1. Introduction

This report describes the design and implementation of a small domain-specific language (DSL) in Scala, called the FuzzyLogicOperator. The DSL supports operations for logical gates such as addition, multiplication, xor, union, intersection. It is designed for users to describe and evaluate boolean functions using variables and scopes. Also, Scalatest tests are described to prove the correctness of this DSL.

2. Language Design

Fuzzy Gate Operations

Fuzzy gates operations are designed using the FuzzyGateOperation trait, which is extended by case classes such as ADD, MULT, XOR, round.

ADD: Performs fuzzy addition of two inputs, capped at 1.

MULT: Performs fuzzy multiplication (reflects how two fuzzy sets "intersect" in strength).

XOR: Computes the absolute difference between two inputs.

Round: round the value into a certain number of decimals (1 by default)

Example of syntax:

val gate1 = FuzzyGate("gate1", ADD(Input("A"), Input("B")))

This defines a fuzzy gate named gate1, which adds the inputs A and B. The values of A and B are provided later when evaluating the gate.

Fuzzy Set Operations

Fuzzy sets are defined as map of elements, where each element has a name and a value. Fuzzy sets operations includes ADD, MULT, XOR, UNION, INTERSECTION, COMPLEMENT, round. They are applied element-wise.

ADD: Adds the values of corresponding elements in two sets, capped at 1.

MULT: Multiplies the values of corresponding elements in two sets.

XOR: Computes the absolute difference in membership values of corresponding elements in two sets.

Round: round the value into a certain number of decimals (1 by default)

UNION: Takes the maximum value of each element between two sets.

INTERSECTION: Takes the minimum value of each element between two sets.

COMPLEMENT: Inverts the value of each element (1 - value).

alphaCut: Extracts elements from a fuzzy set whose values are no less than a specified threshold.

Example Syntax:

val setA = FuzzySet("A", List(Element("x1", 0.5), Element("x2", 0.7)))

val setB = FuzzySet("B", List(Element("x1", 0.3), Element("x2", 0.8)))

val unionResult = FuzzySetOperations.union(setA, setB)

This defines two fuzzy sets A and B, and then computes the union of these sets, which should be ("x1", 0.3), ("x2", 0.7).

Evaluation Model:

The DSL uses ScalaTest to test the behavior of the fuzzy operations

FuzzyGate Tests

There are add, mult, xor operations test between two values. The composite gate test ensures the nested gate operations (combination of add, mult, xor) is correct. There is also an Error Handling test check verify this DSL could throw an exception when the input of a gate is missing. Another test is used for verifying the 1 cap on addition and variable override.

FuzzySet Tests

I take the example data in the hw1 description, and test the add, mult, xor, intersect, union, complement, and alpha-cut for two fuzzy set.

3. Semantics

Static Semantics

- Type Checking: The DSL ensures that the expressions are type-safe. When an `Int` assigned to the gate, the result is implicitly converted to a `Double`. And the compiler would prevent assigning value other than numbers to gates.

There is also an input check defined in the FuzzyGateOperation trait. If an Input in a gate has not been assigned a value, the input function will throw an error during evaluation.

val compositeGate = FuzzyGate("compositeGate", XOR(gate1.operation, Input("C")))

When C is not defined beforehand, it would return

Input values for compositeGate not found.

As for fuzzyset, when an element only appears in one fuzzy set, that value is assigned to 0.0 in the set.

Handling Floating-Point Precision

Since the DSL uses Double for values, and Double operations would cause round problems in Scala. (e.g., 0.2999999999 instead of 0.3). To prevent this, all results are rounded to using a custom round function mentioned above.

Dynamic Semantics

- Evaluation: Expressions are evaluated recursively. For example, in the expression

val gate1 = FuzzyGate("gate1", ADD(MULT(Input("A"), Input("B")), Input("C"))) Scope(gate1, (Input("A"), 0.5))

Scope(gate1, (Input("B"), 0.7))

Scope(gate1, (Input("C"), 0.2))

the multiplication is evaluated first, followed by the addition.

For fuzzy sets, operations are applied element-by-element. Fuzzy set operations are performed would iterating over matched elements and apply their operations on the values.

val setA = FuzzySet("A", List(Element("x1", 0.5), Element("x2", 0.7)))

val setB = FuzzySet("B", List(Element("x1", 0.3), Element("x2", 0.4)))

val addResult = FuzzySetOperations.add(setA, setB)

The result is List(Element("x1", 0.8), Element("x2", 1.0))

4.Implementation

This DSL is implemented in Scala using case classes and pattern matching to represent fuzzy gates, fuzzy sets, and fuzzy operations evaluate single gate operations and fuzzy set operation. Each operation is implemented as a case class extending a base trait, and an evaluate function is responsible for recursively computing the result.

FuzzyGateOperation Trait

This trait is the base for all fuzzy gate operations. ADD, MUL, XOR and Input are defined extending on this trait.

sealed trait FuzzyGateOperation  
  
case class Input(name: String, value: Option[Double] = None) extends FuzzyGateOperation  
case class ADD(a: FuzzyGateOperation, b: FuzzyGateOperation) extends FuzzyGateOperation  
case class MULT(a: FuzzyGateOperation, b: FuzzyGateOperation) extends FuzzyGateOperation  
case class XOR(a: FuzzyGateOperation, b: FuzzyGateOperation) extends FuzzyGateOperation

FuzzyOperations

This object contains the logic of add, mult and xor. Every value is rounded to 1 decimal by default, and the value of add is capped at 1.

object FuzzyOperations {  
 def round(value: Double, precision: Int = 2): Double = {  
 val scale = Math.*pow*(10, precision)  
 Math.*round*(value \* scale) / scale  
 }  
  
 def add(a: Double, b: Double): Double = Math.*min*(1.0, *round*(a + b))  
 def mult(a: Double, b: Double): Double = *round*(a \* b)  
 def xor(a: Double, b: Double): Double = *round*(Math.*abs*(a - b))  
}

FuzzyGateEvaluator

This object is used for evaluating the fuzzy gate operations. It recursively applies operations like ADD, MULT, and XOR to Input from a scope by using pattern matching. The evaluation process checks for undefined inputs and throws an exception if any input is missing with the Input function.

object FuzzyGateEvaluator {

def evaluate(gate: FuzzyGate, inputs: Map[String, Double]): Double = {

def evalOperation(op: FuzzyGateOperation): Double = op match {

case Input(name, \_) => inputs.getOrElse(name, throw new IllegalArgumentException(s"Input $name is not defined"))

case ADD(a, b) => FuzzyOperations.add(evalOperation(a), evalOperation(b))

case MULT(a, b) => FuzzyOperations.mult(evalOperation(a), evalOperation(b))

case XOR(a, b) => FuzzyOperations.xor(evalOperation(a), evalOperation(b))

}

evalOperation(gate.operation)

}

}

FuzzySet and FuzzySetOperations

FuzzySet is a list of elements. Element is a pair with name and value.

FuzzySetOperations contains the logic of set operations: add, mul, xor, union, intersect, complement and alpha-cut. Similarly, all values are rounded to 1 decimal. In the DLS, the missing elements in each set are filled with 0.0.

case class Element(name: String, value: Double)  
  
case class FuzzySet(name: String, elements: List[Element]) {  
 def eval(): List[Element] = elements  
}  
  
object FuzzySetOperations {  
  
 def round(value: Double, precision: Int = 2): Double = {  
 val scale = Math.*pow*(10, precision)  
 Math.*round*(value \* scale) / scale  
 }  
  
 def elementMap(set: FuzzySet): Map[String, Double] = {  
 set.elements.map(e => e.name -> e.value).toMap  
 }  
  
 def union(setA: FuzzySet, setB: FuzzySet): FuzzySet = {  
 val mapA = *elementMap*(setA)  
 val mapB = *elementMap*(setB)  
  
 val allKeys = mapA.keySet ++ mapB.keySet  
 val newElements = allKeys.map { key =>  
 val valueA = mapA.getOrElse(key, 0.0)  
 val valueB = mapB.getOrElse(key, 0.0)  
 Element(key, *round*(Math.*max*(valueA, valueB)))  
 }.toList  
  
 FuzzySet(s"**$**{setA.name}\_UNION\_**$**{setB.name}", newElements)  
 }  
  
 def intersection(setA: FuzzySet, setB: FuzzySet): FuzzySet = {  
 val mapA = *elementMap*(setA)  
 val mapB = *elementMap*(setB)  
  
 val allKeys = mapA.keySet ++ mapB.keySet  
 val newElements = allKeys.map { key =>  
 val valueA = mapA.getOrElse(key, 0.0)  
 val valueB = mapB.getOrElse(key, 0.0)  
 Element(key, *round*(Math.*min*(valueA, valueB)))  
 }.toList  
  
 FuzzySet(s"**$**{setA.name}\_INTERSECTION\_**$**{setB.name}", newElements)  
 }  
  
 def complement(setA: FuzzySet): FuzzySet = {  
 val newElements = setA.elements.map {  
 case Element(nameA, valueA) =>  
 Element(nameA, *round*(1 - valueA))  
 }  
 FuzzySet(s"**$**{setA.name}\_COMPLEMENT", newElements)  
 }  
  
 def add(setA: FuzzySet, setB: FuzzySet): FuzzySet = {  
 val mapA = *elementMap*(setA)  
 val mapB = *elementMap*(setB)  
  
 val allKeys = mapA.keySet ++ mapB.keySet  
 val newElements = allKeys.map { key =>  
 val valueA = mapA.getOrElse(key, 0.0)  
 val valueB = mapB.getOrElse(key, 0.0)  
 Element(key, *round*(Math.*min*(1.0, valueA + valueB)))  
 }.toList  
  
 FuzzySet(s"**$**{setA.name}\_ADD\_**$**{setB.name}", newElements)  
 }  
  
 def mult(setA: FuzzySet, setB: FuzzySet): FuzzySet = {  
 val mapA = *elementMap*(setA)  
 val mapB = *elementMap*(setB)  
  
 val allKeys = mapA.keySet ++ mapB.keySet  
 val newElements = allKeys.map { key =>  
 val valueA = mapA.getOrElse(key, 0.0)  
 val valueB = mapB.getOrElse(key, 0.0)  
 Element(key, *round*(valueA \* valueB))  
 }.toList  
  
 FuzzySet(s"**$**{setA.name}\_MULT\_**$**{setB.name}", newElements)  
 }  
  
 def xor(setA: FuzzySet, setB: FuzzySet): FuzzySet = {  
 val mapA = *elementMap*(setA)  
 val mapB = *elementMap*(setB)  
  
 val allKeys = mapA.keySet ++ mapB.keySet  
 val newElements = allKeys.map { key =>  
 val valueA = mapA.getOrElse(key, 0.0)  
 val valueB = mapB.getOrElse(key, 0.0)  
 Element(key, *round*(Math.*max*(valueA, valueB) - Math.*min*(valueA, valueB)))  
 }.toList  
  
 FuzzySet(s"**$**{setA.name}\_XOR\_**$**{setB.name}", newElements)  
 }  
   
 def alphaCut(setA: FuzzySet, alpha: Double): List[String] = {  
 setA.elements.collect {  
 case Element(nameA, valueA) if valueA >= alpha => nameA  
 }  
 }  
}

FuzzyLogicDSL Object

The FuzzyLogicDSL object handles the creating, scoping, evaluation and testing of fuzzy gates within the DSL. It uses gates map to manage the definitions of fuzzy gates. And it use the inputScope map to handle the scope of inputs for each gate. TestGate utilizes these functions above and compare the result with the expectedResult defined beforehands.

object FuzzyLogicDSL {

var gates: Map[String, FuzzyGate] = Map()

var inputScope: Map[String, Map[String, Double]] = Map()

def Assign(gate: FuzzyGate): Unit = {

gates += (gate.name -> gate)

}

def AssignInput(gateName: String, input: Input, value: Double): Unit = {

val currentScope = inputScope.getOrElse(gateName, Map())

inputScope += (gateName -> (currentScope + (input.name -> value)))

}

def Scope(gate: FuzzyGate, inputAssignment: (Input, Double)): Unit = {

AssignInput(gate.name, inputAssignment.\_1, inputAssignment.\_2)

}

def TestGate(gateName: String, expectedResult: Double): Boolean = {

val gate = gates.getOrElse(gateName, throw new IllegalArgumentException(s"Gate $gateName not found"))

val inputValues = inputScope.getOrElse(gateName, throw new IllegalArgumentException(s"Input values for $gateName not found"))

val result = FuzzyGateEvaluator.evaluate(gate, inputValues)

result == expectedResult

}

}

Tools and Libraries

Scala: It is the language used for the DSL. Case classes and pattern matching are essential for representing and evaluating fuzzy gates and operations.

ScalaTest: It is used for convenient unit testing.

Standard Libraries: Math library is used for calculation.

5. Type System

The type system supports both integer and floating-point values within the range from 0 to 1 for the value of the gate, but all values are internally represented as `Double` for simplicity. And round functions are implemented to avoid round problems. Name of the element are all String type.