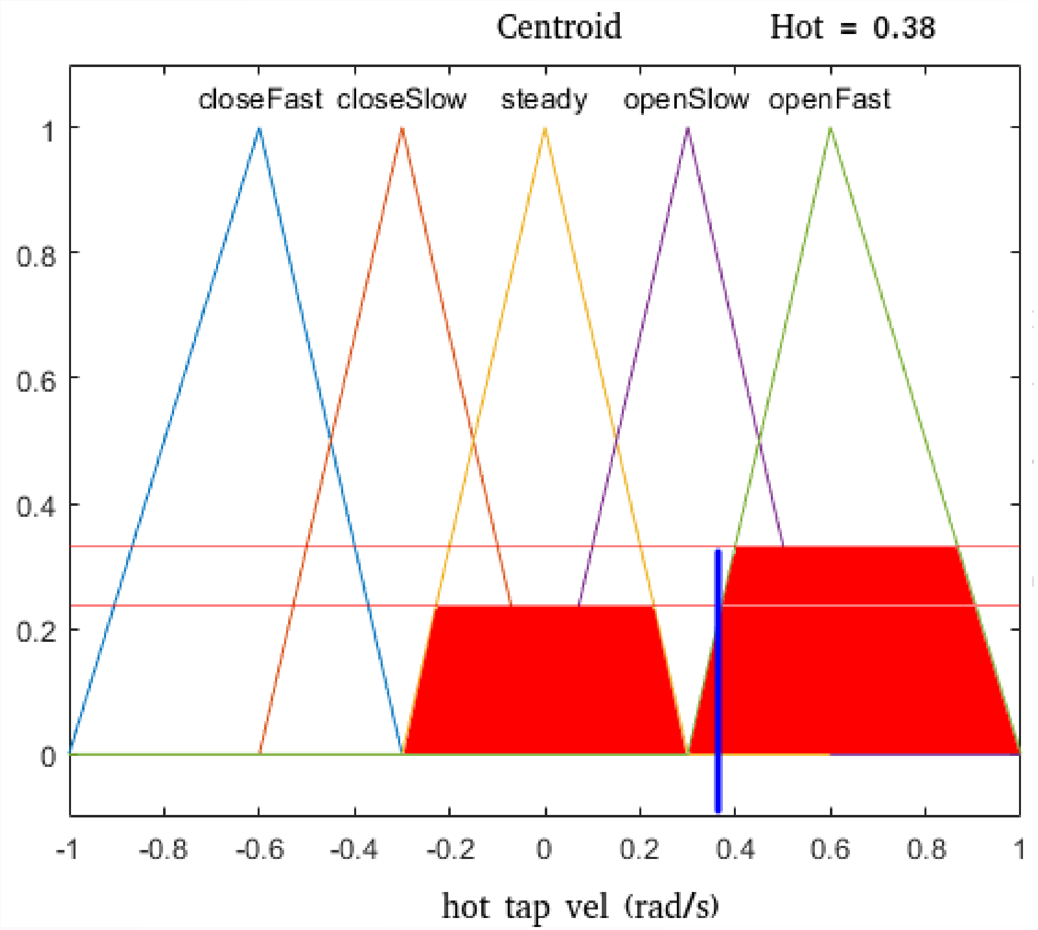
flow = -0.3 →$\mu_{good}(-0.3) = 0.375$

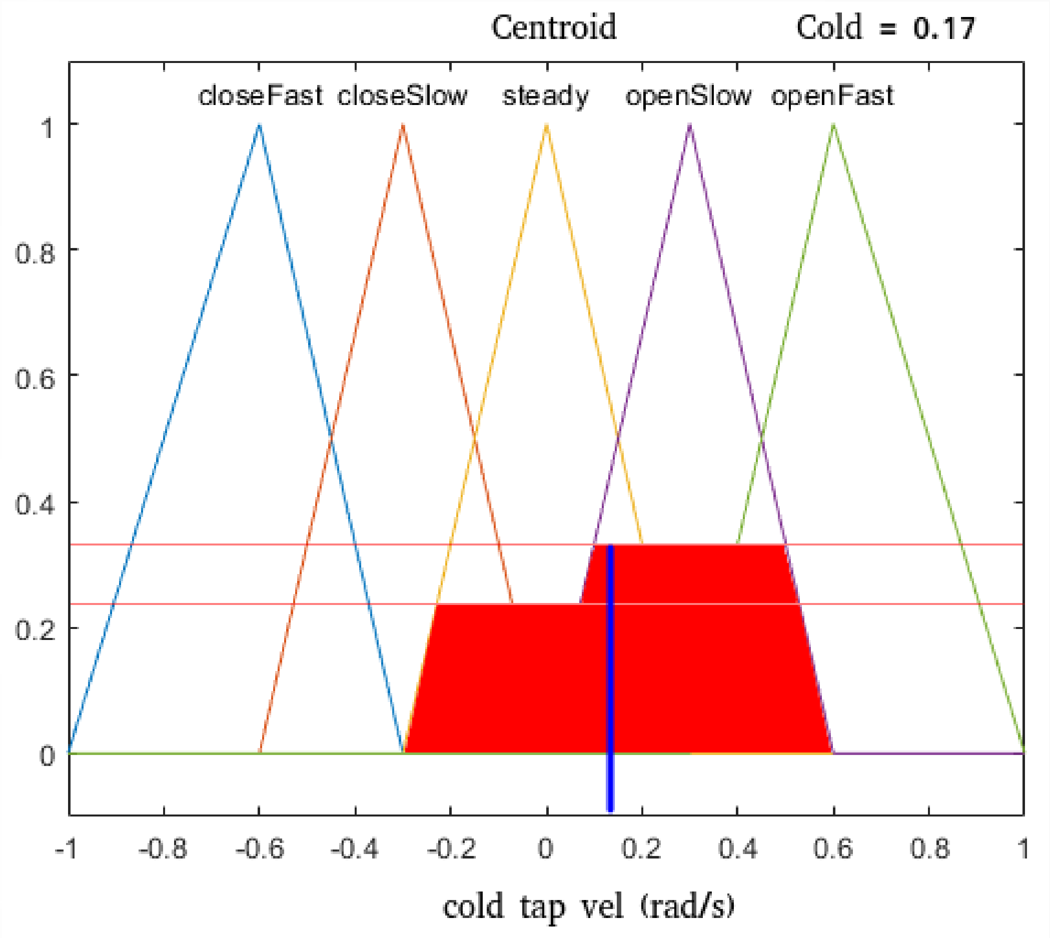




openFast
Degree of membership
hot tap vel (rad/s)
steady
hot tap vel (rad/s)
closeFast closeSlow steady openSlow openFast
hot tap vel (rad/s)
openSlow
Degree of membership
cold tap vel (rad/s)
steady
cold tap vel (rad/s)
closeFast closeSlow steady openSlow openFast
cold tap vel (rad/s)

- Defuzzyfication using Centroid





- Having the fuzzy output to be Hot = 0.38 and Cold = 0.17

- This is one possible approach

- Fuzzy are very customizable: And, Or, Implication, Aggregation and defuzzyfication operators are left to the designer

- Some configurations have special names, for example:

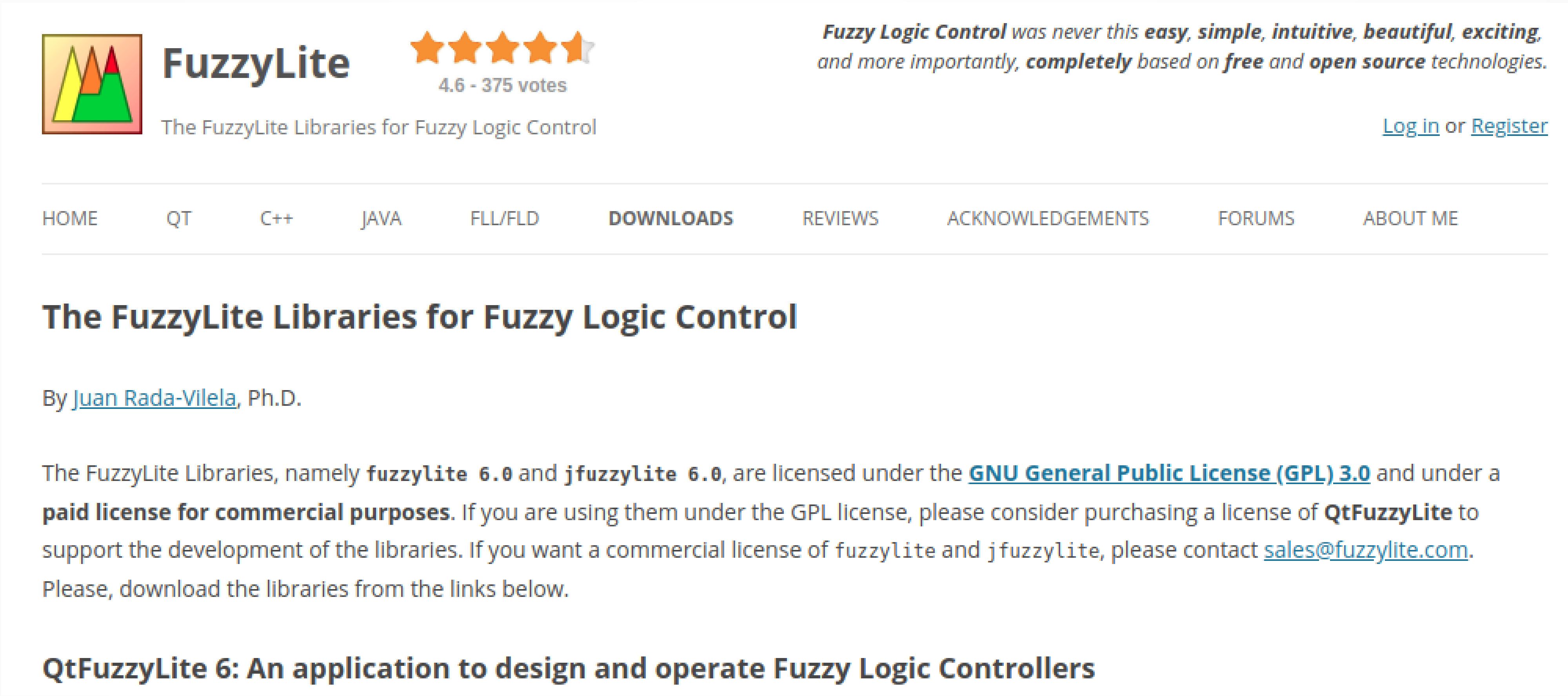
    - **And** = Minimum
    - **Or** = Maximum
    - **Implication =** Product
    - **Aggregation** = Maximum

- This configuration is known as “Mamdani”, and there are a lot more

- Advantages
  - Works surprisingly well
  - Don’t need to know a lot about the system
  - Easy to implement
  - Easy to change
  - MIMO fuzzy systems are as easy as SISO

- Disadvantages:
  - A lot of parameters to consider
  - Hard to optimize
  - No better option than to test and see what works better

- Fuzzylite
  - An open source library in C++
  - Free and easy to use



FuzzyLite

4.6 - 375 votes

The FuzzyLite Libraries for Fuzzy Logic Control

***Fuzzy Logic Control** was never this **easy**, **simple**, **intuitive**, **beautiful**, **exciting**, and more importantly, **completely** based on **free** and **open source** technologies.*

Log in or Register

HOME QT C++ JAVA FLL/FLD **DOWNLOADS** REVIEWS ACKNOWLEDGEMENTS FORUMS ABOUT ME

The FuzzyLite Libraries for Fuzzy Logic Control

By Juan Rada-Vilela, Ph.D.

The FuzzyLite Libraries, namely **`fuzzylite 6.0`** and **`jfuzzylite 6.0`**, are licensed under the **GNU General Public License (GPL) 3.0** and under a **paid license for commercial purposes**. If you are using them under the GPL license, please consider purchasing a license of **QtFuzzyLite** to support the development of the libraries. If you want a commercial license of `fuzzylite` and `jfuzzylite`, please contact sales@fuzzylite.com. Please, download the libraries from the links below.

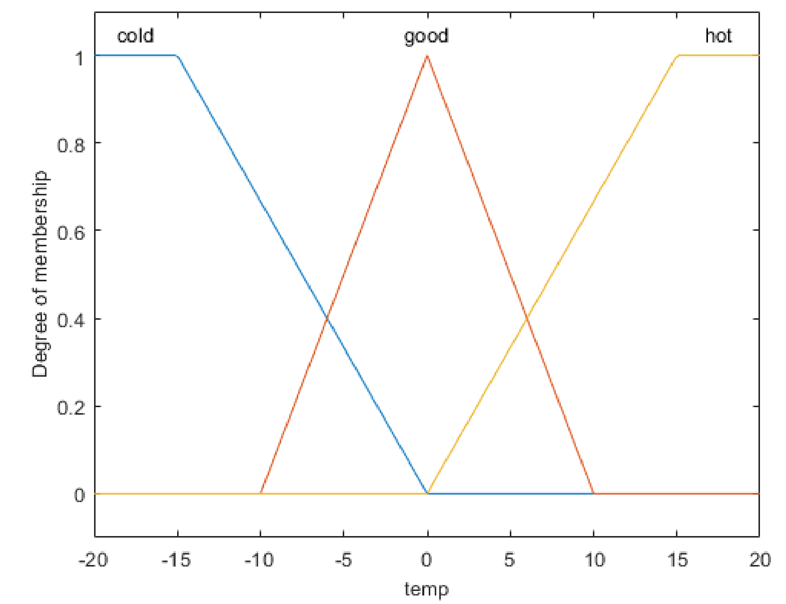
QtFuzzyLite 6: An application to design and operate Fuzzy Logic Controllers

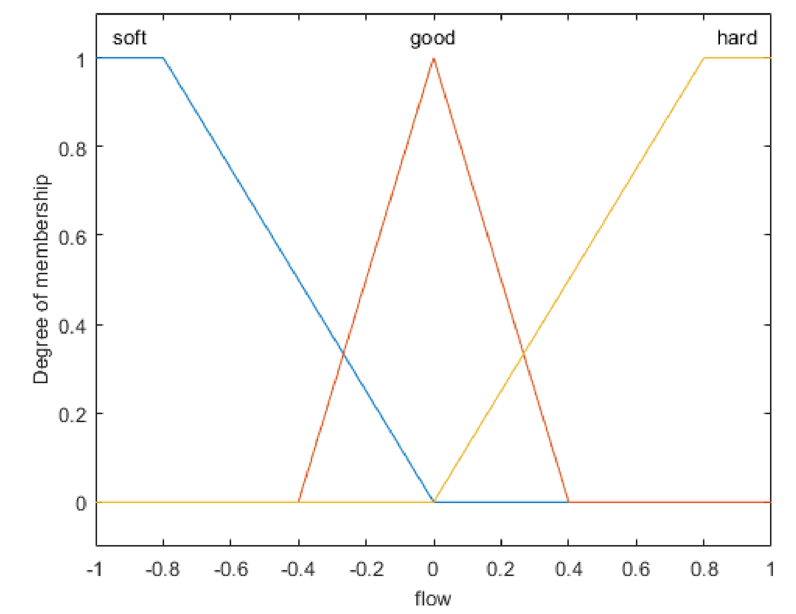
Source: https://fuzzylite.com/downloads/

- After downloading and compiling the project, we can start our project

- Step 1: Initializing Engine and Input variables:

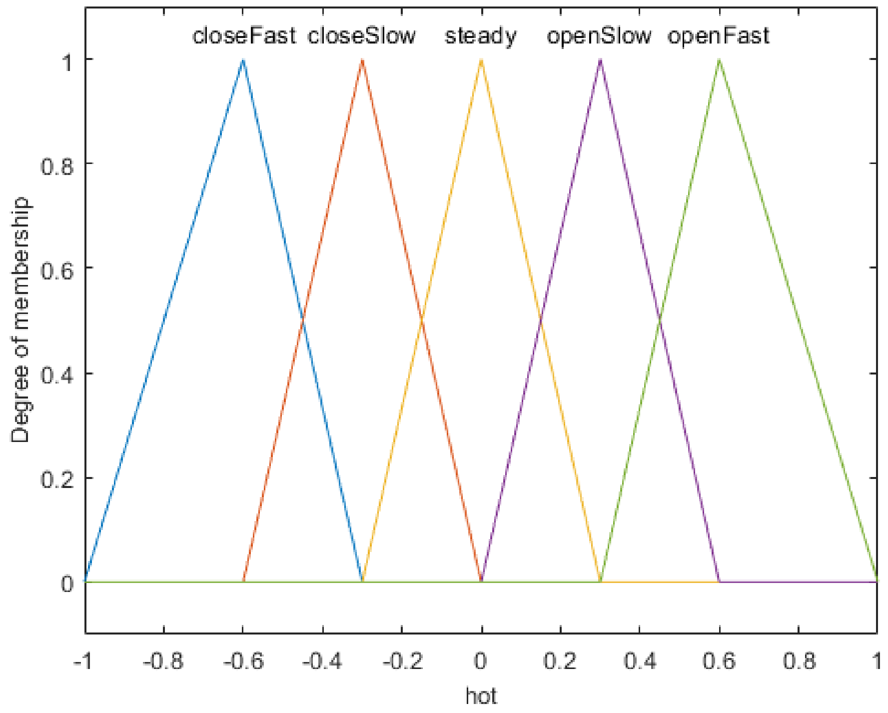
```
Engine* engine = new Engine;
engine->setName("TempControl");
engine->setDescription("");

InputVariable* temp = new InputVariable;
temp->setName("temp");
temp->setDescription("");
temp->setEnabled(true);
temp->setRange(-20.000, 20.000);
temp->setLockValueInRange(false);
temp->addTerm(new Ramp("cold", 0.000, -15.000));
temp->addTerm(new Triangle("good", -10.000, 0.000, 10.000));
temp->addTerm(new Ramp("hot", 0.000, 15.000));
engine->addInputVariable(temp);

InputVariable* flow = new InputVariable;
flow->setName("flow");
flow->setDescription("");
flow->setEnabled(true);
flow->setRange(-1.000, 1.000);
flow->setLockValueInRange(false);
flow->addTerm(new Ramp("soft", 0.000, -0.800));
flow->addTerm(new Triangle("good", -0.400, 0.000, 0.400));
flow->addTerm(new Ramp("hard", 0.000, 0.800));
engine->addInputVariable(flow);
```





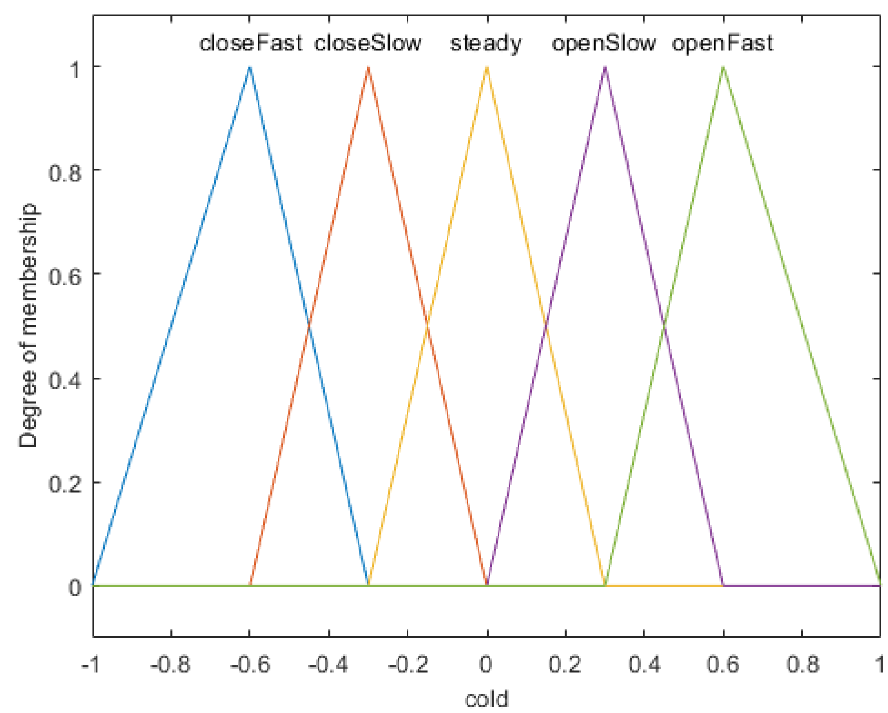
- Step 2: Defining outputs each aggregation and defuzzification operator

```
OutputVariable* hot = new OutputVariable;
hot->setName("hot");
hot→setDescription("");

hot->setEnabled(true);
hot->setRange(-1.000, 1.000);
hot->setLockValueInRange(false);

hot->setAggregation(new Maximum);
hot->setDefuzzifier(new Centroid(100));
hot->setDefaultValue(fl::nan);
hot->setLockPreviousValue(false);

hot->addTerm(new Triangle("closeFast", -1.000, -0.600, -0.300));
hot->addTerm(new Triangle("closeSlow", -0.600, -0.300, 0.000));
hot->addTerm(new Triangle("steady", -0.300, 0.000, 0.300));
hot->addTerm(new Triangle("openSlow", 0.000, 0.300, 0.600));
hot->addTerm(new Triangle("openFast", 0.300, 0.600, 1.000));

engine->addOutputVariable(hot);
```



- Step 2.2: Second output

```
OutputVariable* cold = new OutputVariable;
cold->setName("cold");
cold→setDescription("");

cold->setEnabled(true);
cold->setRange(-1.000, 1.000);
cold->setLockValueInRange(false);

cold->setAggregation(new Maximum);
cold->setDefuzzifier(new Centroid(100));
cold->setDefaultValue(fl::nan);
cold->setLockPreviousValue(false);

cold->addTerm(new Triangle("closeFast", -1.000, -0.600, -0.300));
cold->addTerm(new Triangle("closeSlow", -0.600, -0.300, 0.000));
cold->addTerm(new Triangle("steady", -0.300, 0.000, 0.300));
cold->addTerm(new Triangle("openSlow", 0.000, 0.300, 0.600));
cold->addTerm(new Triangle("openFast", 0.300, 0.600, 1.000));

engine->addOutputVariable(cold);
```



- Step 3: Defining rules, And, Or and implication operators (mamdani configuration)

```
RuleBlock* mamdani = new RuleBlock;
mamdani->setName("mamdani");
mamdani->setDescription("");
mamdani->setEnabled(true);

mamdani->setConjunction(new Minimum);
mamdani->setDisjunction(new Maximum);
mamdani->setImplication(new AlgebraicProduct);
mamdani->setActivation(new General);

mamdani->addRule(Rule::parse("if temp is cold and flow is soft then cold is openSlow and hot is openFast", engine));
mamdani->addRule(Rule::parse("if temp is cold and flow is good then cold is closeSlow and hot is openSlow", engine));
mamdani->addRule(Rule::parse("if temp is cold and flow is hard then cold is closeFast and hot is closeSlow", engine));

mamdani->addRule(Rule::parse("if temp is good and flow is soft then cold is openSlow and hot is openSlow", engine));
mamdani->addRule(Rule::parse("if temp is good and flow is good then cold is steady and hot is steady", engine));
mamdani->addRule(Rule::parse("if temp is good and flow is hard then cold is closeSlow and hot is closeSlow", engine));

mamdani->addRule(Rule::parse("if temp is hot and flow is soft then cold is openFast and hot is openSlow", engine));
mamdani->addRule(Rule::parse("if temp is hot and flow is good then cold is openSlow and hot is closeSlow", engine));
mamdani->addRule(Rule::parse("if temp is hot and flow is hard then cold is closeSlow and hot is closeFast", engine));

engine->addRuleBlock(mamdani);
```

- Step 4: Setting inputs and calculating outputs

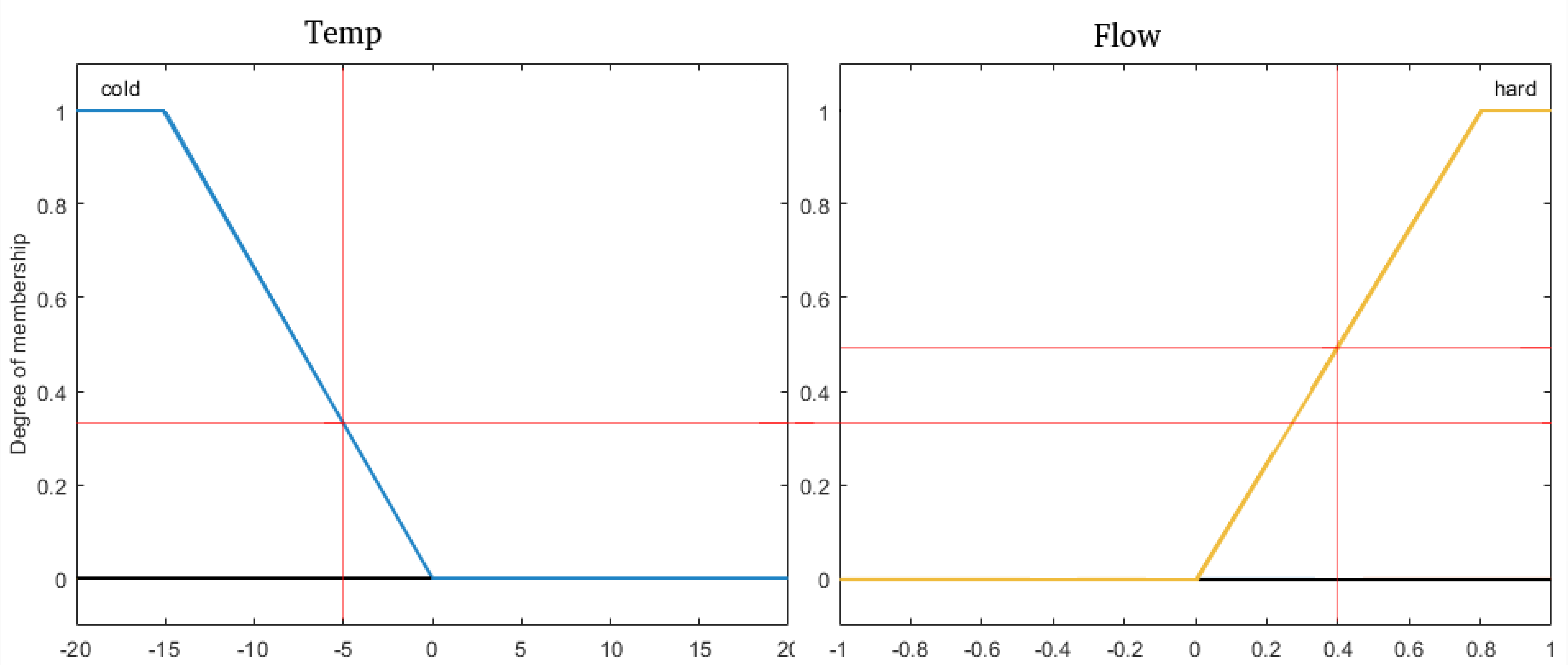
```
temp->setValue(-5);
flow->setValue(-0.3);

engine->process();

cout << "Hot valve value: " << hot->getValue() << endl;
cout << "Cold valve value: " << cold->getValue() << endl;
```

- That’s all !
- For further details: https://fuzzylite.com/cpp/
- Documentation: https://fuzzylite.github.io/fuzzylite/

- And for a two input, two output rule

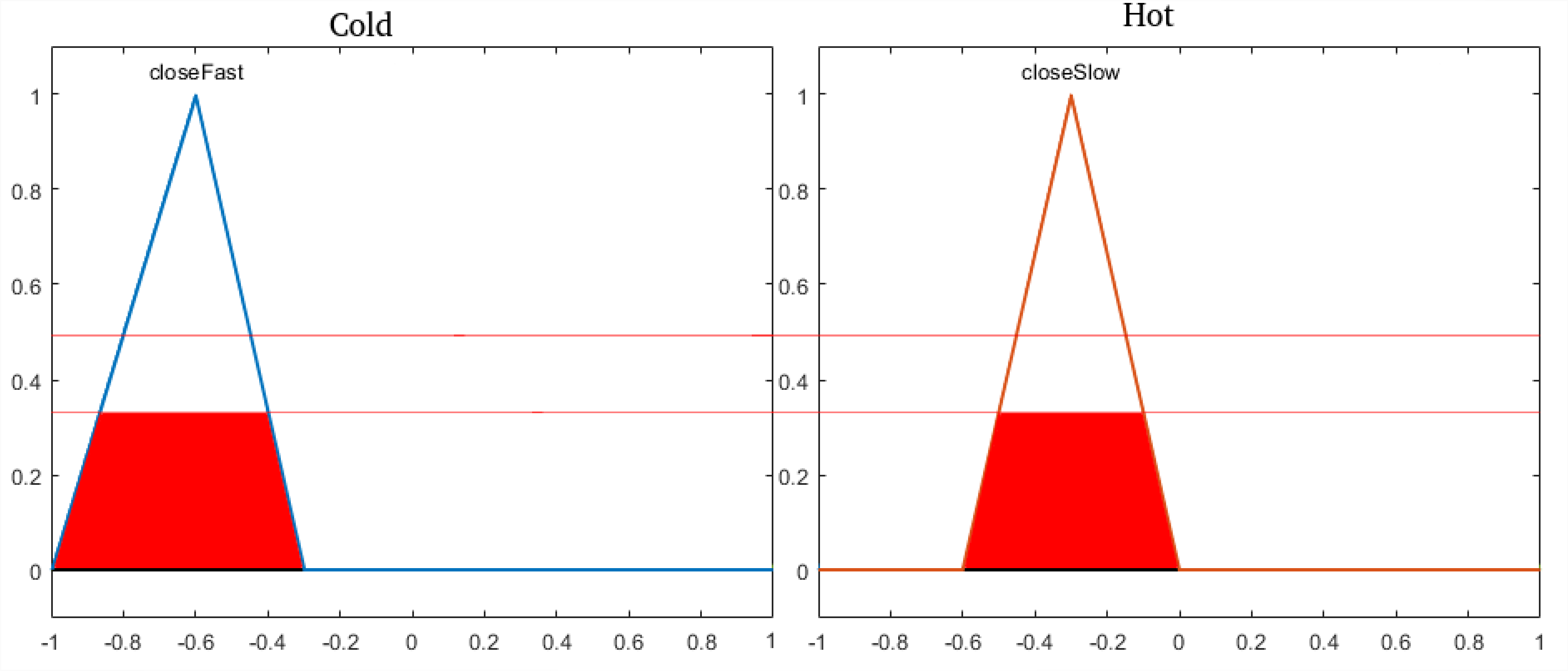
  if “temp” is “cold” **and** “flow” is “hard” then “cold” is “closeFast” and “hot” is “closeSlow”

  - temp = -5 → $\mu_{cold}(-5) = 0.333$
  - flow = 0.4 → $\mu_{hard}(0.4) = 0.5$



And usually calculates the Minimum **Or usually** calculates the Maximum

• Thus, the output for this specific rule is



• The number of outputs don't change anything!

• The next step is to unify all the output information from the many rules