Generation of Fuzzy (Type 2) Logic Based Systems

Contents
- Introduction
- Fuzzy Logic (Type 2)
- Implementation of Fuzzy Logic
- Example of Fuzzy Logic Using Framework
- Conclusion
6.1 Introduction

This chapter focuses on design and development of fuzzy logic based systems. Further the fuzzy logic is extended to type 2 fuzzy logic. Fuzzy logic based systems has components that consists of fuzzy rules, fuzzifier, de-fuzzifier, inference engine, a type reducer for type 2 fuzzy systems. Each of these components has specific usage for generating a part of the system to be developed. The detailed development approach for each of these components is presented in this chapter. The chapter elaborates on fuzzification and defuzzification methods, membership functions, fuzzy sets, linguistic variable and their applicability to deduce fuzzy rule to infer advice from the knowledge base for the application under development. Various fuzzy inference models are taken into consideration depending upon their practical usability. The rules and fuzzy inference can be saved for future use of the application. The chapter mentions in detail the advantages of using type 2 fuzzy logic; this is justified by measuring the degree of similarity through membership functions. The chapter mentions about the application areas where fuzzy inference based systems are used.

6.2 Fuzzy Logic (Type 2)

Fuzzy logic can be defined as generalization of crisp data. In other words it was design to represent vague information for a given problem domain specified under specific domain range. Fuzzy logic was developed by Prof. Zadeh [4] to represent real life attribute into machine system.

Since there was lot of criticism about type 1 fuzzy logic and its membership function that there was no uncertainty associated with it contradicted with the meaning of the word ‘fuzzy’. Since the meaning of the word fuzzy conveys uncertainty, hence type 2 fuzzy logic was proposed as an extension to type 1 fuzzy logic by Prof. Zadeh in 1975[4]. Later Jerry mendel and Johnl[1], Q.Liang et.al [8] worked on concept of type 2 fuzzy logic to deduce fuzzy to fuzzy mapping thus being a second order of the fuzzy logic. Hence type 2 fuzzy logic is not generalization of crisp value, but it represents uncertain and vague conditions which are often encountered in day to
day activities of human being into machines by specifying values under the principle of footprint of uncertainty[2], [3], [9].

The membership function of type 2 fuzzy set is three dimensional, where the third dimension is the value of the membership function at each point on its two-dimensional domain that is called its footprint of uncertainty (FOU). The term footprint of uncertainty is very useful, because it not only focuses our attention on the uncertainties inherent in a specific type-2 membership function, whose shape is a direct consequence of the nature of these uncertainties, but it also provides a very convenient verbal description of the entire domain of support for all the secondary grades of a type-2 membership function as shown by Mendel and John[1], and Mendel[6]. Another method for interval type 2 fuzzy system was also proposed by Wu and Mendel[11].

Various fuzzy editors or tools or programs like FuzzyCOPE [https://www.aut.ac.nz/resources/research/research_institutes/kedri/downloads/pdf/fuzzycope.pdf], Fuzzy Logic Inference Engine, Fuzzy Clips [http://www.graco.unb.br/alvares/DOUTORADO/omega.enm.unb.br/pub/doutorado/disco2/ai.iit.nrc.ca/IR_public/fuzzy/fuzzyClips/fuzzyCLIPSFiles.html], NEFCLASS [http://fuzzy.cs.uni-magdeburg.de/nefclass/], Fuzzy logic development environment for embedded systems, fuzzy logic inferencing toolkit [http://www.mathworks.in], FuzzyTECH [http://www.fuzzytech.com/] are available on the internet with a limited functionalities which includes terms and conditions. Software like Matlab provides fuzzy logic tool box[http://www.mathworks.in] with specified Fuzzification and Defuzzification methods, however this files are not easy to configure with other source code, and require much effort to deploy them on web.

The developed library incorporates type 2 fuzzy membership functions; hence the inference generated from the system is similar to human reasoning and understandability. Thus when type 2 fuzzy inference is applied along with artificial neural network, an expert neuro-fuzzy based advisory can be obtained in the required domain area. This functionality is also made available in the developed framework from the generic library. The users of the framework have to specify the
rules and select the membership functions to apply fuzzy inference. The practical implementation of type 2 fuzzy logic has been depicted in section 6.3 of this chapter.

### 6.3 Implementation of Fuzzy Logic

A fuzzy logic based system consists of four basic components, fuzzifier, inference engine, rule base and defuzzifier. Type 2 fuzzy logic has an additional component called as type reducer, which helps in conversion of type 2 fuzzy logic to type 1 fuzzy logic using type 2 membership functions that models uncertainty to represent it in type 1 fuzzy logic based systems. Figure 6.1 shows the components that are developed for the library. These components include the existing methods that are already developed as well as some modified methods which are proposed by researchers of the domain under study.

![Figure 6.1: Fuzzy Components of the Library](image)

The structure of fuzzy logic as depicted in Figure 6.1 is utilized by framework in order to generate fuzzy logic based systems. The description and usages of these components are as follows.
**Fuzzifier:** First of all, when the input will be directed to the fuzzy inference system they need to be fuzzified so that they can be determined to which degree they belong for the fuzzy sets through membership functions. Fuzzification of the inputs amounts to either table lookup or evaluation of the function. Generally a membership function is associated which determines the degree of measurement for the given number within a specified range. For sake of simplicity it is advisable to consider a range between 0 and 1. The triangular membership function can divide the specified inputs into three different values pointing to linguistic variables which are listed under a table for a given problem under consideration. Similarly a trapezoidal function can divide the specified inputs into four different values pointing to linguistic variable. This mechanism is included in the library, which is depicted in code snippet 1 as follows:

**Code Snippet 1:** This code snippet operates on triangular membership function for the purpose of fuzzification of the input.

```csharp
Triangularmf(double a, double b, double c)
{
    Params.Add(a); Params.Add(b); Params.Add(c);
    range.Add(Params[0]);
    range.Add(Params[Params.Count - 1]);
    this.CalculateCenterOfMass();
    this.calculateSpread();
}
protected void CalculateCenterOfMass()
{
    double stepN = 20;
    double start = this.Range[0];
    double end = this.Range[1];
    double step = (end - start) / (stepN - 1);
    double uppersum = 0;
    double lowersum = 0;
    for (double j = start; j < end; j = step + j)
    {
    }
```
double value = this.getOutput(j);
if (value > 0)
{
    uppersum = value * j + uppersum;
    lowersum = value + lowersum;
}

centerofmass = (uppersum / lowersum);

protected void calculateSpread()
{
    spread = range[1] - range[0] - 1;
}

Fuzzy rule base: A rule base is generated which collects information regarding every possible outcome that may occur when a specified condition is encountered. A fuzzy rule base is a collection of various such IF-THEN rules where the IF part acts an antecedent and THEN part acts a consequent for the firing rules. The firing rule strength is determined by the aggregating all the output by an inference engine. To generate a rule following code snippet 2 is used by the library. This rule is documented for future use of the system.

Code Snippet 2: This code snippet specifies a function to generate rule for the rule base of the system.

public string GenerateRule()
{
    String rule = "IF ";
    for (int j = 0; j < InputRules.Count; j++)
    {
        rule = rule + InputRules[j].Variable + " is " + InputRules[j].MembershipValue;
        if (j != InputRules.Count - 1)
Inference engine: The job of inference engine in fuzzy logic based system is to derive appropriate answers from rule base by evaluation of rules that are to be considered. The rules are extracted from the rule base with the help of membership function that measures degree of similarity. The library also incorporates the same mechanism. The rules are evaluated on the basis of certain checks and conditions. Furthermore the evaluated rules are inserted and extracted from rule base whenever required as shown in code snippet 3.

Code Snippet 3: This code snippet shows how the rule items are inserted and extracted from the rule base.

```csharp
protected void insertInRuleItem(RuleItem item)
{
    if (item != null)
    {
        inrules.Add(item);
    }
}
```
Knowledge Base: The inference engine extracts rules from rule base on basis of evaluation using membership functions. These selected rules are documented to create a knowledge base; hence whenever expert system is to be developed, the evaluated rules can be readily extracted from the knowledge base. The framework developed allows documenting the rules for their future use and modifications.

Defuzzification: when the inference engines computes its process the final decision is the generalized decision made on set of all rules that are specified under a rule base for certain situation. When the decisions are required to be obtained in crisp form we need to defuzzify them. In case of type 2 fuzzy systems an additional component known as ‘type reducer’ is required to convert second order fuzzy sets to simple fuzzy sets, which can be defuzzified to crisp values. There are various defuzzification procedures[10] like centroid, bisector, middle of maximum, smallest of maximum, largest of maximum, weighted fuzzy mean, etc. which are used as per requirement of the problem under consideration. The developed library incorporates various defuzzification procedures that facilitates user of the framework to select any of the defuzzification methods. Code snippet 4 show defuzzication that uses centroid method.

Code Snippet 4: The code snippet shows defuzzification of fuzzy sets.

```java
public double DeFuzzification(List<FuzzySet> Sets, LingVariable variable) {
    double value = 0;
    }
```
configuration.DefuzzificationType = DefuzzificationType.Centroid;
for (int i = 0; i < Sets.Count; i++)
{
    if (Sets[i].Variable == variable.Name)
    {
        value = Centroid(Sets[i].Set, variable);
    }
}
return value;

The Classes for Fuzzy Logic

This section describes the major classes of the library that supports development of fuzzy logic based systems.

Class Name → FuzzyLogic.cs
Description → This is the main class of the fuzzy logic development process. All other classes are called through this class by creating the object of this class whenever required. The class calls main methods of the fuzzy logic systems and the results are returned after applying proper fuzzy inference. The fields and method of the class are described as follows:
### Fields

<table>
<thead>
<tr>
<th>✓ config</th>
<th>It holds the configurations of the fuzzy logic system to be developed.</th>
</tr>
</thead>
</table>

### Methods

| ✓ Fuzzification | Method for conversion of the data into required fuzzy sets. |
| ✓ Defuzzification | Method for conversion of fuzzy set data to a crisp number. |
| ✓ Type2Defuzzification | Method for conversion of type 2 fuzzy set to type 1 fuzzy sets. |

#### Class Name ➔ FuzzySet.cs

**Description** ➔ This class groups the data into fuzzy sets and keeps track of the generated fuzzy set. The fields, properties and method of the class are described as follows:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ lingvar</td>
<td>Names of the linguistic variable</td>
<td></td>
</tr>
<tr>
<td>✓ fnums</td>
<td>list of fuzzy numbers.</td>
<td></td>
</tr>
<tr>
<td>✓ ni</td>
<td>Number of inputs to the neural network (First layer)</td>
<td></td>
</tr>
<tr>
<td>✓ set&lt;List&gt;</td>
<td>sets the list of fuzzy numbers.</td>
<td></td>
</tr>
<tr>
<td>✓ Variable</td>
<td>to set or get the name of the fuzzy variable.</td>
<td></td>
</tr>
</tbody>
</table>

#### Class Name ➔ LinguisticVariable.cs

**Description** ➔ This class is used for linguistic variables that are defined by user during development of the system using frame work. The fields, properties and method of the class are described as follows:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ lingvar</td>
<td>Names of the linguistic variable</td>
<td></td>
</tr>
<tr>
<td>✓ fnums</td>
<td>list of fuzzy numbers.</td>
<td></td>
</tr>
<tr>
<td>✓ ni</td>
<td>Number of inputs to the neural network (First layer)</td>
<td></td>
</tr>
<tr>
<td>✓ set&lt;List&gt;</td>
<td>sets the list of fuzzy numbers.</td>
<td></td>
</tr>
<tr>
<td>✓ Variable</td>
<td>to set or get the name of the fuzzy variable.</td>
<td></td>
</tr>
</tbody>
</table>

| ✓ FuzzySet | Constructor of the class that takes fuzzy set and linguistic variable as argument. It assign linguistic variable to fuzzy set. |
| ✓ MembershipValue | Determines the degree of the membership function by calculating the value. |
### Class Name: Rule.cs

**Description:** It adds and extracts rules from the rule base. The fields, properties and method of the class are described as follows:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ conn → List of connector that associates rules.</td>
<td>✓ Connections → gets or sets connection.</td>
<td>✓ InsertRuleItem → Adds in going rule item to the list.</td>
</tr>
<tr>
<td>✓ inrule → list of incoming rules.</td>
<td>✓ InputRules → gets input rules.</td>
<td>✓ ExtractRuleItem → Adds out going rule item to the list.</td>
</tr>
<tr>
<td>✓ outrule → list of outgoing rules.</td>
<td>✓ OutputRules → gets output rules.</td>
<td></td>
</tr>
</tbody>
</table>

### Class Name: Membershipfunction.cs

**Description:** This class gathers different types of membership functions. The fields, properties and method of the class are described as follows:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ label → label for the linguistic variable.</td>
<td>✓ MFs → gets or sets membership functions.</td>
<td>✓ addMF → Adds a membership function for the variable.</td>
</tr>
<tr>
<td>✓ funcs → list of membership functions.</td>
<td>✓ Name → Name of the linguistic variable.</td>
<td>✓ getMFName → gets name of membership function.</td>
</tr>
<tr>
<td>✓ range → range of the linguistic variable.</td>
<td>✓ Range → gets or sets range for linguistic variable.</td>
<td>✓ removeMF → removes assigned membership function.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Property</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>label for the linguistic variable.</td>
<td>addMF</td>
</tr>
<tr>
<td>funcs</td>
<td>list of membership functions.</td>
<td>getMFName</td>
</tr>
<tr>
<td>range</td>
<td>range of the linguistic variable.</td>
<td>removeMF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Property</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFs</td>
<td>gets or sets membership functions.</td>
<td>addMF</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the linguistic variable.</td>
<td>getMFName</td>
</tr>
<tr>
<td>Range</td>
<td>gets or sets range for linguistic variable.</td>
<td>removeMF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Property</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn</td>
<td>List of connector that associates rules.</td>
<td>InsertRuleItem</td>
</tr>
<tr>
<td>inrule</td>
<td>list of incoming rules.</td>
<td>ExtractRuleItem</td>
</tr>
<tr>
<td>outrule</td>
<td>list of outgoing rules.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Property</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>gets or sets connection.</td>
<td></td>
</tr>
<tr>
<td>InputRules</td>
<td>gets input rules.</td>
<td></td>
</tr>
<tr>
<td>OutputRules</td>
<td>gets output rules.</td>
<td></td>
</tr>
</tbody>
</table>
Class Name → InferenceEng.cs

Description → The class deals with inference mechanism of fuzzy logic based system. The fields, properties and method of the class are described as follows:

<table>
<thead>
<tr>
<th>Fields</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ config → configuration status for inference.</td>
<td>✓ Fired Rules: gets the fired rule from the inference engine after evaluation.</td>
<td>✓ EvaluateRules → calculate whether a particular rule can be applied or not.</td>
</tr>
<tr>
<td>✓ Rulesfired → list of rules used by inference.</td>
<td>✓ Implication → Checks whether a particular rule is applied or not.</td>
<td>✓ getRuleValue → gets value of the evaluated rule.</td>
</tr>
<tr>
<td>✓ fuzzysets → list of fuzzy to apply inference.</td>
<td>✓ InferenceEng → Parameterized constructor sets default values for rules.</td>
<td></td>
</tr>
</tbody>
</table>
6.4 Example of Fuzzy System Using Framework

To demonstrate the working of the fuzzy interface of the framework, a case study for an automated braking system available in modern car is implemented. Consider the linguistic variable "Speed" of the car which has 2 membership functions Low and High, in which "Low" has range from 0 mph - 10 mph and "High" from 10 mph - 20 mph. Hence for crisp input of 5 its fuzzy value is Low.

Following steps are followed for deducing the fuzzy inference from the fuzzy rule based systems. From the classes stated previously the fuzzy inference is applied on the case under study. Figure 6.2 shows the fuzzy interface of the framework for considered case study.

Step 1: Define linguistic variables and membership functions

We define the car to have speed for input linguistic variable. For speed: High, Med and Low are the respective range defined for membership functions.

For output linguistic variable Brake is used. For Brake : Release and Push are the respective range defined for membership functions..

```
LingVariable speed = new LingVariable("Speed", VarType.Input); //declare new linguistic variable
speed.setRange(0, 35); // set max - min range for our variables 0 mph -35 mph
speed.addMF(new Trapmf("Low", -10, 0, 10, 15)); // trapmf: trapezoid shape behaviour
speed.addMF(new Trimf("Medium", 10, 20, 30)); // trimf: triangle shape behaviour
speed.addMF(new Trapmf("High", 25, 30, 35, 40));

LingVariable brake = new LingVariable("Brake", VarType.Output);
brake.setRange(0, 65); // Brake Force
brake.addMF(new Trapmf("Released", -10, 0, 20, 41));
brake.addMF(new Trapmf("Pushed", 41, 60, 65, 70));
```
Step 2: The Process of Fuzzification

Fuzzification is the process of evaluating the crisp input against the membership functions of the linguistic variable Speed, membership function is an abstract class, with an abstract method "getOutput", any object inheriting this class needs to define their getOutput(double) function from the library, such as in Trimf and Trapmf. Based upon this feature, more membership functions can be implemented in future.

Fuzzification method then returns a list of fuzzy numbers, each fuzzy number consists of a function name and firing strength from getOutput(double).

The list of fuzzy numbers is then added to a fuzzy set with list of fuzzy numbers and linguistic variable name for further processing.

FuzzyLogic c = new FuzzyLogic(conf);
double speedCrispVal = 30;
FuzzySet fuzzy_speed = new FuzzySet(c.Fuzzification(speedCrispVal,speed), speed.Name);

After we fuzzify all our inputs, add it to a list of fuzzysets:

List<FuzzySet> input_sets = new List<FuzzySet>();
input_sets.Add(fuzzy_speed);

**Step 3: Define fuzzy rules**

Define the rulebase, the rules that our system uses to approximate.

For Rule 1,
"IF Speed is High, Then Brake is pushed."

List<ruleitem> rule1in = new List<ruleitem>();
List<ruleitem> rule1out = new List<ruleitem>();

// the if part of the Rule, add more than one if X1 and X2,
// add another RuleItem in the list
rule1in.AddRange(new RuleItem[1] { new RuleItem("Speed", "High") });

// the then part in the Rule
rule1out.AddRange(new RuleItem[1] { new RuleItem("Brake", "Pushed") });

// List of rules "RuleBase" passed to the Inference Engine
List<rule> rules = new List<rule>();
rules.Add(new Rule(rule1in, rule1out, Connector.And));

**Step 4: Evaluation of the rules**

During the rules evaluation at the Inference system, the firing strength of the output is determined as in this example:

Fuzzy input: X1 = 0.8 Low X2 = 0.5 High.

Rule: "IF X1 Low and X2 High Then Y1 is Low."

Configuration: And connection evaluated by min(X1,X2).

Evaluation: min(X1,X2) => min(0.8,0.5) = Y1 = 0.5 Low.

Fuzzy out: Y1 = 0.5 Low.

If more rules are overlapping, such as "IF X1 is High, THEN Y1 is Low" where the Y1 is 0.7 Low, the maximum of Y1 firing strengths is taken, in this case (0.7).

InferEngine engine = new InferEngine(configure, rules, input_sets);
Step 5: Defuzzification (crisp output)

The Defuzzification method is a method to calculate the fuzzy out to convert to a crisp value. In this system, two methods of defuzzification are used, you can choose between them in Configuration.

Centroid Defuzzification, Modified High Defuzzification.

double crisp_brake_force = c.DeFuzzification(fuzzy_out, brake);

The aforementioned steps are applicable for constant as well as non constant input of data. The fuzzy inference tool of the framework is more important where the linguistic variable is not constant, which can be defined by a set of membership function[7]. The membership function provides measure for degree of similarity between the exact output values to the obtained values. The minor difference can be sorted out under type 2 fuzzy logic using the principle of footprint of uncertainty (FOU). Hence a generalized bell shaped membership function can be obtained using type 2 fuzzy inference over the selected fuzzy inference model and is suitable for hybridizing with artificial neural network [5].

6.5 Conclusion

It is clear from the example demonstrated that the demonstrated fuzzy logic component of the framework has ability to deduce fuzzy inference on different cases. The friendly user interface provided by the framework helps in selection of membership functions and to generate automatic fuzzy rules which can be saved and can be modified at later stage. Based on the membership function selected and fuzzy rules provided for the specific linguistic variable, fuzzy inference (advice) is generated in user friendly format which can be documented in the knowledge base for the purpose of future usage. The framework provides facility to generate source code for the developed case study which can be modified later according to required modification for development of other fuzzy rule based systems. Further the fuzzy component of the framework can be hybridized with artificial neural network to
generate automated neuro-fuzzy advisory systems in different domain areas. This functionality of the framework is shown in chapter 7 of the thesis.

References


