Fuzzy Laboratory 1

Fuzzy Logic Enhanced Time Petri Nets (FLETPN) Model

1. Laboratory Objectives

* Acquiring the concepts
  + Fuzzy logic,
  + Fuzzyfication and defuzzyfication values,
  + Determining the fuzzy logic rules set FLRS,
  + FLETPN model,
* Developments and tests

1. Theoretical considerations

As is well known, the human being works according to some rules that is formulated through education and instinct, the use of concepts and measures is not always accurate but rather "more flexible" in quantification. These aspects led to fuzzy logic, theorized by Lotfi A. Zadeh in 1965. Fuzzy logic uses special variables, "fuzzy" values, thus forming a workspace that is different from the one characterized by the exact numerical values.

Fuzzy variables represent a class or set of *xi* values, which are contained in a certain proportion in a set, the proportion is given by the function *µi (xi)*. The goal is to reduce the complexity of the outside world to a finite number of variables that can be intuitively operated. The way to work with these concepts or variables is in the form of rules of logical implication:

**IF** (*input* ***IS*** *{fuzzy\_set1}*) **THEN** (*action* ***IS*** *{fuzzy\_set2}*),

where "fuzzy\_set1" is the set of values that "input" can take, and "fuzzy\_set2", the set of possible values for "action".

**Example:**

**IF** (*x1* is *A1* *˄* *x2* is *A2* *˄* … *˄* *xm* is *Am* ) **THEN** (*y1* is *A1* *˄* *y2* is *A2* *˄* … *˄* *yn* is *An*),

where *xi ϵ fuzzy\_set1* and *yi ϵ fuzzy\_set2*.

Fuzzy values in “fuzzy\_set” sets can be "LOW", "MEDIUM", "EXTREME", "HOT", "WARM", "CLOSE", "FAR" etc.

Next, we will use the fuzzy set FS = {NL, NM, ZR, PM, PL} with Negative Long, Medium Negative, Zero, Positive Medium and Positive Long for fuzzy\_set1 and fuzzy\_set2.

The operation that converts a real-world variable into a fuzzy variable is called **fuzzyfication** and is performed according to a membership function, see Fig. 2.1.



Figure 2.1 Fuzzyfication function of variables

**Example:**

For a variable x‘, the ZR membership function: *μZR(*x‘*)*= 0.65 and for the membership of PM: *μPM* *(x‘)* = 0.35. Observe that: *μZR(*x‘*)* + *μPM* *(x‘)*= 1.

The inverse operation that converst a fuzzy variable to a numeric value is called **defuzzyfication** and is performed similarly to fuzzyfication by calculating the position of the variable on the abscissa axis, depending on the degrees of the membership of each of the fuzzy values indicated. There exist several possible methods, the center of gravity method is usually used:



where *xi* are the values of the fuzzy levels in the universe of the variable *x*, *μi*(*xi*) is the degree of the fuzzy variable membership at each of the fuzzy levels, and *x* is the crisp result of the defuzzy operation.

A rule can be met in a higher or lower proportion as its condition is met. Thus, several rules can be fulfilled at a given time, and so the value of the command values ("action") takes several fuzzy values in varying proportions.

Figure 2.2 presents the architecture of a control system that uses fuzzy logic. Inputs values that are measured by the sensors are fuzzy, the fuzzy logic control algorithm applied, then the output values are defuzzyfied and applied to the system, then the errors are measured and the cycle resumes.



Figure 2.2 The architecture of a fuzzy logic control system

1. Fuzzy Logic Enhanced Time Petri Nets

Figure 3.1 is a simple component with the Fuzzy Logic Enhanced Time Petri Nets (FLETPN) with two inputs and two outputs. The domains *x1, x2, x3, x4 ϵ [-1,1],* and we consider the amplification factors *w1, w2 ϵ [-10, 10]*. The amplification is used so that the resulting values *x’1* și *x’2* are within an acceptable range.

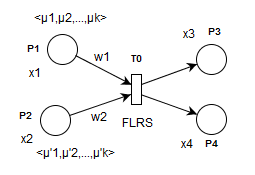


Figure 3.1 FLETPN example



Figura 3.2 The FLETPN model

The model is capable to describe the synchronous and asynchronous reaction of a system to continuous and discreet changes. A place may contain a token that represents a synchronous or asynchronous reaction respectively to a continuous or a discreet change.

Each place has a variable *xi* assigned. A token set in a place *pi*  represents the degrees of a variable’s membership *µi (xi)* to the fuzzy set FS. For a token in our example we have *n=5: <μNL, μNM, μZR μPM, μPL>.* Each arc has a *wi* coefficient.

A transition is executable if and only if there is at least one rule in the corresponding table that can be enabled. If multiple rules can be enabled at any given time, all rules are used, the resulting tokens being the sum of the tokens. A normalization procedure is used so that the output location contains a token that meets the condition *μNL + μNM + μZR + μPM + μPL = 1.*

Execution of an executable transition *ti* implies:

* + - extracting tokens from the input places, marked *otj*;
    - defuzzyfication of all the input variables *xi*;
    - multiplying the variables with the corresponding coefficient *wi*;
    - fuzzyfication of the varibles *xij*;
    - using FLRS with the input variables *xij ;*
    - normalization of the operation that reduces the consequences to one and leads to the injection of a single token in the output places;
    - Injection of the resulting tokens into the output places, marked *toj;*

An example of the solutions that are calculated for two inputs and two outputs is given in Table 1:

Tabel 1: FLRS Example.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x1/x2 | NL | NM | ZR | PM | PL |
| NL | PL,PL | PM,NL | ZR,ZR | NL,PL | ZR,PL |
| NM | PL,PM | NM,ZR | PL,NM | PL,NM | NM,PL |
| ZR | NL,PM | PL,NM | ZR,ZR | ZR,NM | PL,ZR |
| PM | ZR,PL | ZR,PM | NM,PM | PM,PM | NL,NL |
| PL | PM,ZR | PM,NM | ZR,ZR | NM,ZR | PL,NM |

1. Fuzzyfication and defuzzification the values

Example:

Two input values are given *x1*=0.3 and *x2*=0.7 ϵ [-1, 1] and the coefficients *w1*=2 şi *w2*=3.

Required: Calculate outputs *x3* and *x4*.

We multiply each variable by the appropriate coefficient:

*x’1=x1\*w1=0.3\*2=0.6* and

*x’2=x2\*w2=0.7\*3=2.1=>x’2=1*

Fuzzyifying the two values with the function in Figure 2.1, we have for *x’1:*

*μPL=(0.6-0.5)\*1/0.5=0.2 and μPM=1\*(0.5-0.1)/0.5=0.8* so the place *p1* will contain the values:

*μ x’1=<0, 0, 0, 0.8, 0.2>*

And for *x’2* and the place *p2*:

*μ x’2=<0, 0, 0, 0, 1>.*

Considering the FLRS table in the above example, the rules are enabled:

*r1: <x3,x4>* *= <NL,NL>* and *r2:<x3,x4>* *= <PL,NM >* .

For the defuzzyfication we have:

The power of the rule and, if we have a conjunction, it is

*si = μ1 ∙ μ2* or *si = min*{ *μ1, μ2*}.

In our case *s1*=0.8\*1=0.8 and *s2*=0.2\*1=0.2.

*x3*=(NL\*0.8+PL\*0.2)/(0.8+0.2)=((-1)\*0.8+(1)\*0.2)/1=(-0.8+0.2)/1= -0.6.

*x4*=(NL\*0.8+NM\*0.2)/(0.8+0.2)=((-1)\*0.8+(-0.5)\*0.2)/1=(-0.8-0.1)/1= -0.9.

**Exercise:**

Let’s fuzzyficate the values *x1,x2=± No / 15* where *No* = the student's order number in the group. Randomly choose a FLRS table and the values *w1* and *w2* by each student.

  Calculate the outputs *x3* and *x4*.

1. Develop and Test

5.1 Application 1

**Specifications:** Build an application is required to be constructed and build an inverter for the input values *x1, x2, w1, w2* and calculate the output values *x3* and *x4*.

In the mathematical model for the inverter, we consider:

(x1, x2) 🡺 (x3, x4) x3 = - x1; x4 = - x2; w1 = w2 = 1:

x3 = - w1∙x1 ↔ w1∙x1 ϵ [-1,1]

x3 = 1; ↔ w1∙x1 < -1;

x3 = -1; ↔ w1∙x1 > 1;

x4 = - w2∙x2 ↔ w2∙x2 ϵ [-1,1]

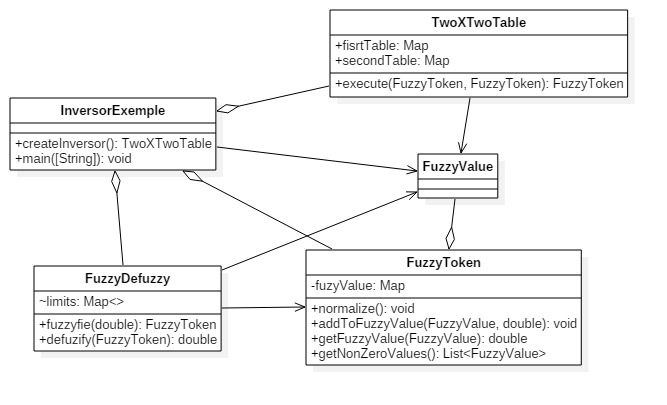
x4 = 1; ↔ w2∙x2 < -1;

x4 = -1; ↔ w2∙x2 > 1;

and the FLRS Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x1 \ x2** | **NL** | **NM** | **ZR** | **PM** | **PL** |
| **NL** | PL, PL | PL, PM | PL, ZR | PL, NM | PL, NL |
| **NM** | PM, PL | PM, PM | PM, ZR | PM, NM | PM, NL |
| **ZR** | ZR, PL | ZR, PM | ZR, ZR | ZR, NM | ZR, NL |
| **PM** | NM, PL | NM, PM | NM, ZR | NM, NM | NM, NL |
| **PL** | NL, PL | NL, PM | NL, ZR | NL, NM | NL, NL |

**The class diagram**

****

**Implementation**. The InversorExemple class has the main() method that gives values to the variables *x1, x2, w1, w2*. With the *fuzzyfie*() method of the *FuzzyDefuzzy* class, a token that is created with the *FuzzyToken* class for *x’1*  and one for *x’2* is instantiated. The *execute*() method of the *TwoXTwoTable* class is called to calculate the outputs. The *FuzzyValue* class contains the values used for *fuzzyfication*. The *FuzzyDefuzzy* class contains the *fuzzyfie*() method that performs the fuzzyfication of a double variable and the *defuzzyfy*() method that performs defuzzyfication. Here is the application.

**import** java.util.EnumMap;

**import** java.util.Map;

**public** **class** InversorExemple {

**public** **static** TwoXTwoTable createInversor() {

//construct tabela1 FLRS for inversor, first output

Map<FuzzyValue, Map<FuzzyValue, FuzzyValue>> ruleTable1 =

**new** EnumMap<>(FuzzyValue.**class**);

Map<FuzzyValue, FuzzyValue> nlLine =

**new** EnumMap<>(FuzzyValue.**class**);

ruleTable1.put(FuzzyValue.***NL***, nlLine);

nlLine.put(FuzzyValue.***NL***, FuzzyValue.***PL***);

nlLine.put(FuzzyValue.***NM***, FuzzyValue.***PL***);

nlLine.put(FuzzyValue.***ZR***, FuzzyValue.***PL***);

nlLine.put(FuzzyValue.***PM***, FuzzyValue.***PL***);

nlLine.put(FuzzyValue.***PL***, FuzzyValue.***PL***);

Map<FuzzyValue, FuzzyValue> nmLine =

**new** EnumMap<>(FuzzyValue.**class**);

ruleTable1.put(FuzzyValue.***NM***, nmLine);

nmLine.put(FuzzyValue.***NL***, FuzzyValue.***PM***);

nmLine.put(FuzzyValue.***NM***, FuzzyValue.***PM***);

nmLine.put(FuzzyValue.***ZR***, FuzzyValue.***PM***);

nmLine.put(FuzzyValue.***PM***, FuzzyValue.***PM***);

nmLine.put(FuzzyValue.***PL***, FuzzyValue.***PM***);

Map<FuzzyValue, FuzzyValue> zrLine =

**new** EnumMap<>(FuzzyValue.**class**);

ruleTable1.put(FuzzyValue.***ZR***, zrLine);

zrLine.put(FuzzyValue.***NL***, FuzzyValue.***ZR***);

zrLine.put(FuzzyValue.***NM***, FuzzyValue.***ZR***);

zrLine.put(FuzzyValue.***ZR***, FuzzyValue.***ZR***);

zrLine.put(FuzzyValue.***PM***, FuzzyValue.***ZR***);

zrLine.put(FuzzyValue.***PL***, FuzzyValue.***ZR***);

Map<FuzzyValue, FuzzyValue> pmLine =

**new** EnumMap<>(FuzzyValue.**class**);

ruleTable1.put(FuzzyValue.***PM***, pmLine);

pmLine.put(FuzzyValue.***NL***, FuzzyValue.***NM***);

pmLine.put(FuzzyValue.***NM***, FuzzyValue.***NM***);

pmLine.put(FuzzyValue.***ZR***, FuzzyValue.***NM***);

pmLine.put(FuzzyValue.***PM***, FuzzyValue.***NM***);

pmLine.put(FuzzyValue.***PL***, FuzzyValue.***NM***);

Map<FuzzyValue, FuzzyValue> plLine =

**new** EnumMap<>(FuzzyValue.**class**);

ruleTable1.put(FuzzyValue.***PL***, plLine);

plLine.put(FuzzyValue.***NL***, FuzzyValue.***NL***);

plLine.put(FuzzyValue.***NM***, FuzzyValue.***NL***);

plLine.put(FuzzyValue.***ZR***, FuzzyValue.***NL***);

plLine.put(FuzzyValue.***PM***, FuzzyValue.***NL***);

plLine.put(FuzzyValue.***PL***, FuzzyValue.***NL***);

//construct tabela2 FLRS for inversor, the second output

Map<FuzzyValue, Map<FuzzyValue, FuzzyValue>> ruleTable2 =

**new** EnumMap<>(FuzzyValue.**class**);

Map<FuzzyValue, FuzzyValue> generalLine =

**new** EnumMap<>(FuzzyValue.**class**);

generalLine.put(FuzzyValue.***NL***, FuzzyValue.***PL***);

generalLine.put(FuzzyValue.***NM***, FuzzyValue.***PM***);

generalLine.put(FuzzyValue.***ZR***, FuzzyValue.***ZR***);

generalLine.put(FuzzyValue.***PM***, FuzzyValue.***NM***);

generalLine.put(FuzzyValue.***PL***, FuzzyValue.***NL***);

ruleTable2.put(FuzzyValue.***PL***, generalLine);

ruleTable2.put(FuzzyValue.***PM***, generalLine);

ruleTable2.put(FuzzyValue.***ZR***, generalLine);

ruleTable2.put(FuzzyValue.***NM***, generalLine);

ruleTable2.put(FuzzyValue.***NL***, generalLine);

//returning FLRS table with two inputs and two outputs

**return** **new** TwoXTwoTable(ruleTable1, ruleTable2);

}

**public** **static** **void** main(String[] args) {

**double** w1 = 0.33;

**double** w2 = 1.5;

//specifying the limits for fuzzyfication, defuzzyfication

FuzzyDefuzzy fuzDefuz =

**new** FuzzyDefuzzy(-1.0, -0.5, 0.0, 0.5, 1.0);

//creating FLRS table for tow inputs and two outputs

TwoXTwoTable inversor = *createInversor*();

//giving the two inputs

**double** x1 = -0.33;

**double** x2 = 0.12;

//multiplying the inputs with the amplification and fuzzyfication factors

FuzzyToken inpToken1 = fuzDefuz.fuzzyfie(x1 \* w1);

FuzzyToken inpToken2 = fuzDefuz.fuzzyfie(x2 \* w2);

//executing the FLRS table

FuzzyToken[] out = inversor.execute(inpToken1, inpToken2);

//displaying the defuzzyfication results

System.***out***.println("x3 :: " + fuzDefuz.defuzzify(out[0]));

System.***out***.println("x4 :: " + fuzDefuz.defuzzify(out[1]));

}

}

**public** **enum** FuzzyValue {***NL***, ***NM***, ***ZR***, ***PM***, ***PL***;}

**import** java.util.EnumMap;

**import** java.util.Map;

**public** **class** FuzzyDefuzzy {

Map<FuzzyValue, Double> limits;

**public** FuzzyDefuzzy(**double** negativeLimit,

**double** negativeMidleValue, **double** zero,

**double** positiveMidleValuem,

**double** positiveLimit) {

limits = **new** EnumMap<FuzzyValue, Double>(FuzzyValue.**class**);

limits.put(FuzzyValue.***NL***, negativeLimit);

limits.put(FuzzyValue.***NM***, negativeMidleValue);

limits.put(FuzzyValue.***ZR***, zero);

limits.put(FuzzyValue.***PM***, positiveMidleValuem);

limits.put(FuzzyValue.***PL***, positiveLimit);

}

//fuzzyfication method

**public** FuzzyToken fuzzyfie(**double** val) {

FuzzyToken toRet = **new** FuzzyToken(0.0, 0.0, 0.0, 0.0, 0.0);

**if** (val <= limits.get(FuzzyValue.***NL***)) {

toRet.addToFuzzyValue(FuzzyValue.***NL***, 1.0);

} **else** **if** (val > limits.get(FuzzyValue.***NL***) && val <=

limits.get(FuzzyValue.***NM***)) {

**double** nm = (limits.get(FuzzyValue.***NL***) - val) /

(limits.get(FuzzyValue.***NL***) - limits.get(FuzzyValue.***NM***));

**double** nl = 1.0 - nm;

toRet.addToFuzzyValue(FuzzyValue.***NL***, nl);

toRet.addToFuzzyValue(FuzzyValue.***NM***, nm);

} **else** **if** (val > limits.get(FuzzyValue.***NM***) && val <=

limits.get(FuzzyValue.***ZR***)) {

**double** zr = (limits.get(FuzzyValue.***NM***) - val) /

(limits.get(FuzzyValue.***NM***) - limits.get(FuzzyValue.***ZR***));

**double** nm = 1.0 - zr;

toRet.addToFuzzyValue(FuzzyValue.***NM***, nm);

toRet.addToFuzzyValue(FuzzyValue.***ZR***, zr);

} **else** **if** (val > limits.get(FuzzyValue.***ZR***) && val <=

limits.get(FuzzyValue.***PM***)) {

**double** pm = (limits.get(FuzzyValue.***ZR***) - val) /

(limits.get(FuzzyValue.***ZR***) - limits.get(FuzzyValue.***PM***));

**double** zr = 1.0 - pm;

toRet.addToFuzzyValue(FuzzyValue.***ZR***, zr);

toRet.addToFuzzyValue(FuzzyValue.***PM***, pm);

} **else** **if** (val > limits.get(FuzzyValue.***PM***) && val <=

limits.get(FuzzyValue.***PL***)) {

**double** pl = (limits.get(FuzzyValue.***PM***) - val) /

(limits.get(FuzzyValue.***PM***) - limits.get(FuzzyValue.***PL***));

**double** pm = 1.0 - pl;

toRet.addToFuzzyValue(FuzzyValue.***PM***, pm);

toRet.addToFuzzyValue(FuzzyValue.***PL***, pl);

} **else** {

toRet.addToFuzzyValue(FuzzyValue.***PL***, 1.0);

}

**return** toRet;

}

//defuzzyfication method

**public** **double** defuzzify(FuzzyToken tk) {

**double** weighSum = 0.0;

**double** allSum = 0.0;

**for** (FuzzyValue fv : tk.getNonZeroValues()) {

weighSum += limits.get(fv) \* tk.getFuzzyValue(fv);

allSum += tk.getFuzzyValue(fv);

}

**return** weighSum / allSum;

}

}

**import** java.util.ArrayList;

**import** java.util.EnumMap;

**import** java.util.List;

**import** java.util.Map;

**public** **class** FuzzyToken {

//representing a fuzzy token

**private** Map<FuzzyValue, Double> fuzzyValues;

**public** FuzzyToken(Double NL, Double NM, Double ZR, Double PM,

Double PL) {

fuzzyValues = **new** EnumMap<>(FuzzyValue.**class**);

fuzzyValues.put(FuzzyValue.***NL***, NL);

fuzzyValues.put(FuzzyValue.***NM***, NM);

fuzzyValues.put(FuzzyValue.***ZR***, ZR);

fuzzyValues.put(FuzzyValue.***PM***, PM);

fuzzyValues.put(FuzzyValue.***PL***, PL);

normalize();

}

//normalizing the values

**public** **void** normalize() {

**double** sum = 0.0;

**for** (FuzzyValue index : fuzzyValues.keySet()) {

sum += fuzzyValues.get(index);

}

**if** (sum != 0.0) {

**for** (FuzzyValue index : fuzzyValues.keySet()) {

fuzzyValues.put(index, fuzzyValues.get(index) /

sum);

}

}

}

**public** **void** addToFuzzyValue(FuzzyValue fuzzyVal, **double** val) {

fuzzyValues.put(fuzzyVal, fuzzyValues.get(fuzzyVal) + val);

}

**public** **double** getFuzzyValue(FuzzyValue val) {

**return** fuzzyValues.get(val);

}

//returning the elements in fuzzyset with value different from zero

**public** List<FuzzyValue> getNonZeroValues() {

List<FuzzyValue> toRet = **new** ArrayList<>();

**for** (FuzzyValue index : fuzzyValues.keySet()) {

**if** (fuzzyValues.get(index) != 0.0) {

toRet.add(index);

}

}

**return** toRet;

}

}

**import** java.util.Map;

**public** **class** TwoXTwoTable {

//constructing FLRS table for two inputs and two outputs

Map<FuzzyValue, Map<FuzzyValue, FuzzyValue>> firstTable;

Map<FuzzyValue, Map<FuzzyValue, FuzzyValue>> secondTable;

**public** TwoXTwoTable(Map<FuzzyValue, Map<FuzzyValue, FuzzyValue>>

firstTable,

Map<FuzzyValue, Map<FuzzyValue, FuzzyValue>> secondTable) {

**this**.firstTable = firstTable;

**this**.secondTable = secondTable;

}

//implementing the FLRS table execution

**public** FuzzyToken[] execute(FuzzyToken firstToken, FuzzyToken

secondToken) {

**if** (firstTable == **null** || secondTable == **null**) {

**throw** **new** RuntimeException("Tables not setted");

}

FuzzyToken toRet1 = **new** FuzzyToken(0.0, 0.0, 0.0, 0.0, 0.0);

FuzzyToken toRet2 = **new** FuzzyToken(0.0, 0.0, 0.0, 0.0, 0.0);

**for** (FuzzyValue firstTokenNonZero :

firstToken.getNonZeroValues()) {

**for** (FuzzyValue secondNonZero :

secondToken.getNonZeroValues()) {

FuzzyValue firstConclision =

firstTable.get(firstTokenNonZero).get(secondNonZero);

FuzzyValue secondConclusion =

secondTable.get(firstTokenNonZero).get(secondNonZero);

**double** valueOfConclision =

firstToken.getFuzzyValue(firstTokenNonZero)

\* secondToken.getFuzzyValue(secondNonZero);

toRet1.addToFuzzyValue(firstConclision,

valueOfConclision);

toRet2.addToFuzzyValue(secondConclusion,

valueOfConclision);

}

}

toRet1.normalize();

toRet2.normalize();

**return** **new** FuzzyToken[] { toRet1, toRet2 };

}

}

Exercises:

1. Test the application for different values of *x1,x2, w1* and *w2* and display *x3* and *x4*.
2. It is required to construct a graphical interface to insert *x1, x2, w1* and *w2*, and after execution, *x3* and *x4* will be displayed in the graphical interface. The graphical interface will also contain a button for executing the transition.
3. Starting from the previous application, it is required to build an application that will work as an adder/ subtractor and will calculate the output values *x3* and *x4*. Output values will be displayed in the graphical interface. The graphical interface will also contain a button for executing the transition.

The mathematical model of the adder / subtractor component:

(x1, x2) 🡺 (x3, x4) x3 = x1 + x2; x4 = x1 – x2; w1 = w2 = 1:

x3 = w1∙x1 + w2∙x2 ; ↔ (w1∙x1 + w2∙x2) ϵ [-1,1]

x3 = -1; ↔ (w1∙x1 + w2∙x2) < -1

x3 = 1; ↔ (w1∙x1 + w2∙x2) > 1

x4 = w1∙x1– w2∙x2 ; ↔ (w1∙x1 – w2∙x2) ϵ [-1,1]

x4 = -1; ↔ (w1∙x1 – w2∙x2) < -1

x4 = 1; ↔ (w1∙x1 – w2∙x2) > 1

and the FLRS table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x1 \ x2** | **NL** | **NM** | **ZR** | **PM** | **PL** |
| **NL** | NL, ZR | NL, NM | NL, NL | NM, NL | ZR, NL |
| **NM** | NL, PM | NL, ZR | NM, NM | ZR, NL | PM, NL |
| **ZR** | NL, PL | NM, PM | ZR, ZR | PM, NM | PL, NL |
| **PM** | NM, PL | ZR, PL | PM, PM | PL, ZR | PL, NM |
| **PL** | ZR, PL | PM, PL | PL, PL | PL, PM | PL, ZR |

The class diagram is required.

Test for different values of *x1 , x2, w1* and *w2*.

1. Verification of knowledge

1) What are fuzzy sizes and how do they work with them?

2) How to change from real variables to fuzzy variables?

3) Explain what is a Fuzzy Logic Enhanced Time Petri Nets Network and what is it used for?

4) What is a token set in a place *pi* on the FLETPN network?

5) What is the usefulness of the Fuzzy Logic Rule Set tables?