FuzzyLite v1.0

A Fuzzy Inference System written in C++

by Juan Rada-Vilela

Supported by the Foundation for the Advancement of Soft Computing

April, 2010

Contents

List of Figures

Chapter 1

Overview

1.1 Introduction

`FuzzyLite v1.0` is a multiplatform, free, and open-source Fuzzy Inference System (FIS) written in C++ and released under the Apache License 2.0, which makes this software freely available for commercial and non-commercial use. The idea behind this FIS is to have a very simple and lite FIS. Simple as in simple to use, simple to understand, and simple to extend, without sacrificing performance. And lite because it requires no additional libraries more than the Standard Template Library included in the C++ Standard Library. It has an object-oriented approach and a clear separation between the headers and sources, so it is easy to extend. Furthermore, it is GUI-agnostic, meaning that the FIS does not require a GUI to run, encouraging its use as a library. Nevertheless, a Qt-based GUI is provided using `FuzzyLite v1.0` as a shared library.

1.2 Features

- Linguistic terms are continuous and the following ones are available: triangular, trapezoidal, rectangular, shoulder, singleton, custom function, and compound (multiple functions).
- Export any fuzzy system to text using a slightly modified version of the Fuzzy Controller Language (FCL).
- Defuzzification using center of gravity (COG).
- Mamdani rule parsing with grammar checking.
- Takagi Sugeno rules of any order (e.g. f(x) = (sin x) / x, f(x) = 0.5 * input-1).
- Weights for each rule.
- TNorm: minimum, product, bounded difference.
- SNorm: maximum, sum, bounded sum.
- Modulation: clipping, scaling.
- Aggregation: maximum, sum, bounded sum.
- Variable sampling size for membership functions to compute area and centroid.
- Triangulation algorithm to compute the area and centroid.
- Hedges: not, somewhat, very, any.

- Very easy to implement and incorporate new linguistic terms, defuzzification methods, fuzzy rules (antecedents and consequents), fuzzy operations (T-Norms, S-Norms, methods for modulation and aggregation), algorithms for computing the area and centroid of linguistic terms, hedges, among other things.

1.3 What's new?

- The GUI is working again.
- A class diagram of `FuzzyLite v1.0`.
- A detailed explanation of `FuzzyLite v1.0`.
- Minor changes.

1.4 What's next?

- Fix the GUI so Takagi Sugeno rules can be tested with their respective graph.
- Improve the `InfixToPostfix` class so infix functions are parsed as one would normally expect.
- Load the fuzzy engine from text using the Fuzzy Controller Language (FCL).
- Include more linguistic terms (e.g. sigmoidal, gaussian, sine, cosine).
- Include more defuzzifiers (e.g. Right Most Maximum, Left Most Maximum, Mean Maximum).
- Make some functions inline to increase performance and check those that are already inline to ensure they do increase performance.
- Create the `configure` script using the `GNU Autotools` in order to easily build `FuzzyLite v1.0` from sources.

1.5 Known bugs

- `InfixToPostfix` conversion might not parse functions as one would normally expect. For example, $\sin(x)/x$ is *not* equivalent to $(\sin x)/x$, the latter yields the result one might expect. Make sure by validating the postfix expression, or by evaluating its results.
- In the GUI, when using a Takagi Sugeno system, the output graphs do not work.

1.6 Building from source

1.6.1 `FuzzyLite v1.0`

At the moment, due to the lack of a building script like `./configure`, the following steps could build `FuzzyLite v1.0`:

1. Create a `C++` Project either in Eclipse IDE or Netbeans IDE.
2. Add all the source files and include files to the project.
3. Add `.` to the includes path in the project properties.

4. Add the directive `-DFL_USE_LOG` to enable the use of logging via `FL_LOG(message)`. In `./include/defs.h` there are more symbols that can be defined via `-D` for further customization.

5. Decide whether to build a library or an executable (in project properties).

6. Use the IDE's normal build.

1.6.2 Graphic User Interface

In order to build the graphical user interface of `FuzzyLite v1.0`, it is necessary to first install `Qt` which can be freely downloaded from http://qt.nokia.com/.

The `Makefile` included within the `./gui` is quite easy to read. Basically, the most important thing here is to copy the `libfuzzylite.dylib` (or whatever the extension be according to your platform) into the folder `./gui/dist` which is where the executable will be put. An alternative is to copy the library into `/usr/local/lib` and comment those lines in the `Makefile` that build and copy the library into the `./gui/dist` directory.

After configuring the `Makefile` to fit your system, a `make all` from `./gui` would build the graphical user interface of `FuzzyLite v1.0`. To run, it suffices to execute `./gui` from `./gui/dist`.

1.7 Acknowledgements

This work was possible thanks to the Foundation for the Advancement of Soft Computing, and specially to Sergio Guadarrama and Luis Magdalena.

Chapter 2

The Model

This chapter is devoted to explain `FuzzyLite v1.0` by means of a class diagram based on UML. For a better comprehension, it is divided in five groups: Fuzzy operations, Linguistic variables and terms, Linguistic terms, Fuzzy rules, Fuzzy engine, and Fuzzy exceptions. It is important to mention that all classes related to `FuzzyLite v1.0` are inside the namespace `fl`.

2.1 Fuzzy operations

Figure 2.1 shows the class diagram for this group. The classes that can be seen in it are briefly explained in the following sections.

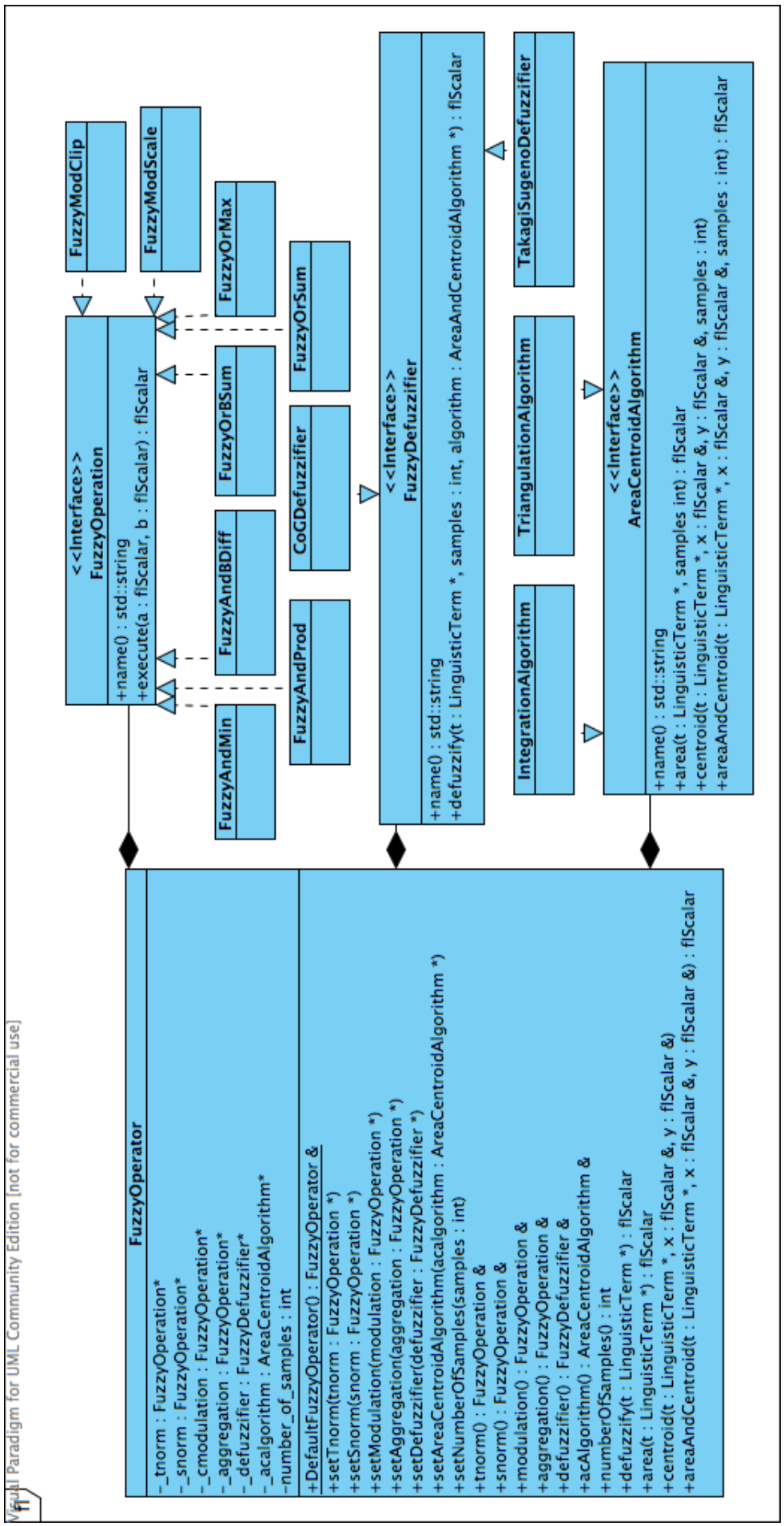


Figure 2.1: Class diagram: Fuzzy operations

2.1.1 FuzzyOperator

FuzzyOperator centralizes all the operations that can be performed in the fuzzy system. It contains the type of T-Norms, S-Norms, modulation, aggregation, defuzzification methods, algorithms for computing the area and centroid of any linguistic term, and the number of samples that are drawn from any linguistic term to be used by the algorithm.

This class has an static default FuzzyOperator instance that can be obtained using the method fl::FuzzyOperator::DefaultFuzzyOperator() anywhere and anytime. The defaults for this operator are the following

- **T-Norm:** FuzzyAndMin.
- **S-Norm:** FuzzyOrMax.
- **Modulation:** FuzzyModClip.
- **Aggregation:** FuzzyOrMax.
- **Defuzzifier:** CoGDefuzzifier.
- **Algorithm:** TriangulationAlgorithm.
- **Number of samples:** 100.

These defaults may be changed at any moment from anywhere, but consider that this is an instance that is shared among all instances from many classes, so be careful about changing these values. Nevertheless, if you need to, you may use different instances among all those classes composed by FuzzyOperator.

2.1.2 FuzzyOperation

This is the interface shared by all T-Norms, S-Norms, and methods for modulation and aggregation. If you want to implement your own operations, you may do so by implementing this interface. The operations included in FuzzyLite v1.0 are:

- T-Norm: minimum (FuzzyAndMin), product (FuzzyAndProd), and bounded difference (FuzzyAndBDiff).
- S-Norm: maximum (FuzzyOrMax), sum (FuzzyOrSum), bounded sum (FuzzyOrBSum).
- Modulation: clipping (FuzzyModClip), scaling (FuzzyModScale).
- Aggregation: maximum (FuzzyOrMax), sum (FuzzyOrSum), bounded sum (FuzzyOrBSum).

2.1.3 FuzzyDefuzzifier

This is the interface shared by all defuzzifiers. If you want to implement a different defuzzifier, you may do so by implementing this interface. The defuzzifiers included in FuzzyLite v1.0 are: Centre of Gravity (CoGDefuzzifier) for Mamdani rules, and TakagiSugenoDefuzzifier for Takagi-Sugeno rules.

2.1.4 AreaAndCentroidAlgorithm

This is an interface used for computing the area and centroid of linguistic terms. The algorithms included in FuzzyLite v1.0 are: TriangulationAlgorithm which is a triangulation algorithm appropriate for terms that can be easily triangulated, and IntegrationAlgorithm which is a regular integration algorithm. This algorithm should be chosen according to the shape of the fuzzy partitions. You may create your own algorithm by implementing this interface.

2.2 Linguistic variables and terms

Figure 2.2 shows the class diagram for the linguistic variables and terms included in FuzzyLite v1.0.

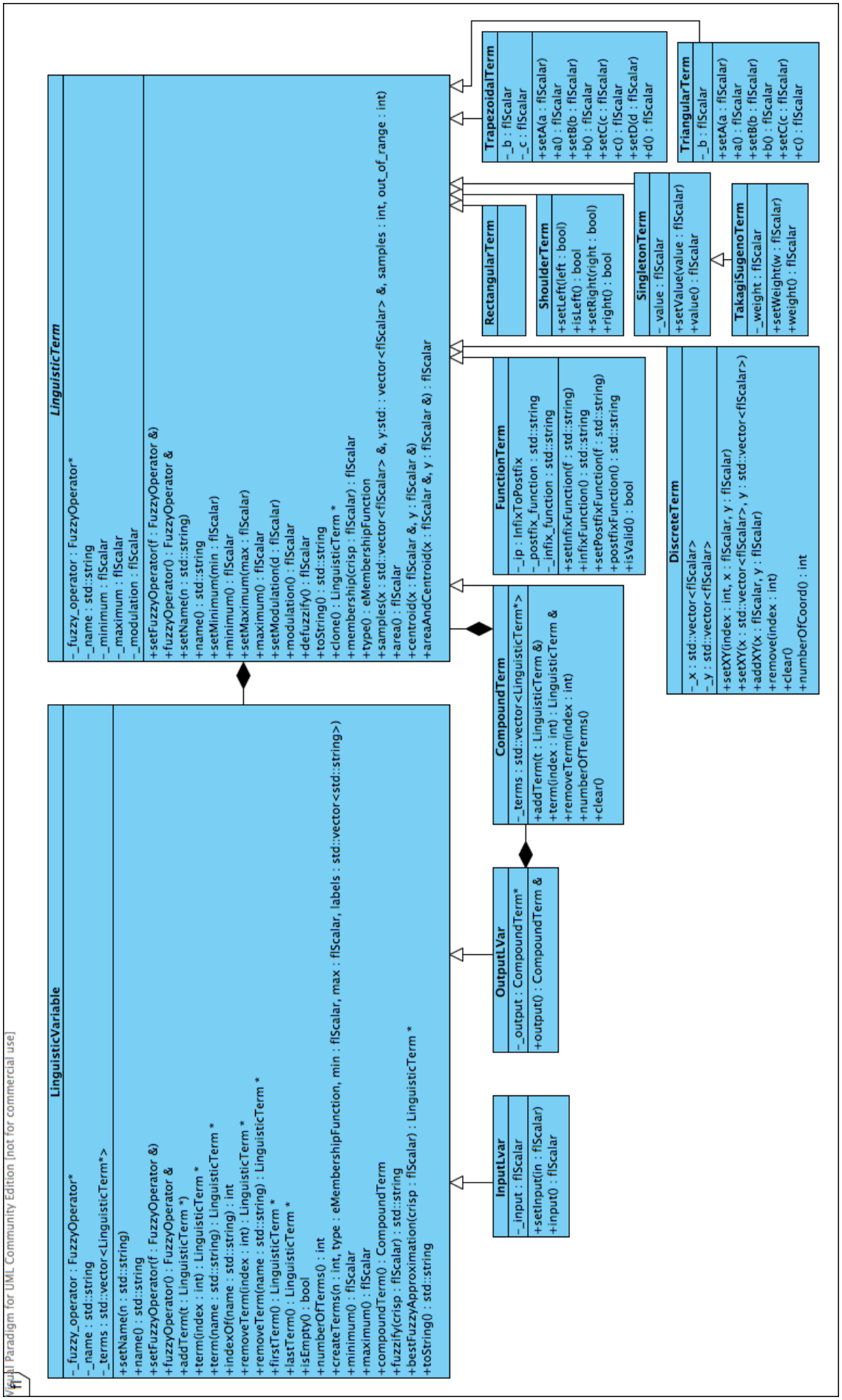


Figure 2.2: Class diagram: Linguistic variables and terms

2.2.1 LinguisticVariable

Linguistic variables are composed of a vector of LinguisticTerms. This is an abstract class that is used as base for input variables (InputLVar) and output variables (OutputLVar). Each input variable has an input value that should be the input received by the system, and each output variable has a compound linguistic term composed by all the linguistic terms added by the activation of the different rules given the input values of the input variables and their respective processing.

2.2.2 LinguisticTerm

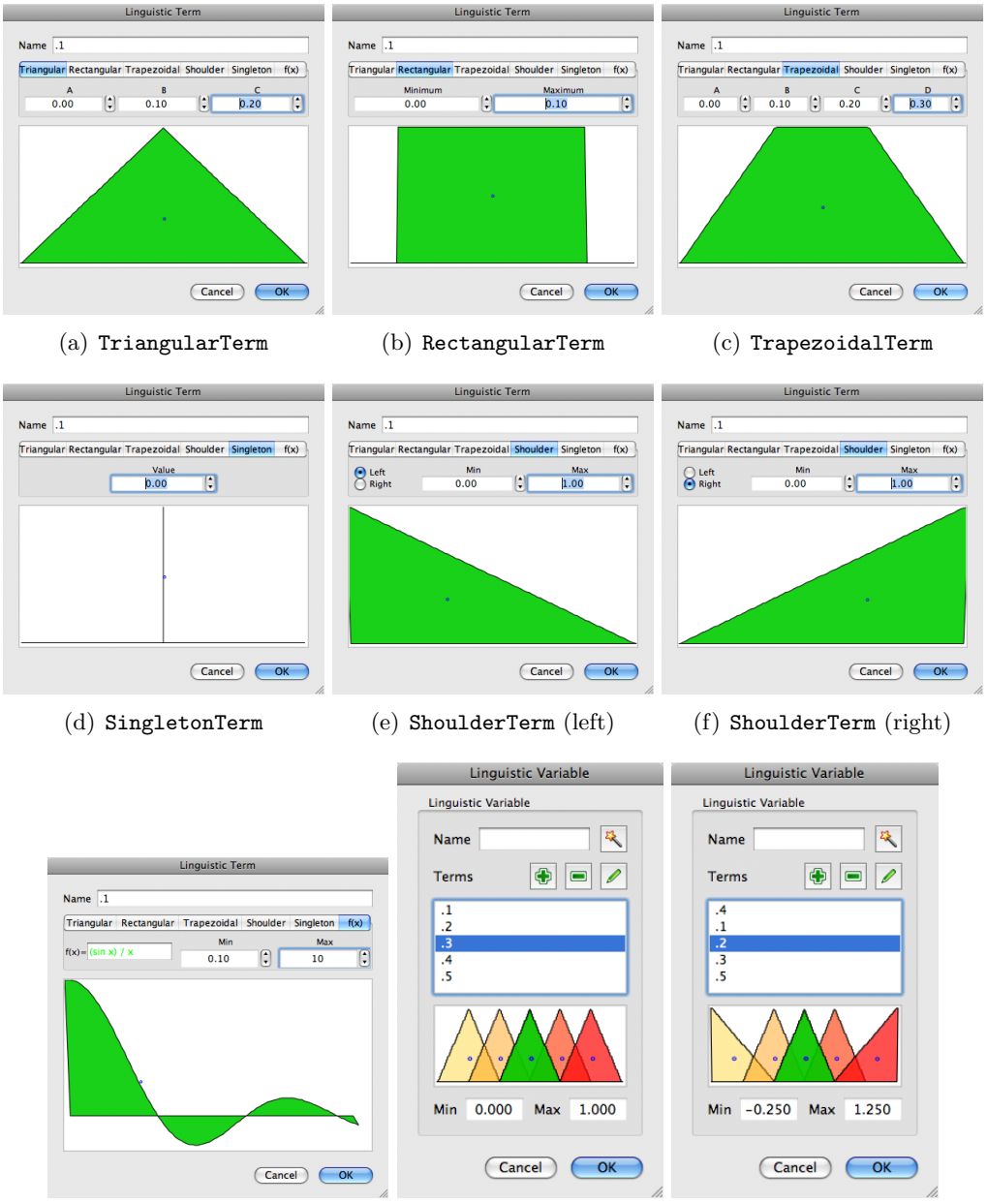
Linguistic terms, also known as fuzzy partitions, define the shape of each label. Each linguistic term has a minimum and maximum that are used to define the limits of the area it covers. Several linguistic terms are included in FuzzyLite v1.0:

- TriangularTerm: is the well known triangular term, it is defined by its three vertices a, b, and c, where a and c are wrappers of minimum() and maximum(), respectively.
- RectangularTerm: is a simple rectangular term delimited by the minimum and maximum of its parent class LinguisticTerm.
- TrapezoidalTerm: defines a trapezoid by its four vertices a, b, c, and d, where a and d are wrappers of minimum() and maximum(), respectively.
- SingletonTerm: defines a singleton for the value v delimited by $v - \delta_{low}$ and $v + \delta_{hi}$ where δ_{low} and δ_{hi} are very small values in order to be able to create samples of this term.
- ShoulderTerm: defines a trapezoid that extends to $\pm\infty$ depending on whether it is left ($-\infty$) or right ($+\infty$).
- DiscreteTerm: is a term composed of a vector of (x, y) coordinates. It is important to note that when sampling this type of term, the result of membership(flScalar crisp) is the value of y for the closest x given the value of crisp.
- FunctionTerm: is a term that is defined as a function $f(x)$ which accepts several mathematical and trigonometrical operations over a given function, for example: setInfixFunction("(sin x) / x"). It is *very* important to remark that the parser used to parse infix functions may not function as expected (e.g. $\sin(x)/x$ is different of $(\sin x)/x$, the latter one gives the result one expects). When in doubt, test your expression by converting it to postfix.
- CompoundTerm: is a term composed of a list of LinguisticTerms. This is used in OutputLVar in order to aggregate all the linguistic terms that were added by the activation of rules, but it may be used as well to create more complex partitions.
- TakagiSugenoTerm: is a term used when dealing with TakagiSugenoRules, it extends from SingletonTerm so it has a value, and it includes a weight as well, both necessary for this kind of system.

Finally, figure 2.3 shows examples of the linguistic terms, obtained by printing the window from the GUI developed for FuzzyLite v1.0.

2.3 Fuzzy rules

Figure 2.4 shows the class diagram for fuzzy rules and all its related classes included in FuzzyLite v1.0.



(a) `TriangularTerm` (b) `RectangularTerm` (c) `TrapezoidalTerm`

(d) `SingletonTerm` (e) `ShoulderTerm` (left) (f) `ShoulderTerm` (right)

(g) `FunctionTerm` (h) Triangular partitions (i) Triangular + Shoulder partitions

Figure 2.3: Linguistic Terms

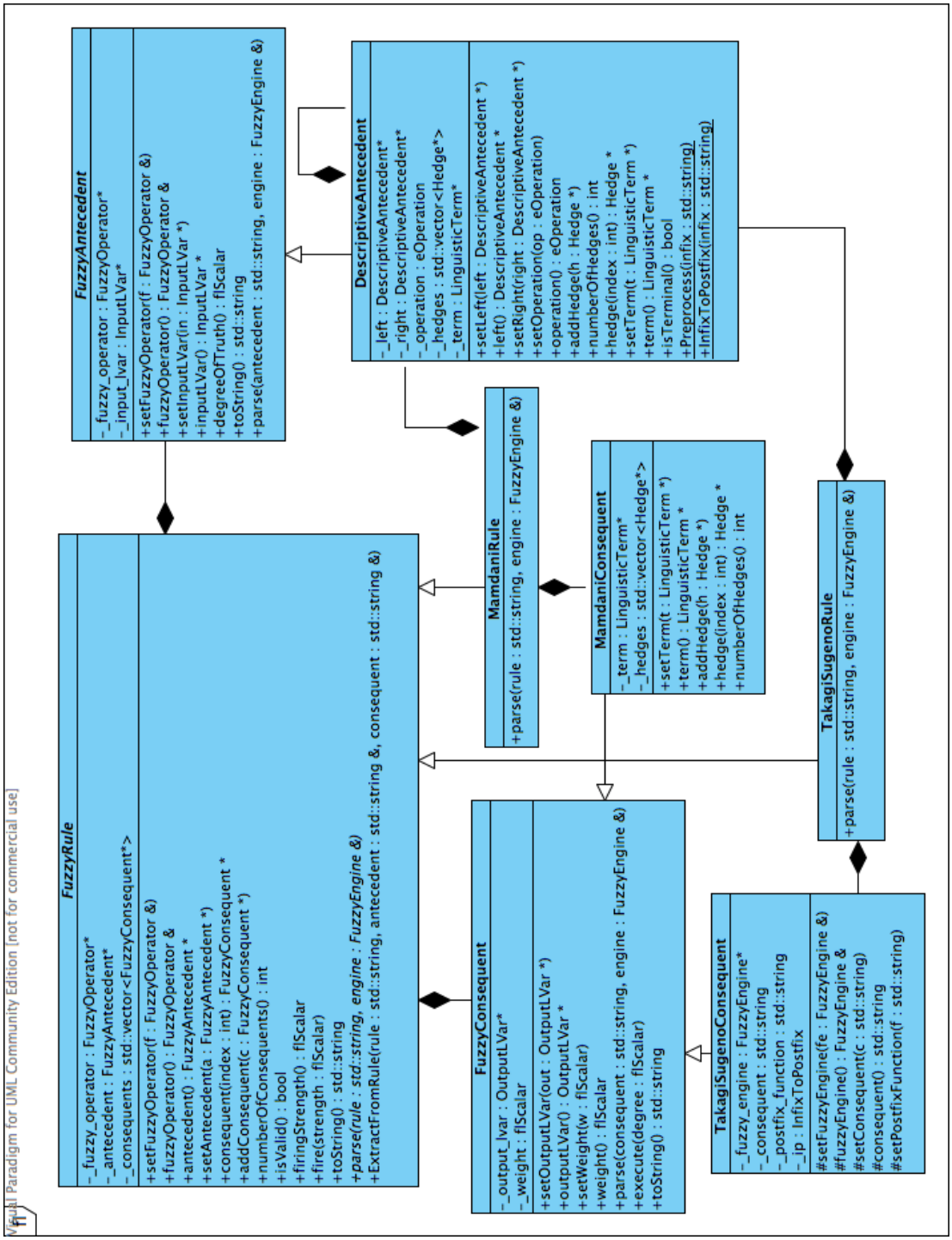


Figure 2.4: Class diagram: Fuzzy rules

2.3.1 FuzzyRule

This abstract class is the base for MamdaniRules and TakagiSugenoRules. It is composed by an antecedent (FuzzyAntecedent) and a list of consequents (FuzzyConsequents). There are two methods that are worth mentioning here: firingStrength() which determines the activation degree of the rule given the inputs in the InputLVars, and fire(flScalar d) which fires the rule with a degree d. When firing the rule, the consequents are modulated according to d and then added to the OutputLVars with the respective weight of the rule. It is also important to remark that parsing the rules is case-sensitive.

FuzzyAntecedent and FuzzyConsequent

These abstract classes represent the antecedent and consequent of a rule, respectively. FuzzyAntecedent has a pointer to the input variable to which it is related in order to access the value of the respective InputLVar, similarly, FuzzyConsequent has a pointer to the output variable to which it is related in order to aggregate the respective modulated linguistic term to the output.

DescriptiveAntecedent

This class is based on FuzzyAntecedent and it is a red-black tree that relates two propositions with an operator, for example, input-1 **is** LOW **and** input-2 **is** GOOD is separated into a left antecedent (input-1 **is** LOW), an operator (**and**), and a right antecedent (input-2 **is** GOOD). Given the recursion of this model, it is possible to create rules of any depth, as it is evaluated bottom-up. This is the class used by MamdaniRule and TakagiSugenoRule.

Hedge

Hedges are modifiers of the propositions (antecedent) and actions (consequent). All hedges must implement the interface Hedge, and must be included when configuring the FuzzyEngine. FuzzyLite v1.0 includes four hedges: not (HedgeNot), somewhat (HedgeSomewhat), very (HedgeVery), any (HedgeAny). In order to use HedgeAny, it is necessary to include a *dummy* linguistic term to comply with the general form of the rules (i.e. *input-1* **is any** LOW); this hedge will always return 1.0, so it does not matter the linguistic term as long as it is within the linguistic variable used. Figure 2.5 shows the class diagram regarding hedges.

MamdaniRule and TakagiSugenoRule

These classes represents the type of rules of a system based on Mamdani's rules, and the one based on Takagi-Sugeno rules, respectively. They extend the abstract class FuzzyRule by implementing the parse() method accordingly. Both classes use DescriptiveAntecedent as the antecedent of the rule, but differ in the consequent: MamdaniRule uses MamdaniConsequent, and TakagiSugenoRule uses TakagiSugenoConsequent.

MamdaniConsequent and TakagiSugenoConsequent

The consequent of a Mamdani rule is of the form: OutputLVar **is** [Hedge*] LinguisticTerm, where [Hedge*] means that none or many hedges may be included.

Conversely, the consequent of a Takagi-Sugeno rule is of the form OutputLVar={*expression*}, where *expression* is a mathematical expression that may include references to the values of the InputLVars, in which case the name of the input variable is used (e.g. "... f_x=0.5 * input-1 + (sin input-2)"). This implementation also allows to include previously computed outputs, in which case the name of the output should be used instead.

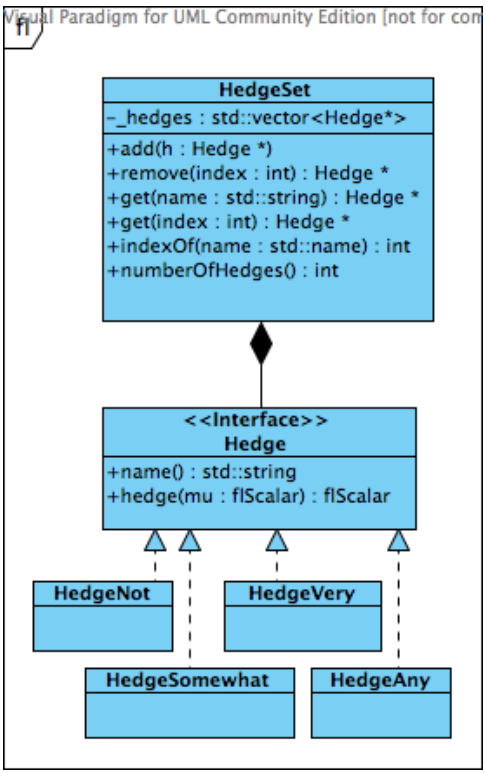


Figure 2.5: Class diagram: Hedges

2.4 Fuzzy engine

The class diagram for this group can be seen in figure 2.6. It contains the class `FuzzyEngine` which is composed by a `FuzzyOperator`, a vector of `RuleBlock`s which contain the `FuzzyRule`s, a `HedgeSet` that contains the hedges registered in the engine, and a vector of `InputLVar`s and `OutputLVar`s for input and output variables (respectively).

The `FuzzyEngine` class contains the whole fuzzy control system. The only methods which might require a bit of explanation are `process( bool )`, and `process(int, bool)`. The former receives a boolean parameter that defaults to `true` and (if true) clears the output of all the output variables. The latter receives an `int` parameter that determines the index of the `RuleBlock` to be fired, and a `bool` that defaults to `true` which defines whether to clear the output of the output variables.

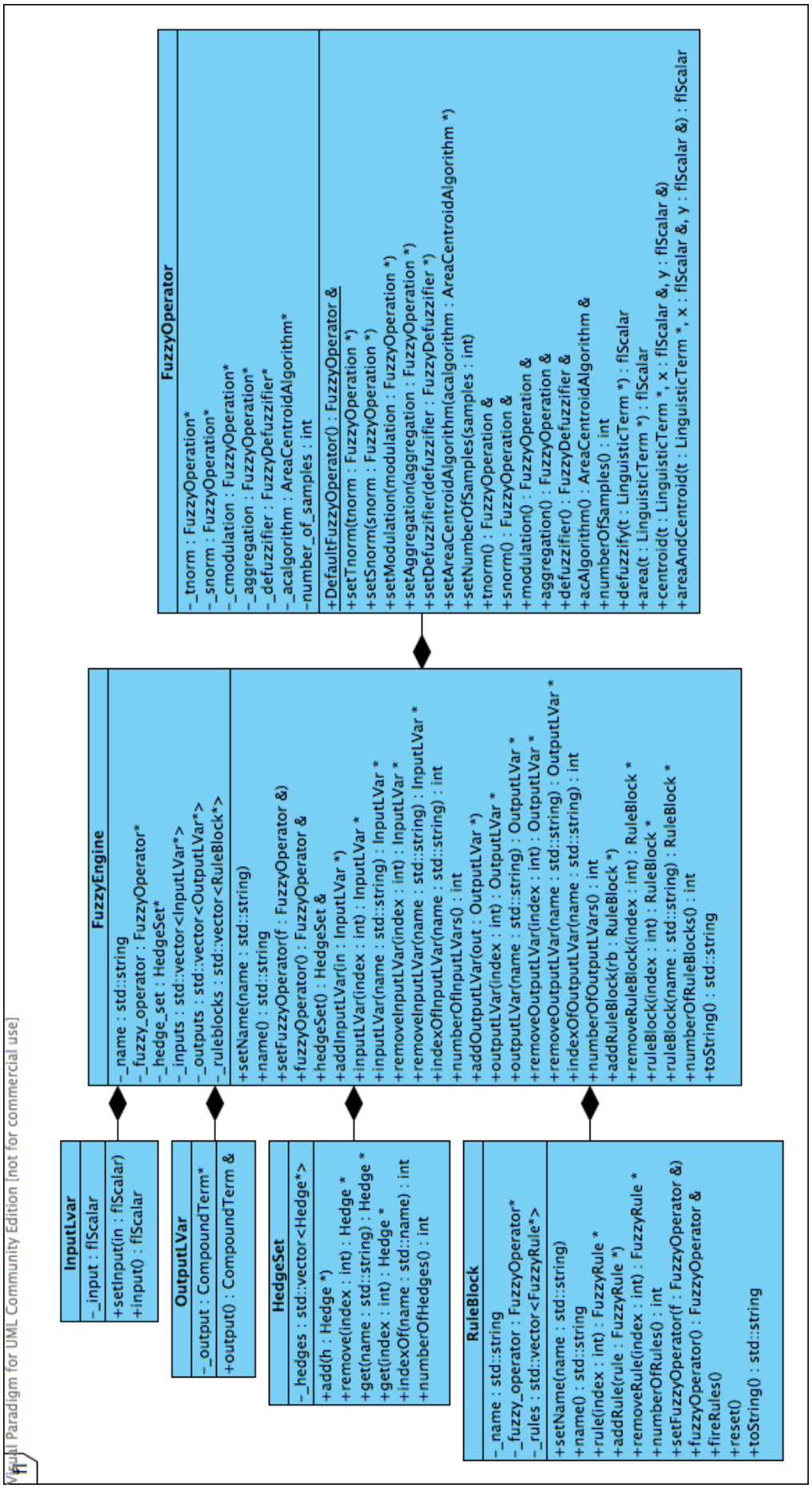


Figure 2.6: Class diagram: Fuzzy engine

2.5 Fuzzy exceptions

This group contains some exceptions that are used within several classes of `FuzzyLite v1.0`. The class `FuzzyException` extends the `std::exception` of the Standard Template Library (STL), and adds additional methods. The other classes derived from `FuzzyException` also contain some static methods to help a bit with the programming. There is not much to say about these exceptions, except to take a look at the code when using them to become familiar. Figure 2.7 shows the class diagram for this group.

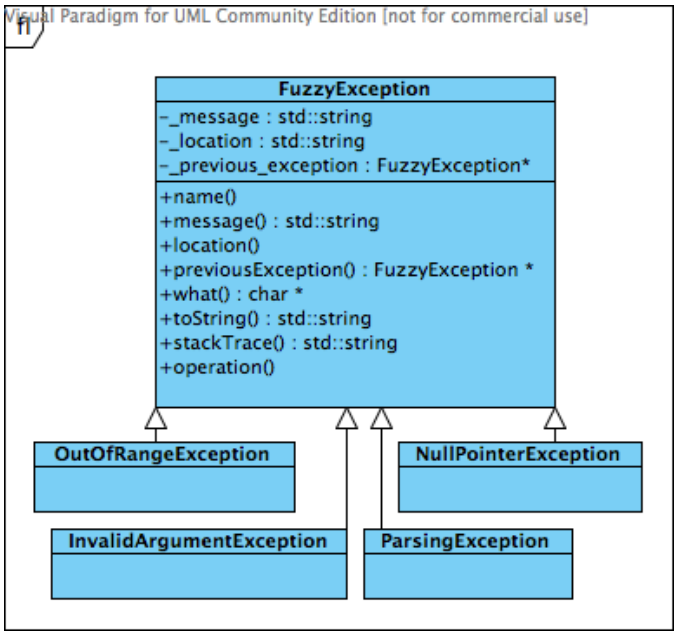


Figure 2.7: Class diagram: Fuzzy exceptions

Chapter 3

Examples

3.1 Example #1: Basic FIS

This is a basic FIS composed of one input variable and one output variable.

```
1   fl::FuzzyEngine engine;
2   engine.hedgeSet().add(new fl::HedgeNot);
3   engine.hedgeSet().add(new fl::HedgeSomewhat);
4   engine.hedgeSet().add(new fl::HedgeVery);
5   fl::InputLVar* energy = new fl::InputLVar("Energy");
6   energy->addTerm(new fl::ShoulderTerm("LOW", 0.25, 0.5, true));
7   energy->addTerm(new fl::TriangularTerm("MEDIUM", 0.25, 0.75));
8   energy->addTerm(new fl::ShoulderTerm("HIGH", 0.50, 0.75, false));
9   engine.addInputLVar(energy);
10
11  fl::OutputLVar* health = new fl::OutputLVar("Health");
12  health->addTerm(new fl::TriangularTerm("BAD", 0.0, 0.50));
13  health->addTerm(new fl::TriangularTerm("REGULAR", 0.25, 0.75));
14  health->addTerm(new fl::TriangularTerm("GOOD", 0.50, 1.00));
15  engine.addOutputLVar(health);
16  fl::RuleBlock* block = new fl::RuleBlock();
17  block->addRule(new fl::MamdaniRule("if Energy is LOW then Health is BAD", engine));
18  block->addRule(new fl::MamdaniRule("if Energy is MEDIUM then Health is REGULAR", engine));
19  block->addRule(new fl::MamdaniRule("if Energy is HIGH then Health is GOOD", engine));
20  engine.addRuleBlock(block);
```

Once the FIS is configured, the control process may begin anytime by setting the input value to the input variables and processing. For example,

```
22  for (fl::flScalar in = 0.0; in < 1.1; in += 0.1) {
23      energy->setInput(in);
24      engine.process();
25      fl::flScalar out = health->output().defuzzify();
26      FL_LOG("Energy=" << in);
27      FL_LOG("Energy is " << energy->fuzzify(in));
28      FL_LOG("Health=" << out);
29      FL_LOG("Health is " << health->fuzzify(out));
30      FL_LOG("--");
31  }
```

The previous code would yield the following results in console (assuming that `FL_USE_LOG` was defined):

```
1 Energy=0
2 Energy is 1.000/LOW + 0.000/MEDIUM + 0.000/HIGH
3 Health=0.249902
4 Health is 1.000/BAD + 0.000/REGULAR + 0.000/GOOD
5 --
6 Energy=0.1
```

```
Energy is 1.000/LOW + 0.000/MEDIUM + 0.000/HIGH
Health=0.249902
Health is 1.000/BAD + 0.000/REGULAR + 0.000/GOOD
--
Energy=0.2
Energy is 1.000/LOW + 0.000/MEDIUM + 0.000/HIGH
Health=0.249902
Health is 1.000/BAD + 0.000/REGULAR + 0.000/GOOD
--
Energy=0.3
Energy is 0.800/LOW + 0.200/MEDIUM + 0.000/HIGH
Health=0.309985
Health is 0.760/BAD + 0.240/REGULAR + 0.000/GOOD
--
Energy=0.4
Energy is 0.400/LOW + 0.600/MEDIUM + 0.000/HIGH
Health=0.394929
Health is 0.420/BAD + 0.580/REGULAR + 0.000/GOOD
--
Energy=0.5
Energy is 0.000/LOW + 1.000/MEDIUM + 0.000/HIGH
Health=0.499902
Health is 0.000/BAD + 1.000/REGULAR + 0.000/GOOD
--
Energy=0.6
Energy is 0.000/LOW + 0.600/MEDIUM + 0.400/HIGH
Health=0.604537
Health is 0.000/BAD + 0.582/REGULAR + 0.418/GOOD
--
Energy=0.7
Energy is 0.000/LOW + 0.200/MEDIUM + 0.800/HIGH
Health=0.689444
Health is 0.000/BAD + 0.242/REGULAR + 0.758/GOOD
--
Energy=0.8
Energy is 0.000/LOW + 0.000/MEDIUM + 1.000/HIGH
Health=0.749902
Health is 0.000/BAD + 0.000/REGULAR + 1.000/GOOD
--
Energy=0.9
Energy is 0.000/LOW + 0.000/MEDIUM + 1.000/HIGH
Health=0.749902
Health is 0.000/BAD + 0.000/REGULAR + 1.000/GOOD
--
Energy=1
Energy is 0.000/LOW + 0.000/MEDIUM + 1.000/HIGH
Health=0.749902
Health is 0.000/BAD + 0.000/REGULAR + 1.000/GOOD
--
```

3.2 Example #2: 3D Pole Balancing

A simulation video of this fuzzy system implemented with `FuzzyLite v1.0` as shown below is available at http://www.youtube.com/watch?v=YOKk8G_5aRA.

```
FuzzyEngine engine("pole-balancing-3d");

InputLVar* anglex = new InputLVar("AngleX");
std::vector<std::string> labels;
labels.push_back("NEAR_0");
labels.push_back("NEAR_45");
labels.push_back("NEAR_90");
```

```
8  labels.push_back("NEAR_135");
9  labels.push_back("NEAR_180");
10 anglex->createTerms(5, LinguisticTerm::MF_SHOULDER, 0, 180, labels);
11 engine.addInputLVar(anglex);
12
13 InputLVar* anglez = new InputLVar("AngleZ");
14 labels.clear();
15 labels.push_back("NEAR_0");
16 labels.push_back("NEAR_45");
17 labels.push_back("NEAR_90");
18 labels.push_back("NEAR_135");
19 labels.push_back("NEAR_180");
20 anglez->createTerms(5, LinguisticTerm::MF_SHOULDER, 0, 180, labels);
21 engine.addInputLVar(anglez);
22
23 OutputLVar* forcex = new OutputLVar("ForceX");
24 labels.clear();
25 labels.push_back("NL");
26 labels.push_back("NS");
27 labels.push_back("ZR");
28 labels.push_back("PS");
29 labels.push_back("PL");
30 forcex->createTerms(5, LinguisticTerm::MF_TRIANGULAR, -1, 1, labels);
31 engine.addOutputLVar(forcex);
32
33 OutputLVar* forcez = new OutputLVar("ForceZ");
34 labels.clear();
35 labels.push_back("NL");
36 labels.push_back("NS");
37 labels.push_back("ZR");
38 labels.push_back("PS");
39 labels.push_back("PL");
40 forcez->createTerms(5, LinguisticTerm::MF_TRIANGULAR, -1, 1, labels);
41 engine.addOutputLVar(forcez);
42
43 RuleBlock* ruleblock = new RuleBlock("Rules");
44 ruleblock->addRule(new MamdaniRule("if AngleX is NEAR_180 then ForceX is NL", engine));
45 ruleblock->addRule(new MamdaniRule("if AngleX is NEAR_135 then ForceX is NS", engine));
46 ruleblock->addRule(new MamdaniRule("if AngleX is NEAR_90 then ForceX is ZR", engine));
47 ruleblock->addRule(new MamdaniRule("if AngleX is NEAR_45 then ForceX is PS", engine));
48 ruleblock->addRule(new MamdaniRule("if AngleX is NEAR_0 then ForceX is PL", engine));
49
50 ruleblock->addRule(new MamdaniRule("if AngleZ is NEAR_180 then ForceZ is NL", engine));
51 ruleblock->addRule(new MamdaniRule("if AngleZ is NEAR_135 then ForceZ is NS", engine));
52 ruleblock->addRule(new MamdaniRule("if AngleZ is NEAR_90 then ForceZ is ZR", engine));
53 ruleblock->addRule(new MamdaniRule("if AngleZ is NEAR_45 then ForceZ is PS", engine));
54 ruleblock->addRule(new MamdaniRule("if AngleZ is NEAR_0 then ForceZ is PL", engine));
55 engine.addRuleBlock(ruleblock);
56
57 FL_LOG(engine.toString());
```

The output from the previous block of code exports the fuzzy system to text using the Fuzzy Controller Language (FCL) as shown below.

```
1  FUNCTION_BLOCK pole-balancing-3d
2
3  VAR_INPUT
4  AngleX: REAL;
5  AngleZ: REAL;
6  END_VAR
7
8  FUZZIFY AngleX
9  TERM NEAR_0 := Shoulder left(0 60);
10 TERM NEAR_45 := Triangular (30 60 90);
```

```
TERM NEAR_90 := Triangular (60 90 120);
TERM NEAR_135 := Triangular (90 120 150);
TERM NEAR_180 := Shoulder right(120 180);
END_FUZZIFY

FUZZIFY AngleZ
TERM NEAR_0 := Shoulder left(0 60);
TERM NEAR_45 := Triangular (30 60 90);
TERM NEAR_90 := Triangular (60 90 120);
TERM NEAR_135 := Triangular (90 120 150);
TERM NEAR_180 := Shoulder right(120 180);
END_FUZZIFY

VAR_OUTPUT
ForceX: REAL
ForceZ: REAL
END_VAR

DEFUZZIFY ForceX
TERM NL := Triangular (-1 -0.666667 -0.333333);
TERM NS := Triangular (-0.666667 -0.333333 5.96046e-08);
TERM ZR := Triangular (-0.333333 5.96046e-08 0.333333);
TERM PS := Triangular (5.96046e-08 0.333333 0.666667);
TERM PL := Triangular (0.333333 0.666667 1);
END_DEFUZZIFY

DEFUZZIFY ForceZ
TERM NL := Triangular (-1 -0.666667 -0.333333);
TERM NS := Triangular (-0.666667 -0.333333 5.96046e-08);
TERM ZR := Triangular (-0.333333 5.96046e-08 0.333333);
TERM PS := Triangular (5.96046e-08 0.333333 0.666667);
TERM PL := Triangular (0.333333 0.666667 1);
END_DEFUZZIFY

RULEBLOCK Rules
RULE 1: if AngleX is NEAR_180 then ForceX is NL;
RULE 2: if AngleX is NEAR_135 then ForceX is NS;
RULE 3: if AngleX is NEAR_90 then ForceX is ZR;
RULE 4: if AngleX is NEAR_45 then ForceX is PS;
RULE 5: if AngleX is NEAR_0 then ForceX is PL;
RULE 6: if AngleZ is NEAR_180 then ForceZ is NL;
RULE 7: if AngleZ is NEAR_135 then ForceZ is NS;
RULE 8: if AngleZ is NEAR_90 then ForceZ is ZR;
RULE 9: if AngleZ is NEAR_45 then ForceZ is PS;
RULE 10: if AngleZ is NEAR_0 then ForceZ is PL;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

3.3 Example 3: Approximating a function

It is also possible to create a fuzzy system to approximate a function. For example, if we were to approximate the function $sin(x)/x$, we could do so by using the with the following code.

```
FuzzyOperator& op = FuzzyOperator::DefaultFuzzyOperator();
op.setDefuzzifier(new TakagiSugenoDefuzzifier);
FuzzyEngine engine("approximation", op);

fl::InputLVar* x = new fl::InputLVar("x");
x->addTerm(new fl::TriangularTerm("NEAR_1", 0, 2));
x->addTerm(new fl::TriangularTerm("NEAR_2", 1, 3));
x->addTerm(new fl::TriangularTerm("NEAR_3", 2, 4));
```

```
x->addTerm(new fl::TriangularTerm("NEAR_4", 3, 5));
x->addTerm(new fl::TriangularTerm("NEAR_5", 4, 6));
x->addTerm(new fl::TriangularTerm("NEAR_6", 5, 7));
x->addTerm(new fl::TriangularTerm("NEAR_7", 6, 8));
x->addTerm(new fl::TriangularTerm("NEAR_8", 7, 9));
x->addTerm(new fl::TriangularTerm("NEAR_9", 8, 10));
engine.addInputLVar(x);

fl::OutputLVar* f_x = new fl::OutputLVar("f_x");
f_x->addTerm(new fl::FunctionTerm("function", "(sin x) / x", 0, 10));
engine.addOutputLVar(f_x);

fl::RuleBlock* block = new fl::RuleBlock();
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_1 then f_x=0.84", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_2 then f_x=0.45", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_3 then f_x=0.04", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_4 then f_x=-0.18", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_5 then f_x=-0.19", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_6 then f_x=-0.04", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_7 then f_x=0.09", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_8 then f_x=0.12", engine));
block->addRule(new fl::TakagiSugenoRule("if x is NEAR_9 then f_x=0.04", engine));

engine.addRuleBlock(block);

int n = 40;
flScalar mse = 0;
for (fl::flScalar in = x->minimum(); in < x->maximum() ;
        in += (x->minimum() + x->maximum()) / n) {
    x->setInput(in);
    engine.process();
    flScalar expected = f_x->term(0)->membership(in);
    flScalar obtained = f_x->output().defuzzify();
    flScalar se = (expected - obtained) * (expected - obtained);
    mse += isnan(se) ? 0 : se;
    FL_LOG("x=" << in << "\texpected_out=" << expected << "\tobtained_out=" << obtained
            << "\tse=" << se);
}
FL_LOG("MSE=" << mse / n);
```

The output of this piece of code is the following.

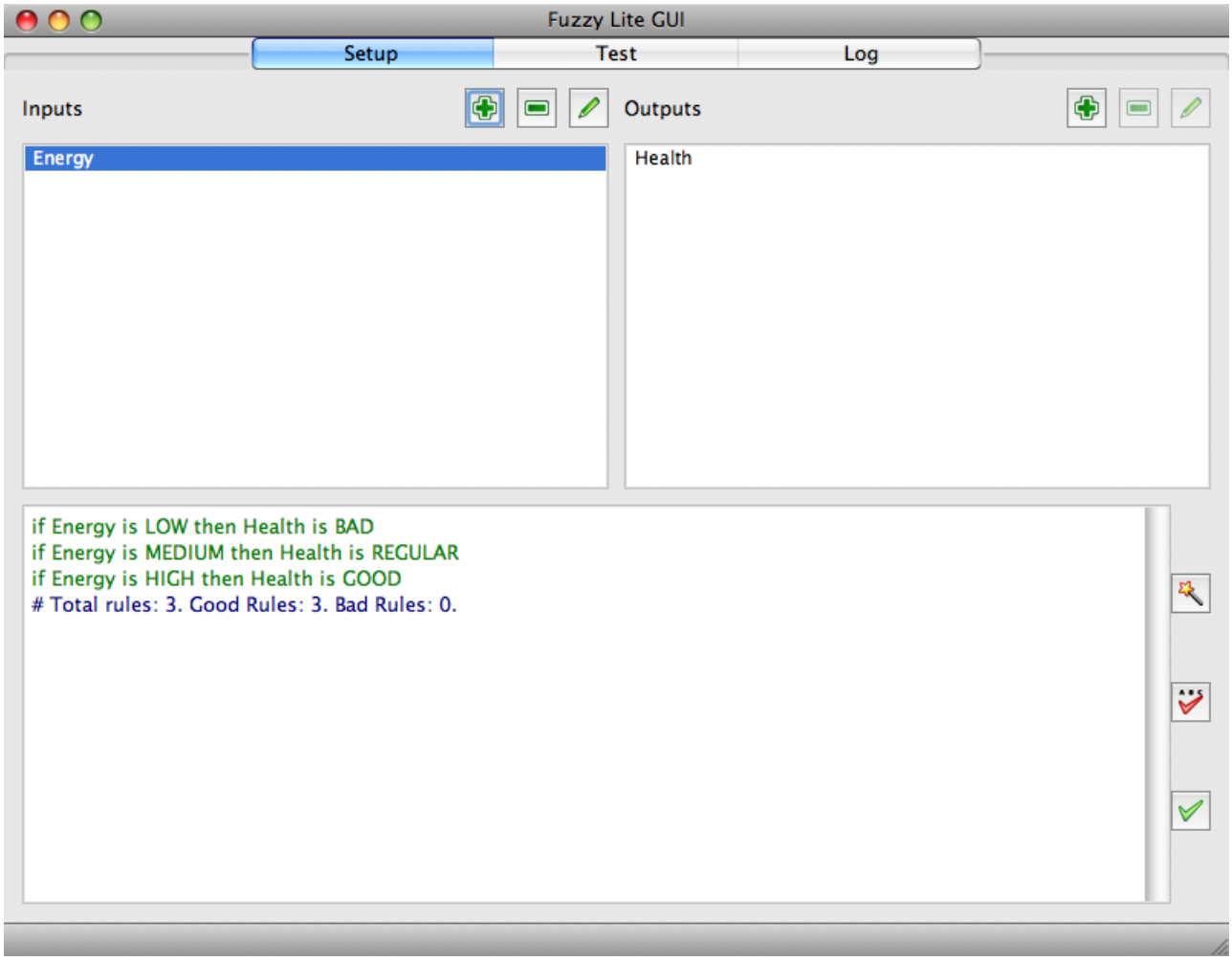
```
x=0.000     expected_out=nan       obtained_out=nan       se=nan
x=0.250     expected_out=0.990       obtained_out=0.840       se=0.022
x=0.500     expected_out=0.959       obtained_out=0.840       se=0.014
x=0.750     expected_out=0.909       obtained_out=0.840       se=0.005
x=1.000     expected_out=0.841       obtained_out=0.840       se=0.000
x=1.250     expected_out=0.759       obtained_out=0.743       se=0.000
x=1.500     expected_out=0.665       obtained_out=0.645       se=0.000
x=1.750     expected_out=0.562       obtained_out=0.547       se=0.000
x=2.000     expected_out=0.455       obtained_out=0.450       se=0.000
x=2.250     expected_out=0.346       obtained_out=0.347       se=0.000
x=2.500     expected_out=0.239       obtained_out=0.245       se=0.000
x=2.750     expected_out=0.139       obtained_out=0.142       se=0.000
x=3.000     expected_out=0.047       obtained_out=0.040       se=0.000
x=3.250     expected_out=-0.033       obtained_out=-0.015       se=0.000
x=3.500     expected_out=-0.100       obtained_out=-0.070       se=0.001
x=3.750     expected_out=-0.152       obtained_out=-0.125       se=0.001
x=4.000     expected_out=-0.189       obtained_out=-0.180       se=0.000
x=4.250     expected_out=-0.211       obtained_out=-0.183       se=0.001
```

```
x=4.500        expected_out=-0.217        obtained_out=-0.185        se=0.001
x=4.750        expected_out=-0.210        obtained_out=-0.188        se=0.001
x=5.000        expected_out=-0.192        obtained_out=-0.190        se=0.000
x=5.250        expected_out=-0.164        obtained_out=-0.153        se=0.000
x=5.500        expected_out=-0.128        obtained_out=-0.115        se=0.000
x=5.750        expected_out=-0.088        obtained_out=-0.078        se=0.000
x=6.000        expected_out=-0.047        obtained_out=-0.040        se=0.000
x=6.250        expected_out=-0.005        obtained_out=-0.007        se=0.000
x=6.500        expected_out=0.033        obtained_out=0.025        se=0.000
x=6.750        expected_out=0.067        obtained_out=0.058        se=0.000
x=7.000        expected_out=0.094        obtained_out=0.090        se=0.000
x=7.250        expected_out=0.114        obtained_out=0.098        se=0.000
x=7.500        expected_out=0.125        obtained_out=0.105        se=0.000
x=7.750        expected_out=0.128        obtained_out=0.112        se=0.000
x=8.000        expected_out=0.124        obtained_out=0.120        se=0.000
x=8.250        expected_out=0.112        obtained_out=0.100        se=0.000
x=8.500        expected_out=0.094        obtained_out=0.080        se=0.000
x=8.750        expected_out=0.071        obtained_out=0.060        se=0.000
x=9.000        expected_out=0.046        obtained_out=0.040        se=0.000
x=9.250        expected_out=0.019        obtained_out=0.040        se=0.000
x=9.500        expected_out=-0.008        obtained_out=0.040        se=0.002
x=9.750        expected_out=-0.033        obtained_out=0.040        se=0.005
MSE=0.001
```

Appendix

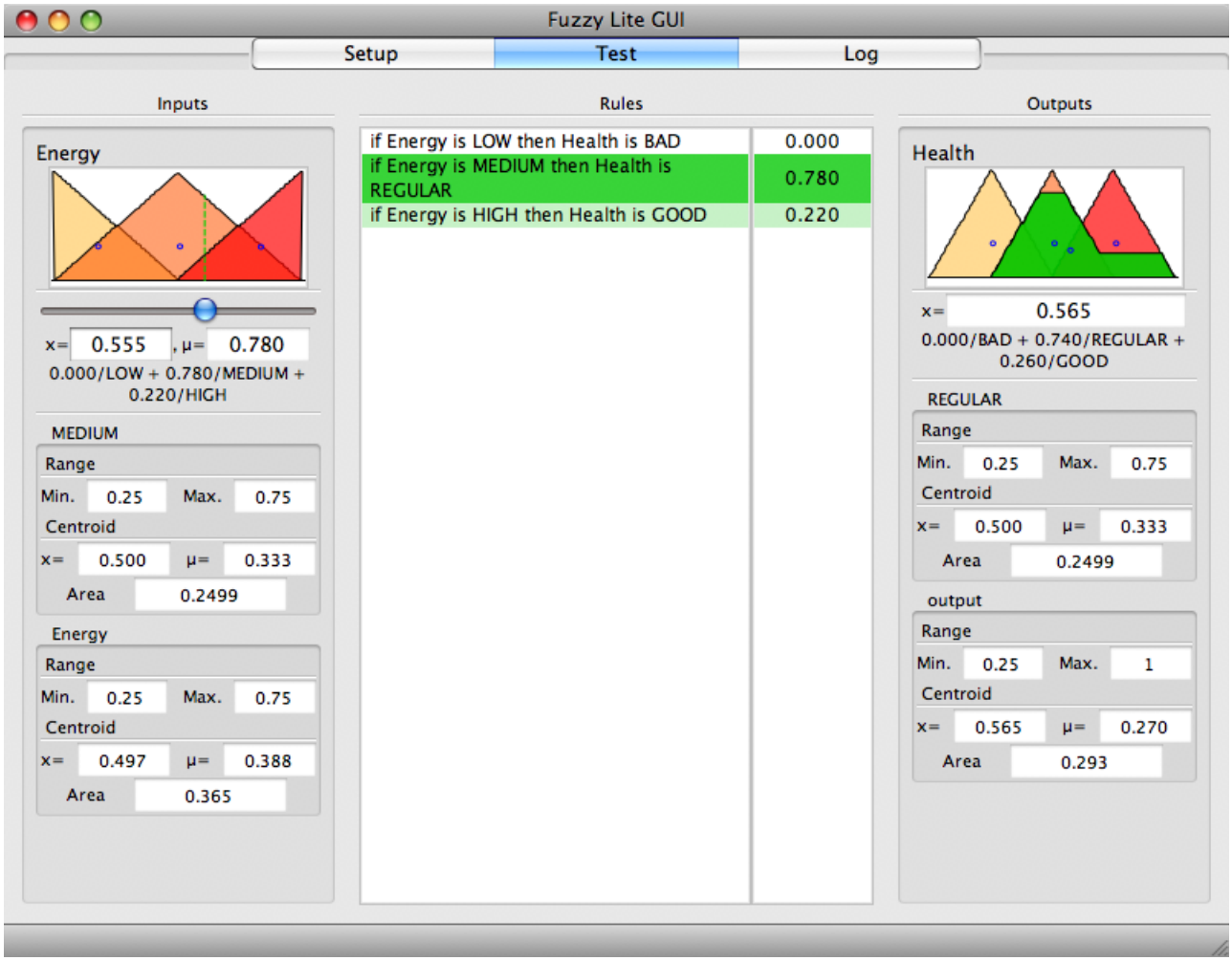
GUI Setup

The following figure shows the Setup part of the GUI build for `FuzzyLite v1.0`.



GUI Setup

The following figure shows the Setup part of the GUI build for `FuzzyLite v1.0`.



License of FuzzyLite v1.0

```
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
```

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS