Vaasan yliopisto
UNIVERSITY OF VAASA
OSUVA Open Science

This is a self-archived – parallel published version of this article in the publication archive of the University of Vaasa. It might differ from the original.

FuzzyLogic.jl: A Flexible Library for Efficient and Productive Fuzzy Inference

Author(s): Ferranti, Luca; Boutellier, Jani

Title: FuzzyLogic.jl: A Flexible Library for Efficient and Productive Fuzzy Inference

Year: 2023

Version: Accepted manuscript

Copyright ©2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Please cite the original version:

Ferranti, L. & Boutellier, J. (2023). FuzzyLogic.jl: A Flexible Library for Efficient and Productive Fuzzy Inference. In *2023 IEEE International Conference on Fuzzy Systems (FUZZ)*. IEEE International Fuzzy Systems conference proceedings. https://doi.org/10.1109/FUZZ52849.2023.10309777

FuzzyLogic.jl: a Flexible Library for Efficient and Productive Fuzzy Inference

1st Luca Ferranti
University of Vaasa
Vaasa, Finland
0000-0001-5588-0920

2nd Jani Boutellier
University of Vaasa
Vaasa, Finland

***Abstract*—This paper introduces FUZZYLOGIC.JL, a Julia library to perform fuzzy inference. The library is fully open-source and released under a permissive license. The core design principles of the library are: user-friendliness, flexibility, efficiency and interoperability. Particularly, our library is easy to use, allows to specify fuzzy systems in an expressive yet concise domain specific language, has several visualization tools, supports popular inference systems like Mamdani, Sugeno and Type-2 systems, can be easily expanded with custom user settings or algorithms and can perform fuzzy inference efficiently. It also allows reading fuzzy models from other formats such as Matlab .fis, FCL or FML. In this paper, we describe the library main features and benchmark it with a few examples, showing it achieves significant speedup compared to the Matlab fuzzy toolbox.**

***Index Terms*—Fuzzy inference, library, toolbox, Julia, Mamdani, Sugeno, Type-2, open-source software**

I. INTRODUCTION

Since their introduction by Zadeh in 1965 [1], fuzzy sets and fuzzy logic have steadily gained popularity as a technique to model uncertainty. They have been successfully applied to several engineering domains such as control theory [2], [3], image processing [4], [5] and data analysis [6]. During the past few decades, different inference systems have been proposed, including Mamdani [7], Sugeno [8], Tsukamoto [9] and more recently Type-2 [10] inference systems. With their growth in engineering and research communities, needs for a common framework to easily implement and compare different inference systems and algorithms have emerged.

In this paper, we introduce FUZZYLOGIC.JL a library for fuzzy inference written in Julia. The core design principles of the library are

- **Features:** The library supports both type-1 and type-2 Mamdani and Sugeno inference systems, and offers out-of-the-box several popular membership functions, defuzzification algorithms and T-norms.
- **Expressive:** The library comes with its own domain specific language (DSL) that allows to describe inference systems.
- **Compatible:** The library can read fuzzy systems from popular description languages, including the IEC 61131-7 Fuzzy Control Language (FCL) [11], IEEE 1855-2016 Fuzzy Markup Language (FML) [12] and MatLab fuzzy systems format. This allows seamless integration with previously developed models without code rewriting.
- **Productive:** The library offers visualization and debugging tools that allows the user to tune their model in an interactive manner.
- **Efficient:** The library achieves state-of-the-art performance. Leveraging the Julia type system and multiple dispatch, this can be achieved without giving up on the dynamic nature of the library. Furthermore, the library allows to synthesize stand-alone optimized code.
- **Expandable:** The library is developed in a modular way, allowing the user to easily expand with, e.g., custom membership functions, T-norms or defuzzifiers. As we show in Section IV, it is also possible to easily craft non-standard types of systems with custom evaluation.

Summarizing, the main contribution of this paper is the introduction of FuzzyLogic.jl, a fully open-source library for fuzzy inference, released under a permissive license available at https://github.com/lucaferranti/FuzzyLogic.jl. We show how our library covers several popular features of fuzzy systems and offers an expressive DSL to synthetically yet clearly describe fuzzy inference systems. We show how our library can be extended to easily implement custom inference systems and achieve state-of-the-art performance by using three popular real-world examples as test cases.

The paper is structured as follows: in Section II we briefly review the status of existing fuzzy inference tools and their main features. In Section III we analyze in more details the features of FuzzyLogic.jl and how it achieves its design principles. In Section IV we demonstrate how to use our library with three benchmark examples, comparing its features and performance to the Matlab Fuzzy Logic toolbox. Finally, conclusions are drawn in Section V.

II. RELATED WORK

In this section we briefly review existing libraries for fuzzy inference. A systematic literature review of open-source fuzzy systems software can be found in [13]. We do not intend to evaluate libraries over each other, the purpose of this section is simply to offer an unbiased review of features in other existing libraries.

The number of software frameworks for leveraging fuzzy logic in expert systems or for fuzzy inference steeply in-

creased in the 1980s [14], introducing works such as FLOPS [15]. Later frameworks, released in the 1990's and 2000's, include NEFCLASS [16], FuzzyShell [14], FuzzyCLIPS [17], FuzzyJess [18] and XFuzzy [19]. XFuzzy featured a graphical user interface for designing fuzzy systems based on the XFL language, and allowed synthesizing the developed systems into C or VHDL [19]. FuzzyCLIPS [17], Java-based FuzzyJess [18] and FuzzyShell [14], on the other hand, provided a system shell based interface mainly for describing fuzzy expert systems. Finally, NEFCLASS [16] concentrated on neuro-fuzzy learning and classification. In a sense closely related to FuzzyLogic.jl, FFLL [20] proposed a fast library for fuzzy logic written in C language.

One of the most advanced open-source fuzzy libraries is jFuzzyLogic [21], a Java library that supports both Mamdani and Sugeno type-1 inference systems and supports FCL parsing. It also allows the user to add custom functions, such as using custom membership functions or T-norms. While the library does not directly support FML, compatibility could be achieved combining jFuzzyLogic with JFML [22], a Java library to parse fuzzy systems specifications compliant with the IEEE 1855-2016 standard. Based on the documentation, jFuzzyLogic does not seem to support Type-2 inference systems. The Python library PyIT2FLS [23] supports both Type-1 and Type-2 Mamdani and Sugeno systems, however it does not support reading FCL or FML.

If not restricted to open-source, worth mentioning is the Matlab Fuzzy Logic toolbox, which allows to design Mamdani and Sugeno systems, both Type-1 and Type-2, has some visualization tools and supports custom functions. The toolbox, however, uses its own .fis format to store the implemented models and does not support reading/writing FCL nor FML.

III. Library design and features

In this section we describe the architecture and design choices of the library, highlighting how these allowed to achieve the design goals presented in Section I. The purpose of this section is to simply highlight the general design principles. For detailed tutorials and complete list of supported features, we refer to the library documentation [24].

As higher-level ambition, the library aims to offer a general framework for fuzzy inference. In addition to offering out-of-the-box a rich repertoire of features, the library should allow to expand existing inference systems with custom algorithms or even allow users to define non-standard inference systems with minimal ad-hoc manual tuning. This allows researchers and industry professionals to easily combine and compare different algorithms. Such flexibility, however, should be achieved without sacrificing speed. Last but not least, the library should make developing inference systems as enjoyable as possible for the users. That is, the code to develop an inference system should be easy to write and easy to read.

To achieve these goals, we chose to implement our library in Julia [25]. This choice was mainly driven by two factors:

- Julia has powerful metaprogramming features like syntactic macros (similar to Lisp). This allows us to define a custom Embedded Domain Specific Language (EDSL), to express inference systems in a Julia-like syntax.
- Julia type system and multiple dispatch functionalities were initially designed to achieve both flexibility and speed. In our library, each algorithm has its own type and the full inference pipeline is written in a generic and modular way. This makes the inference system flexible and allows the users to easily replace each component with their custom implementation. Furthermore, the type of the full inference system has its individual components (T-norm algorithm, defuzzification algorithm, etc.) as type parameters. Leveraging the Julia type inference system, the compiler is already able to determine what algorithm to use at compile time, practically removing any computational overhead of the generic implementation from runtime.

Listing 1: Tipper Mamdani system in FuzzyLogic.jl DSL.

A. Domain-Specific-Language

The library offers an embedded domain specific language to describe fuzzy inference systems in an expressive yet concise manner. As an example of the language, Listing 1 shows how to implement the tipper inference system.

The EDSL is implemented exploiting Julia metaprogramming capabilities such as syntactic macros. In the example above, the macro @mamfis takes as input the Julia code, in the form of a parsed syntax tree and produces new Julia code, which is then evaluated. Inside the macro, the syntax tree can be traversed and transformed to produce new code. This allows to create embedded domain specific languages that preserve the Julia syntax, but have completely different semantics (meaning). For example, the first line of the code, while looking syntactically as a function definition, should actually be read as "create a fuzzy system called tipper, which has input variables called food and service and an output variable called tip". This allows us to program the fuzzy system with a clear Julia-like syntax, while still being able to execute the code.

B. Library API

1) Type system: To best understand the structure and type system of our library, it is instructive to look at a concrete example. We start by looking at the type of the Mamdani inference system built in the previous section, displayed in Listing 2.

Listing 2: Type signature of a Mamdani inference system.

Each inference system has its own type, in the example above MamdaniFuzzySystem. The inference system has however also several type parameters (the types between braces). This means the algorithms used in each part of the inference pipeline are part of the inference system type itself. Each parameter in the type represents a setting in the inference

system that can be tuned. In the case of Mamdani system, these tunable settings are

- Conjuction operator
- Disjunction operator
- Implication algorithm
- Aggregation algorithm
- Defuzzification algorithm

The main advantage of having a type for each algorithm and parameterizing the whole system by its settings is in the gained efficiency. Since the algorithm is determined by the type alone, if the type is known at compile-type, the compiler can determine ahead of time, e.g., what function to use for defuzzification. This way, we keep a generic framework that supports multiple algorithms, but the choice of the algorithm is already done at compile time, removing the need to check what algorithm to use at run time.

Each setting has its own abstract type, for example CentroidDefuzzifier and all possible defuzzifier algorithms are subtypes of AbstractDefuzzifier. This allows writing generic code and makes the API of the library easy to expand with custom functions.

As a small example to show how the framework can be expanded, let us suppose the user would like to add a custom membership function MySingleton with parameter c defined as $\mu_c(x) = 1$ if $x = c$ and $\mu_c(x) = 0$ otherwise. Such a membership function can be added to our framework with the following lines of code

Listing 3: Defining a custom membership function.

In the first part, we are defining a type MySingleton for our membership function which has a parameter c. We are defining this to be a subtype of AbstractMembershipFunction defined in FuzzyLogic.jl to tell the library this should behave as a membership function. In the second part, we are defining how this membership function should be evaluated. This line of code will make objects of type MySingleton callable as functions.

These few lines of code are everything we need to define a new membership function, which can now be used seamlessly with all the built-in functionalities in the library. In a similar manner, we can also define custom T-norms, aggregation methods and defuzzification methods. In the next section, we will also show how we can reuse the library DSL and existing type system to design a non-standard inference system.

2) Variables: Variables in our library are stored in a hash-table, where each key is a symbol corresponding to the variable name and the value is an object of type Variable. This has two parameters, a domain and a hash-table storing the membership functions corresponding to the variable.

3) Rules: The atomic component of a fuzzy rule is a fuzzy relation, for example "service is good". A fuzzy rule is then composed by an antecedent and a consequent. The antecedent is a fuzzy proposition, while the consequent is a vector of fuzzy relations. Fuzzy propositions in our library are stored lazily as a tree, where each leaf is a fuzzy relation. For example, the antecedent of the first rule of tipper is constructed internally as follows.

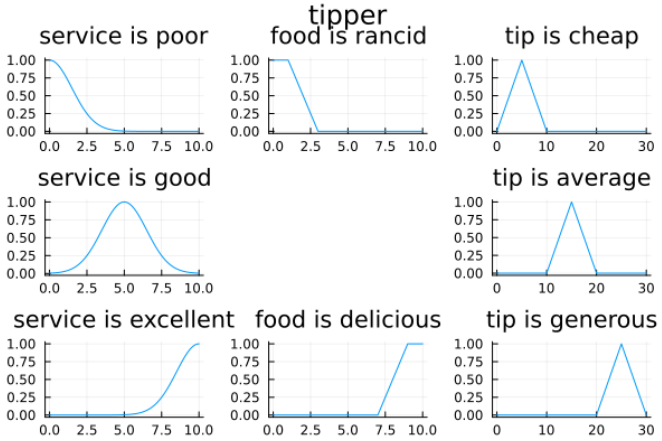

Fig. 1. Graphical visualization of the tipper inference system. Automatically created by the library.

Listing 4: Internal tree representation of a fuzzy rule.

As it can be noticed, a fuzzy relation is simply a pair of symbols, representing a variable and a membership function. Fuzzy relations can be evaluated as a function, giving as input the values of the variables and the lookup tables stored in the fuzzy system to determine what membership functions to use. Similarly, fuzzy propositions can be evaluated by recursively traversing the tree. The functions to use to evaluate the connectives can be read directly from the type of the fuzzy system, and hence are known at compile time.

C. Other features

Plotting: In addition to the features described above. The library also offers plotting functionalities to help visualization and debugging. For example, Figure 1 shows the graphical representation of the tipper inference system created in Listing 1. This plot is automatically generated by simply calling plot(fis). The plotting functionalities are implemented as generic code on the abstract types, meaning that the visualization would work out of the box, e.g., also for custom membership functions implemented by the user.

Interoperability: A core design principle of the library is to facilitate interoperability and minimize manual rewriting of legacy code. For this reason, the library comes with a built-in parser for IEC 61131-7 Fuzzy Control Language, supporting all the required features and several of the optional features. Furthermore, the library is also able to read fuzzy models generated with Matlab Fuzzy Logic Toolbox or models in the Fuzzy Markup Language.

Code generation: While the type system is designed to achieve both general flexibility and efficiency, once a fuzzy model has been designed and tuned, it may be desirable to generate stand-alone code. Currently, the library is able

to compile fuzzy models to an optimized stand-alone Julia function that can be used fully independently from the library.

IV. Examples and benchmarks

In this section we benchmark our library on three testcases, comparing execution time to the Matlab library. The codes to reproduce the experiments can be found at https://github.com/lucaferranti/fuzzieee2023. The first example is the tipper system presented in the previous section.

The second example is a controller for a wall-following robot presented in [26]. This is a Mamdani inference system with 4 input variables, each with 4, 2, 4 and 2 membership functions respectively; 2 output variables with 9 membership functions each, and 41 rules.

The third example is the fuzzy denoising algorithm presented in [4]. This fuzzy inference system has 8 input variables, all with the same membership functions, one output variable and 26 rules. This system is however slightly unconventional, namely it follows the traditional inference system pipeline only until the rule antecedent evaluation. Let

$$\begin{aligned}\lambda_1 &= \max\{r_i : i = 1 \dots 13\}\\ \lambda_2 &= \max\{r_i : i = 14 \dots 26\}\\ \lambda_0 &= \max(0, 1 - \lambda_1 - \lambda_2)\\ y &= (L-1)\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_0},\end{aligned} \tag{1}$$

where r_i is the output of the antecedent of the ith rule, y is the final output and L is the number of gray levels in the original image. This fuzzy system is fairly unconventional in the sense that it uses the result of each antecedent as crisp output of the rule, partitions the rules into two groups that are aggregated separately, and computes the final result from the two aggregated outputs. The implementation of this non-standard fuzzy system is shown in Listing 5, where the first part defines the inference system. While this is not a Mamdani system, we can reuse the `@mamfis` macro, since the Mamdani system holds all the pieces we need. This example also highlights a core feature of FuzzyLogic.jl, as it can specify variables as a vector $x[1:8]$ and it supports syntactic for-loops, which can be used to avoid manual typing when we have several variables constructed in a regular manner. We could also have used a similar for-loop syntax for the rules, by computing each variable index as a function of the rule index, but this would probably have lead to less readable rules.

The function detector defined below is the custom evaluation pipeline described in Equation 1. As mentioned in the previous section, each logical proposition is stored as a tree and can be called as a function. In the first two lines we evaluate the antecedent of each rule at the given inputs. The `fis` is passed as input to the antecedent too, as it contains in its type signature the information on how to evaluate each logical connective. After having evaluated the antecedents, we proceed with the custom inference pipeline. As this example shows, leveraging the library's flexible DSL and modular design, we could implement a non-standard inference system in only a few lines of code, hijacking the traditional inference pipeline to evaluate only part of it and build on top of it our custom inference procedure.

Listing 5: Inference system for image denoising.

TABLE I
Benchmarks on selected examples

Test case	FuzzyLogic.jl	Matlab fuzzy toolbox
Tipper	6 µs	60 µs
Robot controller	92 µs	300 µs
Image denoising	9 µs	120 µs

To conclude the experiments, we implemented the presented examples in our library and report the timings in Table I. For the first and second example, the implementation was presented in Listings 1 and 5. The second example was directly read by an available description in FCL format. The experiments were run on a laptop with 12th Gen Intel(R) Core(TM) i7-1255U processor and 64GB of RAM. As can be seen, our library is significantly faster than Matlab. One reason for this is that Matlab stores the parameters of the algorithm as strings and determines what algorithm to use at runtime, while our library can determine ahead of time, e.g., what defuzzifier implementation to pick, removing any need for comparisons at runtime.

V. Conclusions

In this paper we presented FuzzyLogic.jl, a Julia library for fuzzy inference. We described the main functionalities of the library and how it allows to design, visualize and debug different inference systems (Mamdani and Sugeno, both type 1 and type 2), and use them to perform efficient inference in different application domains. The library being open source and MIT licensed, it is well suited for adoption both in industry and academia. Furthermore, the library is extensible and allows users to add new algorithms or inference systems on top of the basic domain specific language and infrastructure.

References

[1] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[2] K.-S. Tang, K. F. Man, G. Chen, and S. Kwong, "An optimal fuzzy PID controller," *IEEE transactions on industrial electronics*, vol. 48, no. 4, pp. 757–765, 2001.

[3] J. Carvajal, G. Chen, and H. Ogmen, "Fuzzy PID controller: Design, performance evaluation, and stability analysis," *Information sciences*, vol. 123, no. 3-4, pp. 249–270, 2000.

[4] F. Russo and G. Ramponi, "A fuzzy filter for images corrupted by impulse noise," *IEEE Signal Processing Letters*, vol. 3, no. 6, pp. 168–170, 1996.

[5] S. K. Pal, "Fuzzy sets in image processing and recognition," in *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*. IEEE, 1992, pp. 119–126.

[6] P. Melin and O. Castillo, "A review on type-2 fuzzy logic applications in clustering, classification and pattern recognition," *Applied soft computing*, vol. 21, pp. 568–577, 2014.

[7] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International journal of man-machine studies*, vol. 7, no. 1, pp. 1–13, 1975.

[8] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE transactions on systems, man, and cybernetics*, no. 1, pp. 116–132, 1985.

[9] Y. Tsukamoto, "An approach to fuzzy reasoning method," *Advances in fuzzy set theory and applications*, 1979.

[10] N. N. Karnik, J. M. Mendel, and Q. Liang, "Type-2 fuzzy logic systems," *IEEE transactions on Fuzzy Systems*, vol. 7, no. 6, pp. 643–658, 1999.

[11] "IEC 61311-7 – Fuzzy Control Programming," International Electrotechnical Commission, Standard, 2000.

[12] G. Acampora and V. Loia, "Fuzzy control interoperability and scalability for adaptive domotic framework," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 2, pp. 97–111, 2005.

[13] J. Alcalá-Fdez and J. M. Alonso, "A survey of fuzzy systems software: Taxonomy, current research trends, and prospects," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 1, pp. 40–56, 2016.

[14] J. Pan, G. N. DeSouza, and A. C. Kak, "FuzzyShell: a large-scale expert system shell using fuzzy logic for uncertainty reasoning," *IEEE Transactions on fuzzy systems*, vol. 6, no. 4, pp. 563–581, 1998.

[15] J. Buckley, W. Siler, and D. Tucker, "A fuzzy expert system," *Fuzzy sets and systems*, vol. 20, no. 1, pp. 1–16, 1986.

[16] D. Nauck and R. Kruse, "NEFCLASS - a neuro-fuzzy approach for the classification of data," in *Proceedings of the 1995 ACM symposium on applied computing*, 1995, pp. 461–465.

[17] B. Orchard, "FuzzyCLIPS version 6.10 d user's guide," *National Research Council of Canada*, 2004.

[18] R. Orchard, "Fuzzy Reasoning in JESS: The Fuzzyj Toolkit and Fuzzyjess," in *ICEIS (1)*. Citeseer, 2001, pp. 533–542.

[19] D. López, C. Jiménez, I. Baturone, A. Barriga, and S. Sánchez-Solano, "Xfuzzy: A design environment for fuzzy systems," in *1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36228)*, vol. 2. IEEE, 1998, pp. 1060–1065.

[20] M. Zarozinski, "An open-source fuzzy logic library," in *AI Game Programming Wisdom 2 (ed. Steve Rabin))*. Charles River Media, Inc., 2006.

[21] P. Cingolani and J. Alcalá-Fdez, "jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming," *International Journal of Computational Intelligence Systems*, vol. 6, no. sup1, pp. 61–75, 2013.

[22] J. M. Soto-Hidalgo, J. M. Alonso, G. Acampora, and J. Alcala-Fdez, "JFML: A java library to design fuzzy logic systems according to the ieee std 1855-2016," *IEEE Access*, vol. 6, pp. 54 952–54 964, 2018.

[23] A. A. Haghrah and S. Ghaemi, "PyIT2FLS: A new python toolkit for interval type 2 fuzzy logic systems," 2019.

[24] L. Ferranti. (2023) FuzzyLogic.jl documentation. [Online]. Available: https://lucaferranti.com/FuzzyLogic.jl/stable

[25] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[26] M. Mucientes, R. Alcalá, J. Alcalá-Fdez, and J. Casillas, "Learning weighted linguistic rules to control an autonomous robot," *International Journal of Intelligent Systems*, vol. 24, no. 3, pp. 226–251, 2009.