Lab Report on Fuzzy Systems: Implementing a Simple Fuzzy 'What Should I Wear?' System Using Haskell

Seth Kurtenbach

.....

Dr. Keller, Andrew Buck

0.1 Introduction

In this lab I chose to implement fuzzy logic using the purely functional programming language Haskell. I use a toy example of deciding what type of clothing to wear based on input of temperature, humidity, and the level of 'fanciness' of the occasion. My implementation is built on the foundation provided by Gary Meehan and Mike Joy in "Animated Fuzzy Logic," appearing in *The Journal of Functional Programming*, 1993. Meehan and Joy show that implementing fuzzy logic in Haskell is a relatively straightforward proposition.

As a purely functional language, every Haskell function is a function in the mathematical sense. Similarly, Haskell easily defines higher-order functions. Thus, it is natural to treat fuzzy subsets as functions from objects to real numbers in the interval [0, 1], i.e., as their membership functions. It is also natural to treat fuzzy subset operators as higher-order functions: Fuzzy subset operators operate over fuzzy subsets, which we represent as functions. Thus, fuzzy subset operators operate over functions: They are higher-order functions.

This lab report represents a feasibility study of applying Meehan and Joy's foundation to a practical problem. As such, the program should be regarded as a prototype, to be expanded upon in the future. I turn now to the technical description.

0.2 Technical Description

Like any programming language, Haskell comes pre-loaded with a standard library of basic logical operations. Meehan and Joy begin building their foundation by masking these pre-loaded operations so that they can reuse their names in defining the analogous fuzzy logical operators. I followed them in masking the classical logic operators, and I used their redefined fuzzy versions of the operators. I extended their fuzzy logic with additional operators.

They define their operators as follows:

```
instance Logic Double where
true = 1
false = 0
(&&) = min
(||) = max
(not x) = 1 - x
```

```
(x ==> y) = x * y
```

They chose to define conjunction (&&) as min, disjunction (||) as max, complement in the standard way, and implication as correlation product. Because fuzzy logic maps into the unit interval rather than the set {0,1}, the conjunction and disjunction definions are not equivalent to their usual counterparts: product and sum minus product respectively, which are equivalent in classical logic. Meehan and Joy's fuzzy logic represents implication in a strange way, as well, in that it is merely correlation product. This differs from Zadeh's formulation, which follows the classical definition based on negation and disjunction.

I extended their foundation by introducing a framework for conjunction/intersection as product, disjunction/union as sum minus product, and all of the infinite Yager operators. Recall that a Yager operator involves exponentiation, for example:

$$(A \cup_w B)(x) = min\{1, (A(x)^w + B(x)^w)^{1/w}\}, \text{ where w} > 0.$$

I incorporated the above Yager union into Haskell with the following function:

```
yagerOr x p q = if x > 0 then (minimum [1, (((p^x) + (q^x)))^(1/x)])
else undefined.
```

Likewise I incorporated the other Yager operations in a similar manner. These definitions are not quite accurate, so I leave examples of their application to future versions of the prototype.

Fuzzy logic defines linguistic variables on domains of objects, and these variables are assigned predicate functions on the domain as values. For example, in my decision system I model the linguistic variable "temperature", defined on the domain [0, 100]. It takes as values the following predicates: freezing, cold, chilly, warm, hot, sweltering. Each predicate is defined as a fuzzy subset of the domain. In Haskell, defining these predicates is straight-forward:

```
warm :: Fuzzy Temperature
warm = trap_func 57 75 80 90,
```

which reads, "the function 'warm' is a Fuzzy Temperature subset, defined as a trapezoid function with joints at 57, 75, 80, and 90." The numbers here represent degrees Farenheit.

I chose to hard code the domains and fuzzy subsets for this prototype, but this is not necessary in principle. I follow Meehan and Joy in defining a domain as a type synonym for a list of objects of a specified type. Haskell is a strongly typed, statically typed language, and defining domains in this way exploits this feature. For example, I define the domain on which 'fanciness' is defined as:

```
dressedUp :: Domain Fanciness
dressedUp = [0,1..100],
```

which reads, "dressedUp is defined on a domain of objects of the Fanciness type, represented here as [0, 100]."

In addition to the basic predicates, Meehan and Joy provide definitions of hedges, so that we can modify the predicates with terms like "very", "somewhat", "extremely", and "barely". The effect of hedges is to change the predicate's slope, thereby raising or lowering an object's membership value in the predicate. I give examples of this later in the report. The technical definition of a hedge in their Haskell code is:

```
hedge :: Double -> Fuzzy a -> Fuzzy a
hedge pwr f (x) = if fx == 0 then 0 else fx ** pwr
        where fx = f (x),
```

which reads, "a hedge takes a number and a fuzzy subset value and raises the fuzzy subset value to the power of the number."

I constructed rules based on conversations with friends and my own judgment about what clothing would be appropriate in different scenarios. Meehan and Joy provide a function that fires all of the applicable rules in a domain and then defuzzifies via the centroid function, defined as follows:

```
centroid :: Domain Double -> Fuzzy Double -> Double
centroid dom f = (sum (zipWith (*) dom fdom))/(sum fdom)
        where fdom = map f dom,
```

which reads, "the centroid function takes a domain of double precision numbers and a subset defined on that domain, and returns a double precision number. It maps the subset onto the domain, which Haskell treats as function application. It then multiplies the resulting value by the original domain value, sums these values, and divides by the summation of the subset values."

Meehan and Joy designed the following framework for firing all rules and taking their centroid, which I use in my system, for example:

```
how_dressy :: Occasion -> Heaviness
how_dressy occ = centroid dressedUp (

        rulebase (+) [
                sickDay occ ==> jammies,
                relaxedOccasion occ ==> relaxedClothes,
                walMart occ ==> professor,
                inPublic occ ==> businessCasual,
                special occ ==> businessPro,
                fancy occ ==> extremely businessPro]) .
```

This function takes an occasion measurement (degree of fanciness), fires all of the rules on it, then takes the centroid of the result.

This concludes my technical description. Next I describe the decision system's design.

0.3 Design Description

The primary work that went into the design of the system concerned the predicate definitions. I chose to define all of my domains on the interval [0, 100] for simplicity, and predicates are functions defined on this domain.

The temperature predicates are defined as follows:

```
type Temperature = Double

sweltering :: Fuzzy Temperature
sweltering = up 85 100

hot :: Fuzzy Temperature
hot = up 60 90

warm :: Fuzzy Temperature
warm = trap_func 57 75 80 90

chilly :: Fuzzy Temperature
chilly = trap_func 46 55 60 70

cold :: Fuzzy Temperature
cold = down 40 50
```

```
freezing :: Fuzzy Temperature
freezing = down 32 42
```

Humidity predicates are defined as follows:

```
type Humidity = Double

arid :: Fuzzy Humidity
arid = down 15 30

dry :: Fuzzy Humidity
dry = down 40 50

pleasant :: Fuzzy Humidity
pleasant = trap_func 30 50 60 70

damp :: Fuzzy Humidity
damp = tri_func 65 75 85

muggy :: Fuzzy Humidity
muggy = up 80 85
```

Heaviness predicates are defined as follows:

```
type Heaviness = Double

nothing :: Fuzzy Heaviness
nothing = down 0 7

light :: Fuzzy Heaviness
light = trap_func 3 8 12 15

regular :: Fuzzy Heaviness
regular = trap_func 13 21 28 35

layers :: Fuzzy Heaviness
layers = trap_func 30 40 50 65

bundled :: Fuzzy Heaviness
bundled = trap_func 60 70 80 85

parka:: Fuzzy Heaviness
parka = up 80 85
```

Fanciness predicates are defined as follows:

```
type Fanciness = Double
```

```
professor :: Fuzzy Fanciness
professor = down 10 20

jammies :: Fuzzy Fanciness
jammies = trap_func 15 25 35 45

relaxedClothes :: Fuzzy Fanciness
relaxedClothes = trap_func 35 50 55 70

businessCasual :: Fuzzy Fanciness
businessCasual = trap_func 55 70 80 90

businessPro :: Fuzzy Fanciness
businessPro = up 80 90
```

Finally, Occasion predicates are defined as follows:

```
type Occasion = Double

sickDay :: Fuzzy Occasion
sickDay = trap_func 10 15 20 25

relaxedOccasion :: Fuzzy Occasion
relaxedOccasion = trap_func 20 30 40 50

walMart :: Fuzzy Occasion
walMart = down 10 25

inPublic :: Fuzzy Occasion
inPublic = trap_func 40 50 60 70

special :: Fuzzy Occasion
special = trap_func 65 75 85 90

fancy :: Fuzzy Occasion
fancy = up 85 90
```

Having chosen the predicate definitions, the next step was to devise fuzzy logic rules of inference. Fuzzy logic rules of inference have the form,

1. If Linguistic Variable U is Predicate A, then Linguistic Variable B is Predicate V.
2. Linguistic Variable U is Predicate A'.
3. Therefore, Linguistic Variable B is Predicate V',

where A' and V' denote degrees of membership to A and V respectively.

I chose to implement the following rules governing clothing decision making on the 'Heaviness' dimension:

1. If Temp is Hot and Humidity is Muggy, Then Heaviness is Barely Nothing.
2. If Temp is Hot, Then Heaviness is Light.
3. If Temp is Warm and Humidity is Pleasant, Then Heaviness is Regular.
4. If Temp is Warm and Humidity is Damp, Then Heaviness is Barely Regular.
5. If Temp is Warm and Humidity is Very Damp, Then Heaviness is Slightly Layers.
6. If Temp is Slightly Cold and Humidity is Damp, Then Heaviness is Extremely Layers.
7. If Temp is Cold, Then Heaviness is Bundled.
8. If Temp is Freezing and Humidity is Dry, Then Heaviness is Parka.
9. If Temp is Freezing and Humidity is Arid, Then Heaviness is Extremely Parka.

Note that the "Slightly" hedge raises the membership function value to the power of $\frac{1}{3}$, thereby shifting the shape to the left. Thus, "Slightly Cold" actually indicates more cold. Future versions of the prototype will avoid this counterintuitive formulation, but for now I'll point it out and move on.

The motivation behind the 'Heaviness' dimension rules is that higher humidity makes one lose heat more slowly, so one feels relatively hotter, and will want to wear relatively less clothes than at the same temperature with lower humidity. For example, if it is 85 degrees and muggy, then one wants to be wearing less than at 85 and pleasant. The wisdom of this decision is tested by the numbers that we shall see in the results.

I implemented the following rules on the 'Fanciness' dimension:

1. If Occasion is WalMart, Then Fanciness is Professor.
2. If Occasion is SickDay, Then Fanciness is Jammies.
3. If Occasion is RelaxedOccasion, Then Fanciness is RelaxedClothes.

4. If Occasion is InPublic, Then Fanciness is BusinessCasual.
5. If Occasion is Special, Then Fanciness is BusinessPro.
6. If Occasion is Fancy, Then Fanciness is Extremely BusinessPro.

These rules are ordered from least fancy occasion to fanciest occasion, and similarly the fanciness of the appropriate clothing style is similarly ordered. By the "Professor" predicate I refer to the habits of mathematics and physics professors. The pertinent rule captures the similarity of their fashion to that of regular Wal-Mart customers.

Finally, I define my hedges in the following way, based on the above technical hedge definition. Only "barely" is my own; the rest come from Meehan and Joy.

```
very, extremely, somewhat, slightly, barely :: Fuzzy a -> Fuzzy a
very = hedge 2
extremely = hedge 3
somewhat = hedge 0.5
slightly = hedge (1/3)
barely = hedge 0.2
```

This concludes my discussion of the system's design. Next I discuss results of the system.

0.4 Results

My goal for the system is to deliver reasonable numbers to the inputs, based on the interpretation that all domains are [0, 100], representing a degree of some sort, whether it is actual temperature, or degree of fanciness, where 100 is the fanciest one can get. The result of testing a domain member's fuzzy set membership is always a number in the unit interval. I intentionally use different scales in the hope that it helps avoid confusion. Let us turn to some results.

1. How cold is 32 degrees Farenheit?

```
*Fuzzy> cold 32
    1.0
```

2. How hot is 32 degrees Farenheit?

```
*Fuzzy> hot 32
```

```
    0.0
```

3. How hot is 72 degrees Farenheit?

```
*Fuzzy> hot 72

    0.4
```

4. How warm is 72 degrees Farenheit?

```
*Fuzzy> warm 72

    0.8
```

5. How muggy is 85% humidity?

```
*Fuzzy> muggy 85

    1.0
```

6. How dry are the elements in the humidity range [30,45]?

```
*Fuzzy> map dry [30..45]

    [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.9,0.8,0.7,0.6,0.5]
```

7. If the temperature is 40, and the humidity is 65, how heavy should I dress?

```
*Fuzzy> how_many_layers 40 65

    73.6
```

8. If the temperature is 90, and the humidity is 50, how heavy should I dress?

```
*Fuzzy> how_many_layers 90 50

    9.4
```

9. If the occasion is 50 degrees fancy, how fancy should I dress?

```
*Fuzzy> how_dressy 50

    73.5
```

10. On two dimensions: If the occasion is 80 degrees fancy, how fancy should I dress? If the temperature is 25 and the humidity is 20, how heavy should I dress? [Call this the Alaskan Ball]

```
*Fuzzy> how_dressy 80
```

```
        92.4

    *Fuzzy> how_many_layers 25 20

        84.8
```

11. Same as #6, but evaluate using product conjunction rather than min conjunction.

```
*Fuzzy> how_many_layers' 90 50

        9.4
```

12. Same as #5, using product conjunction.

```
*Fuzzy> how_many_layers' 40 65

        73.5
```

13. Same as #10, using Zadeh implication instead of correlation product implication.

```
*Fuzzy> how_many_layers' 40 65

        50.4
```

14. Same as #6, using Zadeh implication.

```
*Fuzzy> how_many_layers 90 50

        49.6
```

15. If it is 100 degrees Farenheit, with 100 percent humidity, how heavy should I dress, using Zadeh implication?

```
*Fuzzy> how_many_layers 100 100

        49.1
```

16. Same as #13, using product conjunction instead of min conjunction.

```
*Fuzzy> how_many_layers' 100 100

        49.1
```

17. Same as #13, using correlation product implication instead of Zadeh implication.

```
*Fuzzy> how_many_layers 100 100

        6.5
```

18. Same as #15, but with correlation min implication instead of correlation product implication.

```
*Fuzzy> how_many_layers 100 100

        6.5
```

18. If it is 0 degrees and 0 humidity, how heavy should I dress? Use correlation product implication and min conjunction.

```
*Fuzzy> how_many_layers 0 0

        85.6
```

20. Same as #17, using Zadeh implication.

```
*Fuzzy> how_many_layers 0 0

        52.8
```

21. If the occasion is 0 degrees fancy, how fancy should I dress, using (a) Zadeh implication? Using (b) correlation product implication?

```
(a) *Fuzzy> how_dressy 0

        48.7

   (b) *Fuzzy> how_dressy 0

        7.5
```

22. If the occasion is 100 degrees fancy, how fancy should I dress, using Zadeh implication?

```
(a) *Fuzzy> how_dressy 100

        51.1

   (b) *Fuzzy> how_dressy 100

        93.8
```

23. What values result from the warmness of 60 degrees and the aridity of 20% humid, using (a) product conjunction, and (b) min conjunction?

```
*Fuzzy> warm 60 &&& arid 20

        0.1

   *Fuzzy> warm 60 && arid 20
```

```
0.2
```

24. Use (a) correlation product implication and (b) correlation min implication to judge the appropriate heaviness if it is 75 temperature and 50 humidity.

(a)
```
*Fuzzy> how_many_layers 75 50
        21.0
```

(b)
```
*Fuzzy> how_many_layers 75 50
        20.4
```

0.5 Analysis of Results

The most striking and surprising result is that Zadeh implication is utterly uninformative. I expected Zadeh implication to perform the best, as it best represents logical implication in the classical sense. However, for all input values, Zadeh implication delivered results clustered very closely around 50. This puzzles me, and it seems like it is probably an error. If it is not an error, then it must be due to a combination of the mathematics of Zadeh implication and the number of rules being fired on each input. In either case, it warrants further examination.

On a similar note, correlation min implication and correlation product implication are almost indistinguishable on this data set, although they are both very reasonable. I suspect that their similarity is due to the large number of rules being fired, which might have a washing out effect. I am surprised that Result #24 returns a correlation product that is higher than the correlation min result. This result also warrants further investigation.

With respect to the general approach to engineering fuzzy systems, I think the Haskell foundation developed by Meehan and Joy is great. What complicated this lab for me was that rather than simply importing the available program corresponding to their paper, I recreated all of their code first-hand as I read through their paper, partially to familiarize myself with the system, as well as fuzzy logic in general. However, I could have imported the program and merely defined my desired domains, linguistic variables, and predicates, and this would have easily generated a functioning fuzzy decision system. Future work to be done on this approach is to generalize it, so that a user can define domains, linguistic variables, and predicates "on the fly", rather than creating and editing source code. This should be a straight-forward proposition, but I did not investigate that option in this lab.

I attempted to extend the framework by implementing the class of Yager operators, but was unsuccessful. The framework would benefit from such an extension in the future. Such an extension may help reconcile the difficulties with Zadeh implication.

Finally, in order to make the output more readable, I would like to represent the predicates that output values fall under. This can be easily looked up by consulting the predicate definitions, so it should be simple to append to the output.