Fuzzy Logic in the Functional Paradigm

P15194468

Sam Fuller

2019

# Abstract

Summarise the project

One short paragraph

# Acknowledgements

John North – De Montfort University – Project Supervisor

David Elizondo – De Montfort University – Tutor for AI Programming MSc Module

Francisco Chiclana – De Montfort University – Tutor for Fuzzy Logic MSc Module

# Dictionary

|  |  |
| --- | --- |
| Term | Definition |
| Element | The program is separated into elements, an element has an identifier an assignment operator, and an output. |
| Identifier | 1. The name of a function, it is used when defining and calling a function. |
| Function | All elements with the same identifier represent one function. |
| Argument | A value passed to a function; all arguments are immutable. |

Contents

[Abstract 1](#_Toc19635341)

[Acknowledgements 1](#_Toc19635342)

[Dictionary 1](#_Toc19635343)

[1 Introduction 4](#_Toc19635344)

[2 The Functional Language 5](#_Toc19635345)

[2.1 Background 5](#_Toc19635346)

[2.2 Functionality 5](#_Toc19635347)

[2.2.1 Types 5](#_Toc19635348)

[2.2.2 Operators 5](#_Toc19635349)

[2.2.3 Function Syntax 7](#_Toc19635350)

[2.2.4 Argument Syntax 7](#_Toc19635351)

[2.2.5 Caching 8](#_Toc19635352)

[2.2.6 Calling Functions 8](#_Toc19635353)

[2.3 Testing 8](#_Toc19635354)

[2.4 Examples 8](#_Toc19635355)

[2.4.1 The Fibonacci Sequence 8](#_Toc19635356)

[2.4.2 The Fizz Buzz Game 8](#_Toc19635357)

[3 The Compiler 10](#_Toc19635358)

[3.1 Background 10](#_Toc19635359)

[3.2 Functionality 10](#_Toc19635360)

[3.2.1 Compiler.pl 10](#_Toc19635361)

[3.2.2 LexicalParser.pl 10](#_Toc19635362)

[3.2.3 SyntaxParser.pl 10](#_Toc19635363)

[3.2.4 JSGenerator.pl 10](#_Toc19635364)

[3.2.5 function.js 10](#_Toc19635365)

[3.3 Testing 10](#_Toc19635366)

[4 The Fuzzy Library 11](#_Toc19635367)

[4.1 Background 11](#_Toc19635368)

[4.2 Functionality 11](#_Toc19635369)

[4.2.1 Fuzzy Membership Functions 11](#_Toc19635370)

[4.2.2 Fuzzy Logic Rules 11](#_Toc19635371)

[4.2.3 Fuzzy Logic Analysis 11](#_Toc19635372)

[4.3 Testing 11](#_Toc19635373)

[4.4 Example 11](#_Toc19635374)

[5 Conclusion 12](#_Toc19635375)

[6 References 13](#_Toc19635376)

[7 Appendices 14](#_Toc19635377)

# Introduction

# The Functional Language

It’s a functional language

Everything is defined as a function, functions can have arguements

All arguments and functions are immutable

It compiles to javascript

This allows it to use HTML + CSS as an output

It should not be thought of as a functional replacement for javascript but a functional addition to javascript

It compiles to javascript, and allows for both its functions to be called in javascript and javascript functions to be called in the code

## Background

Other functional languages

Languages that compile to JavaScript

## Functionality

### Types

The language is dynamically typed using JavaScript’s type definition to allow arguments to be passed between the languages.

There is a heavy focus on arrays, with many operators and functions for them

### Operators

There are many operators in this language, many of them use JavaScript equivalents, some of them use pre-written functions

#### Assignment Operators

|  |  |  |
| --- | --- | --- |
| Operator | Name | Description |
| = | Equals | Lazy assignment, calculated results are stored in memory. |
| <- | Gets | Lazy assignment, calculated results are not stored in memory. |

Assignment operators are used in element to separate the input and output and define if the output will be cached in memory. Every element will have exactly one assignment operator.

#### Mathematical Operators

|  |  |
| --- | --- |
| Operator | Name |
| a+b | Addition |
| a-b | Subtraction |
| a\*b | Multiplication |
| a/b | Division |
| a%b | Modulus |
| \*\* | Exponentiation |

Mathematical operators are the same as JavaScript’s mathematical operators, this does not make compiling them easier and the operators could be changed to compile differently, however; it does make the language more consistent with JavaScript.

#### Logical Operators

|  |  |
| --- | --- |
| Operator | Name |
| a&&b | And |
| a||b | Or |
| !a | Not |

Logical operators are also equal to JavaScript’s Logical operators, this has the same reasoning as the mathematical operators. These operators accept one or two Booleans and output one boolean

#### Relational Operators

|  |  |
| --- | --- |
| Operator | Name |
| a>b | Greater than |
| a<b | Less than |
| a>=b | Greater than or equal to |
| a<=b | Less than or equal to |
| a==b | Equal to |
| a!=b | Not equal to |

Relational operators, like both mathematical and logical operators, are the same as JavaScript’s relational operators. These operators accept two values and return a Boolean.

#### Array Operators

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Description | Input | Output |
| [a..b] | Generates a List of all integers between a and b inclusive | a and b are numbers | Array |
| #a | Length of an array | a is an array | Number |
| a:b | Concatenation of two arrays | a and b are arrays | Array |
| ::a | Concatenation of a array | a is an array | String |
| :+a | Sum of an array | a is an array | Number |
| :\*a | Product of an array | a is an array | Number |
| :&a | Returns true element if ALL elements are true | a is an array | Boolean |
| :|a | Returns true element if ANY element is true | a is an array | Boolean |

Array operators do not have a JavaScript equivalent, and will instead be replaced by function calls. The array operators perform actions to arrays or create arrays.

#### Layout

|  |  |
| --- | --- |
| Operator | Description |
| /\*\* | Function documentation comment opener |
| /\* | Generic comment opener |
| \*/ | Generic comment closer |
| , | Separates arguments |
| | | Where, used to define arguments in inputs |
| ; | End of element |
| ( |  |
| ) |  |

The layout operators are used for comments, to break up code and for syntax reasons.

### Function Syntax

Single element functions, recursive functions, multiple element functions

An element has an identifier, possible inputs, an assignment operator then an output

Identifier(input1, input2) = input1 + input2;

An element can have multiple arguments separated by commas

The assignment operator can either be a = (enables caching) <- (disables caching)

The output must evaluate to a value that can be returned, including null

An identifier must start with only the English characters A-Z and a-z but may contain a digit after the first character, only English letters and digits can be used in an identifier. The same is true of argument names.

### Argument Syntax

Identifier(input1, input2) = input1 + input2;

An element can have multiple arguments separated by commas

Identifier = 3;

An element can have no arguments, the brackets are not needed

Identifier(5) = 50;

An element can have an argument that is expected to be a value, in this case the element will only trigger if the first argument is equal to 5, this can work with strings too

Identifier(x>10) <- x^3;

An element can have an argument with Boolean logic, this assumes the argument is the first item to appear

Identifier(x| even(x)) <- x/2;

The | operator allows for an argument to appear embedded in the Boolean logic, the argument name appears before, the Boolean test appears after.

### Caching

If caching is on for an element, the result will be stored in cache

The cache is effectively filled with elements with defined arguments

Identifier(5) = 50;

### Calling Functions

Both functions in this language and javascript can be called from the code. This is done identically to how the element is defined (Identifier(5) or Identifier). The compiler determines if a function is internal or external and acts accordingly.

## Testing

The language does not need to be tested; it is an abstract idea not an actual program. The compiler is the program that uses the language and compiles it to JavaScript therefore the compiler shall be tested.

## Examples

### The Fibonacci Sequence

/\*\*Fibonacci sequence is declared recursively\*/

/\*declaring known fib numbers\*/

fib(0) <- 0;

fib(1) <- 1;

/\*declaring unknown fib numbers recursively\*/

fib(x) = fib(x-1) + fib(x-2);

The Fibonacci sequence is the sequence of numbers where each element is defined as the sum of the previous two. The second to fifth Fibonacci numbers are 1, 2, 3, 5, 8. This example shows recursion within the program, the final element has caching enabled to significantly speed this recursion up. The output of this program is the element of the Fibonacci sequence in x position.

### The Fizz Buzz Game

/\*\*defining fizz recursively\*/

fizz(x<3) <- "";

fizz(3) <- "fizz";

fizz(x) <- fizz(x-3);

/\*\*defining buzz recursively\*/

buzz(x<5) <- "";

buzz(5) <- "buzz";

buzz(x) <- buzz(x-5);

/\*\*the output is the concatenation of both fizz and buzz\*/

stringout(x) = fizz(x)+buzz(x);

/\*\*if the cardinality of out for x is 0, return x, else return the concatenation of out\*/

singleout(x|stringout(x) == "") = x;

singleout(x) = stringout(x);

fizzbuzz(x>0) <- mapsingleout([0..x]);

The Fizz Buzz game is a sequence, where each multiple of 3 is replaced with fizz, each multiple of 5 is replaced with buzz and multiples of both are replaced with fizzbuzz. The first 20 values of the sequence are: 1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz, 16, 17, fizz ,19, buzz. This example shows string and list manipulation within the language.

# The Compiler

## Background

## Functionality

### Compiler.pl

### LexicalParser.pl

### SyntaxParser.pl

### JSGenerator.pl

### function.js

## Testing

# The Fuzzy Library

## Background

## Functionality

### Fuzzy Membership Functions

#### Triangular Membership

#### Trapezoidal Membership

#### Gaussian Membership

### Fuzzy Logic Rules

### Fuzzy Logic Analysis

## Testing

## Example

# Conclusion

# References

# Appendices