Summary of 'Animated fuzzy logic' by Gary Meehan and Mike Joy

Luis W. Chavez Quiroz
lwchave2@illinois.edu
UIN: 669863000
Presentation: youtube.com/
Code: https://github.com/luiswally/animated-fuzzy-logic

Overview

Meehan and Joy explored fuzzy logic through a Haskell implementation (Meehan & Joy, 1998). The aforementioned research paper will be summarized for an audience unfamiliar with the concept of fuzzy logic but with a basic understanding of Haskell. The summary presented herein will cover fuzzy logic and its Haskell implementation in the following sections: Basics, explore high-level concepts of fuzzy logic, Applications, present both test cases and real world use cases, Haskell Implementation, highlight the intricacies of implementing fuzzy logic via Haskell, and Conclusion, recap how fuzzy logic is naturally implemented in a functional programming language.

Before proceeding, we wish to highlight the motivation behind studying fuzzy logic. I recently needed to purchase a rice cooker and the marketing phrase 'Fuzzy logic technology' caught my attention. The class project presented herein gave me the opportunity to explore this topic. The goal of this project was to gain an understanding of what fuzzy logic is and explore a Haskell implementation of it. Having a basic to intermediate level understanding of Haskell at the end of this course (CS 421) I am glad to report that the knowledge gained from this course allowed me to implement the fuzzy logic demos presented in the research paper as well as create my own demos which will be presented later in this paper (see *demo.hs* and *gains.hs* in github repo /project/).The biggest difficulty encountered was recreating the demos presented in the paper as the demos assume access to the programs presented in the paper, which were not accessible to the public. Dependencies such as functions and special subsets were rebuilt to get demos working, and creating our own demo was particularly challenging.

Fuzzy logic high-level overview

Fundamentals

Fuzzy logic is not too dissimilar from boolean logic, in fact boolean logic can be considered to be a special case of fuzzy logic (Karimi, 2009, 302-309). Boolean logic is binary, that is to say a predicate is either *true* or *false*, 1 or 0, fuzzy logic expands the domain of predicates to a continuous range, by convention the unit interval [0, 1]. As truth values approach the bounds of the unit interval, 0 or 1, the values become *crisper*, and as they approach the center of the unit interval, 0.5, the values become *fuzzier*.

Standard logical connectives ($\wedge$, $\vee$, $\neg$) behave similarly in fuzzy logic as they do in boolean logic, any implementation of such must have the following properties: (1) $\wedge$ and $\vee$ definitions must be t-norm and t-conorm, (2) connectives must be continuous, and (3) Connectives must adhere to De Morgan's Laws. For $\wedge$ and $\vee$ to be t-norm, they should be associative, commutative, and monotonic. Furthermore, the t-conorm must be defined to be the complement of the t-norm under the negation operator. A popular implementation of the standard logical connectives satisfying De Morgan's Law and ensuring continuity is Zadeh's implementation (Zadeh, 1965). As presented by Meehan and Joy (Meehan & Joy, 1998):

$\perp(a, b) = 1 - \top(1 - a, 1 - b)$ $\qquad$ $\top()$ - t-norm function $\qquad$ $\perp()$ - t-conorm function

$x \wedge y = min(x, y)$ $\qquad$ $x \vee y = max(x, y)$ $\qquad$ $\neg x = 1 - x$

Subsets

A fuzzy subset, F, is a set of pairs that contain each element from a superset, X, and the corresponding degree to which the element belongs to the fuzzy subset. This may be written as $F = \{< x, \mu_F(x) > | x \in X\}$, where $\mu_F(x)$, characteristic function, ranges from [0, 1]. $\mu_F(x) = 1$ if x is definitely in F, and as $\mu_F(x)$ approaches 0, x's degree of belonging to F decreases accordingly. Given that X is the domain of F, the characteristic function $\mu_F(x)$ and fuzzy subset F are identical. See *Haskell Implementation* for implementation implications.

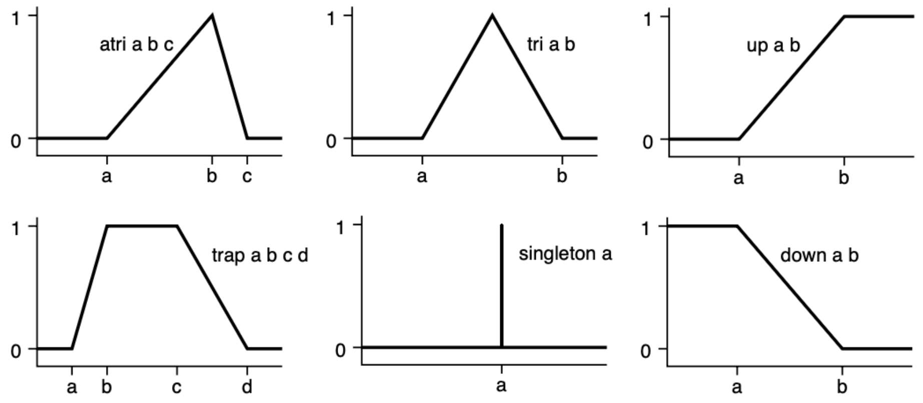

Figure 1. Standard fuzzy subset distributions

There are several standard fuzzy subset distributions, as shown by Meehan and Joy (Meehan & Joy, 1998) in Figure 1. While the domain of a fuzzy subset F is the entire superset of interest, X, the support of F, $\sigma(F)$, is the range of values in X with a non-zero truth value, i.e.: $\sigma(F) = \left\{\mu_F(x) \neq 0 | x \in X\right\}$. As alluded to in *Fundamentals*, the fuzziness of a fuzzy subset F can be measured by the characteristic function's $\mu_F(x)$ propensity to cluster around 0.5.

Hedges, fuzzy numbers, and defuzzification

Fuzzy logic has the benefit of being able to directly model implicit uncertainties in a typical classification or categorization problem. For instance, say we wish to measure the lucrativeness of an employee's yearly raise. We may choose to define an upward fuzzy subset with 3% and 10% as the floor and ceiling respectively, anything past 10% is definitely lucrative and would be cut off using boolean logic. While intuitively, we can say a 4% raise isn't very lucrative, a 8% raise is pretty lucrative, and a 9% raise is very lucrative, standard boolean logic would rate all these raises as NOT lucrative. Fuzzy logic allows us to provide flexibility to the classification lucrative via dampening or intensifying the characteristic function (fuzzy subset) to skew elements in one direction, lucrative or not lucrative.

For simplicity, say a fuzzy value of 0.5 is sufficient to classify a raise as lucrative. For an individual named *Very*, their career ambitions may be higher than average and thus are more selective with their raise appraisals:

```
lucrative :: Fuzzy Percentage
lucrative = up 3 10
*Demo> very lucrative 4
2.040816326530612e-2
*Demo> very lucrative 8
0.5102040816326531
*Demo> very lucrative 9
0.7346938775510203
```

However, the career driven one-percenters like *Extremely*, will take nothing less than excellence:

```
*Demo> extremely lucrative 4
2.9154518950437313e-3
*Demo> extremely lucrative 8
0.3644314868804665
*Demo> extremely lucrative 9
0.629737609329446
```

As seen in the demo above, the two individuals assessed their raises differently, *Very* considered two of their raises (8% and 9%) as lucrative, while *Extremely* only considered one (9%) worthy. Hedges serve to capture these differing assessments or by the use of qualifiers in the base assessment, common hedges are *very*, *extremely*, *somewhat*, *slightly*:

$$\mu_{extremely\ F}(x) = \mu_F(x)^3 \qquad \mu_{very\ F}(x) = \mu_F(x)^2$$

$$\mu_{somewhat\ F}(x) = \mu_F(x)^{1/2} \qquad \mu_{slightly\ F}(x) = \mu_F(x)^{1/3}$$

Additionally hedges can be used to approximate numbers by turning them into fuzzy subsets with distributions varying depending on the qualifier used, shown in increasing fuzziness, i.e. softer restrictions: *near*, *around*, *roughly*.

Once a fuzzy subset is arrived at through whatever series of permutations are conducted, a fuzzy value by which to infer a decision is often helpful in the decision making process. Two popular approaches to achieve this are: (1) determining the centroid of the distribution or (2) reporting the one of the maxima of the fuzzy subset such as: minimum, median, or maximum.

Applications

Fuzzy logic is used in a wide variety of embedded systems, home appliances, artificial intelligence, and pretty much anywhere where decision making is coarser and ambiguities arise (Firouzi et al., 2020). Network traffic control is notoriously difficult given the dynamic nature of network traffic and ever increasing demands. Internet gateways have been shown to benefit from fuzzy logic controllers as results show a reduction in latency and increase in accuracy when compared to standard operating controls over cloud (Firouzi et al., 2020). Temperature and pressure control applications can also benefit from fuzzy logic given the inherently subjective nature of 'cold water' or 'hot room temperature (Meehan & Joy, 1998). Another common fuzzy logic application lies in everyday rice cookers (Toothman, 2008). By varying pressure, temperature, and moisture, fuzzy logic rice cookers can cook rice quickly, with varying degrees of texture (soft, hard, sticky, wet). To anybody familiar with cooking, it can be very easy to overshoot temperature or duration under temperature, fuzzy logic provides persistent adjustments to parameters such as these.

Database queries are also another application of fuzzy logic. An everyday task, list filtering can be abused via functional programming and fuzzy logic to provide a quick solution to a task that might otherwise involve setting up a framework of parameters and constraints. Haskell naturally lends itself to set operations over a fuzzy subset as the fuzzy subset is implemented as a function, thus querying a structured database requires nothing more than pattern matching the database structures with our characteristic equation, as appropriate (Meehan & Joy, 1998):

```haskell
ffilter :: Fuzzy a -> [a] -> [(a, Double)]
ffilter p xs = filter ((/=) 0 . snd) (map (\x -> (x, p x)) xs)
```

Similar to the profitable example by Meehan and Joy (Meehan & Joy, 1998), an analysis over my personal muscle gaining plan was used. The two fuzzy subsets used were to measure weekly weight gain and average daily protein intake over a week. The first used a skewed triangle distribution, the reasoning being that excessive weight gain is not muscle growth, and the latter used an upward distribution as for this exercise, excessive protein consumption is not a worry:

```haskell
proteinIntake :: Fuzzy Protein
proteinIntake = up 50 120
muscleGrowth :: Fuzzy Weight
muscleGrowth = atri 0.25 0.85 1.20
```

Using the above fuzzy subsets, last month's records were analyzed using the following hedges and fuzzy subsets:

```haskell
records :: [Record]
records = [("2023-07-12", 75.00, 1.15), ("2023-07-19", 40.65, 0.80),
          ("2023-07-25", 94.20, 1.10), ("2023-08-03", 105.92, 1.00)]

p1 record = muscleGrowth (weight record)
p2 record = muscleGrowth (weight record) && proteinIntake (protein record)
p3 record = somewhat muscleGrowth (weight record) && very proteinIntake (protein record)
```

demo:

```
*Gains> ffilter p1 records
[(("2023-07-12",75.0,1.15),0.143),(("2023-07-19",40.65,0.8),0.917),(("2023-07-25",94.2,1.1),0.286),(("2023-
08-03",105.92,1.0),0.571)]
*Gains> ffilter p2 records
[(("2023-07-12",75.0,1.15),0.143),(("2023-07-25",94.2,1.1),0.286),(("2023-08-03",105.92,1.0),0.571)]
*Gains> ffilter p3 records
[(("2023-07-12",75.0,1.15),0.128),(("2023-07-25",94.2,1.1),0.399),(("2023-08-03",105.92,1.0),0.638)]
```

It's clear that the first filter was only concerned with weight gain, as the best performing record was "2023-07-19", however, when considering protein intake to, the controlling record is "2023-08-03" for both of the other filters. It should be noted that the last filter was stricter when it came to protein consumption, but more lax with regards to weight gain requirements, explaining the increase in favorability of "2023-08-03" via the use of hedges on both fuzzy subsets *muscleGrowth* and *proteinIntake*.

Haskell Implementation

To implement fuzzy logic in Haskell Meehan and Joy (Meehan & Joy, 1998) create a fuzzy variable class *Logic* to house different instances of fuzzy variables (e.g. Double, Boolean, etc) and to shadow standard operators and overload standard logical connectives. The overload is then specified in each instance of the *Logic* class. Resulting in standard operators behaving uniquely when interacting with instances of the *Logic* class:

```haskell
-- Fuzzy variable class allows shadowing of standard operators for fuzzy variables ONLY
class Logic a where
   true, false :: a
   (&&), (||)  :: a -> a -> a
   not         :: a -> a

-- overloading standard operators to work on fuzzy instances of Logic class
and, or :: Logic a => [a] -> a
and      = foldr (&&) true
or       = foldr (||) false

any, all :: Logic b => (a -> b) -> [a] -> b
any p     = or . map p
all p     = and . map p
```

From *Subsets*, since a fuzzy subset and its characteristic function are identical, we can represent a fuzzy subset via its characteristic function. Since Haskell is a functional language, this facilitates the manipulation of the fuzzy subset (characteristic function) via a straightforward use of higher-order functions as every function in Haskell is a first-class citizen. In short, we can assign our characteristic function to a fuzzy subset, pass the fuzzy subset as a parameter to a function, and return the new fuzzy subset or function. All of this accomplished without the need to keep track of state in our program. For instance, in *Applications* we demoed the use of hedges and union functions (&& operator) via the use of higher order functions.

Fuzzy subset in haskell is just a function, thus any fuzzy subset operation we wish to perform is simply a functional in haskell (higher order function), because of this. We do not need to define or keep track of unique states to expand upon different applications and uses of fuzzy logic.

References

Firouzi, R., Rahmani, R., & Kanter, T. (2020). An autonomic IoT gateway for smart home using fuzzy logic reasoner. *Procedia Computer Science*, *177*, 102-111. 10.1016/j.procs.2020.10.017

Karimi, H. A. (2009). *Handbook of Research on Geoinformatics*. IGI Global. 10.4018/978-1-59140-995-3.ch038

Meehan, G., & Joy, M. (1998, September). Animated fuzzy logic. *Journal of Functional Programming*, *8*(5), 503-525. 10.1017/S0956796898003177

Toothman, J. (2008, April 4). How Rice Cookers Work. *HowStuffWorks*. https://home.howstuffworks.com/rice-cooker2.htm

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, *8.3*, 338-353. 10.1016/S0019-9958(65)90241-X