# Fuzzy Expert Systems Jocelyn Ireson-Paine [www.j-paine.org](http://www.j-paine.org/)

This report introduces the basic concepts of fuzzy logic and fuzzy-logic rules, and works through two examples.

The first example, which I adapted from an academic paper on fuzzy expert systems, forecasts men’s shoe sizes from their height. It shows how an expert system interprets its rules, a process that I’ve been able to summarise in rather less than half a page. As everyone knows about shoes, you might want to use it to help your audience understand fuzzy expert systems.

The second example is adapted from the same paper, and concerns pricing of goods. It recaps concepts from the first example, and introduces new ones. These include the ability of fuzzy expert systems to reconcile apparently incompatible demands, such as a price having to be high to maximise takings, but low to maximise sales. The example also shows how fuzzy expert systems accommodate vague words such as “about”, “very”, “somewhat”, and the logical operations “and”, “or”, and “not”. The shoe-size system doesn’t use these, but we’ll probably need them in a realistic KPIs knowledge base.

## Fuzzy Concepts

Now let’s learn about fuzzy sets. A good starting point is to note that many concepts, and perhaps even most concepts, don’t have sharp boundaries. In maths, we distinguish between odd numbers and even numbers. There’s a sharp boundary between them: no integer can be both odd and even.

Nor can an integer be partly odd, or partly even. The number 1 is 100% odd, as it were; the number 2 is 100% even. There are no integers that are 33% odd or 50% even.

But elsewhere, things are less clear-cut. I am 6 foot 4 inches (1.93 metres) and in Britain, I count as tall. Someone who is 6 foot would just about count as tall; someone who is 5 foot 10, perhaps not. You can’t find a height ℎ such that ℎ is not tall, but ℎ+1cm is tall. Instead, there’s a gradual transition from not-tall to tall.

Similarly for concepts such as “wet” (of weather), “profitable” (of a business), “cheap” (of a good), “safe” (of a process or machine), “big” (of a shoe), “friendly” (of a dog or person), and thousands of others. You’ll be able to think of plenty from your own business, and it might be a good idea to have a few ready to show your audience.

## “Crisp”

Fuzzy-logic people often use the adjective “crisp”. The word is good to know, as a quick way to distinguish sharp things from fuzzy things. For example, “odd” and “even” are crisp concepts. “Tall” is a fuzzy concept.

## Crisp Sets

There is a very well established part of maths called set theory. A set is just a collection of things, the set’s “elements”. In maths, sets are written with curly brackets {}, and elements are often represented by letters. So {𝑎, 𝑏, 𝑐} is a set. Its elements are 𝑎, 𝑏, and 𝑐.

In my work for you, we’re mainly concerned with sets of measurements, i.e. of numbers. For example, the set {1.93} has one element, 1.93. This could represent a measurement such as my height in metres.

Sets can have many elements, even an infinite number of elements. We could have a set which contains all the numbers from 1.5 upwards, or all the numbers from 1.5 to 2.3.

Such sets can be depicted as line segments. Conventionally, we draw these horizontally, like this:



Note that this is a *crisp* set. Conventional set theory deals only with crisp sets, so doesn’t bother to use the word “crisp”. But as we’re working with fuzzy sets too, we need to make clear which is which.

So, the above is a crisp set. Every number is either 100% *in* the set, or 100% *not in* it. Here, the numbers 1.5 to 2.3 are in; all others are not in.

Another way to think about the above set is that we can represent it by labelling every number with either 0 or 1. So we label all the numbers between 1.5 and 2.3 with 1. All the numbers < 1.5 get labelled with 0. So do all the numbers > 2.3. The 0’s and 1’s are membership values: 1 means “in”, and 0 means “not in”.

Diagram, schematic

Description automatically generated

## Fuzzy Sets

That’s for a crisp set. For a fuzzy set, we just say that as well as membership values *of* 0 and 1, we can have values anywhere *between* 0 and 1. We can represent this in a graph, by plotting membership on the y axis, like this:

Diagram

Description automatically generated

As is conventional, I’ve used the symbol μ (Greek letter mu) for membership. My Greek tutor — I once lived in Athens — would be horrified at how badly I’ve handwritten it, but I’m not good at drawing with a mouse. Anyway, the graph represents a fuzzy set for the measurement ℎ, which could, for instance, be height again. To find out what the membership value for any particular ℎ is, we just draw a line upwards from that ℎ until it meets the curve, then go leftwards to get its μ value.

## Representing Fuzzy Sets in the Computer

I said we can think of fuzzy sets as labelling every possible measurement with a membership value. That’s a nice point of view mathematically, because it leads on in a natural way from crisp sets. But it’s not practical for computer use, because we’d have to store an infinite quantity of measurements and labels.

Instead, we represent fuzzy sets as functions from x to y. To make things even simpler, fuzzy-logic people often restrict these to simple straight-line shapes:



In the above, these are: a line sloping down from μ=1 to μ=0; a line sloping up from 0 to 1; a triangle, sloping up and then down; and a trapezoid. The top of the trapezoid is assumed to be level at μ=1. The triangle and trapezoid don’t have to have the same slope on their left and right sides.

You might now think that this is too restrictive. Don’t we want smooth curves, for greater precision? But the concepts the fuzzy sets are representing are approximate anyway, and apparently the simple shapes above are often good enough.

You probably don’t need to know this, but these simple shapes are also very economical inside a computer. For example, consider a triangle whose basepoints are at 1.8 and 2.5 and whose peak is at 2.1. My code just represents that as a data structure or record with three fields:

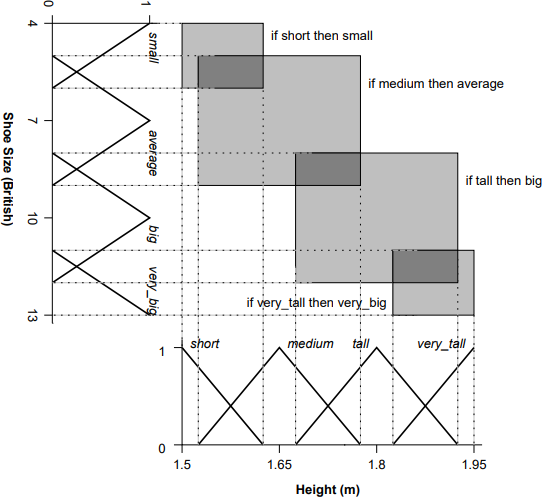
tri( 1.8, 2.1, 2.5 )

## A Published Example: Mapping Height to Shoe Size

I’m now going to go on to an example published in a paper on fuzzy logic and expert systems. It’s only small, but it would be easy for you to explain to a non-expert audience, and it uses the same techniques as more realistic examples. It’s also a good example for me to test my code on.

The example is a simple expert system which predicts a man’s shoe size from his height. Be warned that it uses British shoe sizes, on a scale that goes from 4 to 11. A quick Google showed me that Canadian sizes differ from British by either one size or half a size, depending on which website I read. However, this doesn’t affect the principle. It might make a nice discussion point with which to engage your audience.

The paper is "Animated Fuzzy Logic”, by Gary Meehan and Mike Joy, *J. Functional Programming* 1, January 1993, and the example starts on page 15. First, here is a graphical representation of the expert system’s rules:



This is Figure 8 on page 15, “The fuzzy rule base for the height → shoe size expert system”.

To understand it, we need first to understand the fuzzy sets it uses. Look at the height scale at the bottom. This is in metres, and has four fuzzy sets with it: short, medium, tall, and very\_tall. The sets medium and tall are given as triangles; short is a downwards line; and very\_tall an upwards line. As in my drawings, membership runs from 0 to 1 on the y axis.

The shoe size scale has four more fuzzy sets: small, average, big, and very\_big. In Figure 8, it’s rotated by 90°, but it still works in the same way as height.

Next, let’s look at the rules. The authors write these as follows:

if short then small  
if medium then average  
if tall then big  
if very\_tall then very\_big

Intuitively, these say that short men need small shoes, medium men need average shoes, and so on. In effect, the authors have expressed a correlation between height and shoe size. They’ve done this by dividing the height domain and the shoe size domain each into four zones, then pairing each height zone with a corresponding shoe-size zone.

Another way to say this is that the authors have approximated a height-to-shoe-size function. They’ve done so by dividing it into patches, where each patch is expressed by one rule.

## How It Works: Domains

I need to start by showing the knowledge base I wrote. I had to translate the rules from that paper into a notation that my expert-system shell understood. But as well as the rules, I had to give it information about the domains, i.e. about heights and shoe sizes and the fuzzy sets the paper imposes upon them. This is it:

low( shoe\_size, 4 ).  
high( shoe\_size, 13 ).  
scale( shoe\_size, 'British shoe sizes' ).  
fuzzy\_set( shoe\_size, small, down( 4, 6 ) ).  
fuzzy\_set( shoe\_size, average, tri( 5, 7, 9 ) ).  
fuzzy\_set( shoe\_size, big, tri( 8, 10, 12 ) ).  
fuzzy\_set( shoe\_size, very\_big, up( 11, 13 ) ).

low( height, 1.5 ).  
high( height, 1.95 ).  
scale( height, metres ).  
fuzzy\_set( height, short, down( 1.5, 1.62 ) ).  
fuzzy\_set( height, medium, tri( 1.53, 1.65, 1.77 ) ).  
fuzzy\_set( height, tall, tri( 1.68, 1.8, 1.92 ) ).  
fuzzy\_set( height, very\_tall, up( 1.83, 1.95 ) ).

The details of notation are unimportant — these are actually data structures in Prolog — but what is important is the information conveyed. One aspect is the fuzzy sets’ shapes. These are specified by the down, tri and up. Thus, tri( 5, 7, 9 ) represents the fuzzy set average. It is a triangle with a peak at a shoe-size value of 7, and with base points at 5 and 9. To make this clear, here are the sets represented by down( 4, 6 ) and tri( 5, 7, 9 ) :

Diagram, schematic

Description automatically generated

Please note three things. First, the values here are not quite the same as in the paper. The paper doesn’t state the actual numbers, it only shows graphs. So I had to look at Figure 8 and guess the positions on the x axes.

Second, the notation here is convenient for me to write, and easy to make Prolog use. But I am more used to typing than a lot of people, and to Prolog notation. In the final version of the system, you may want something such as input via a graphical user interface. We can do that, basically putting it on top of what I already have. For the demo, however, I suggest we stick with this. It will probably hardly get seen, anyway.

And third, there is nothing magic about my notation. Any expert system will need some way to represent sets and scales, but it doesn’t need to look exactly like the above, or to use the words “down” and “tri”, and so on.

## How It Works: Plotting Domains

What do the height and shoe-size domains, with their fuzzy sets, actually look like? I needed to be able to find errors, including errors in my fuzzy sets. To help with this, I wrote some graphing routines, including one which plots the fuzzy sets in a domain. It’s demonstrated below. In the screenshot, the window on the right is Prolog. I start it up and type commands to read the shoe-size knowledge base, set a height value of 1.75 metres as input, and plot the height domain. My code pops the graph up as a window displayed in Microsoft Photos:Graphical user interface

Description automatically generated

I was able to plot the shoe-size domain in the same way. This is it: Graphical user interface

Description automatically generated

I wrote other versions of these routines, to do things such as plotting membership tests. You’ll see these later; and some get used by the explanation-generator which I talk about below. I think it will be highly useful to have equivalents of these in the final system, as it will help whoever designs or maintains the knowledge bases.

## How It Works: A Single Rule

To recap, I’ve explained the ideas behind the height-to-shoe-size example, displayed the paper’s Figure 8, showing how the authors split it into four rules, and shown the domains used and how I specified these in my knowledge base. I’ll now deal with my rules. Here are my translations of those in the paper:  
  
if height be short then shoe\_size := small.  
if height be medium then shoe\_size := average.  
if height be tall then shoe\_size := big.  
if height be very\_tall then shoe\_size := very\_big.

My notation isn’t quite the same as in the paper. The reasons for this aren’t important, but one is that I didn’t want to use the word is in the condition, because it means something special to Prolog. Hence the be. I think this is readable, since English does still use “be”, as a subjunctive in phrases such as “If this be treason” and “If this be error and upon me”.

As with the domains, in a final version of the system, we can settle on whatever rule notation is best for users, possibly including graphical entry of rules.

Going back to how rules work, let’s see what the system would do if the knowledge base contained only the single rule  
if height be tall then shoe\_size := big.

The first step is for the system to evaluate the condition, height be tall. This is a membership test. For it, the system needs the value of height. Now, I’ve cheated here. In a real expert system, the user would be asked for this data, perhaps inputting it via a menu. But in this example, I’m testing that the core of the system works, and I don’t care how the inputs get into it as long as they do. So I’ve placed the value directly into Prolog’s memory:Graphical user interface

Description automatically generated

## How It Works: Rule Conditions And Membership Testing

If we look at the fuzzy sets in the height domain from my screenshot above, and drawing a line upward from 1.75, we can see that membership of this height in tall is somewhere around 0.6.

I can refine this by asking Prolog. In the expert system shell, I implemented several diagnostic commands, so that I could sit back and probe the rules and domains and the system’s understanding thereof. One command, plot, I’ve already shown. Another is eval. This calculates the results of various expressions and displays them on the terminal. The expressions can be membership tests, which plot also accepts. Using either or both of these as in the screenshot below, I see that a height of 1.75 metres is regarded as tall with a membership of 0.58 So a bit tall, but not very.Graphical user interface

Description automatically generated with medium confidence

Anyway, we are interested in evaluating the condition height be tall in the rule  
if height be tall then shoe\_size := big.

And the above has done that. In a crisp rule, a condition would evaluate to either true or false, corresponding to the crisp membership values *of* 1 or 0. In a fuzzy rule, it evaluates to the fuzzy version thereof, namely a membership value *between* 1 and 0. We now have that. It’s 0.58.

## How It Works: Rule Conclusions

Having got a membership from the condition, we must now apply it to the rule’s conclusion, shoe\_size := big. What we do is to weight the fuzzy set big with it. The idea behind this is that in general, we’ll have several rules, each concluding a particular fuzzy set. We want all these sets to be considered, but we want some to be considered more strongly than others. These, the strong ones, are those whose conditions gave the highest memberships.

So we weight the set big by multiplying it by 0.58. Here’s what this looks like. Once again, I’ve plotted from Prolog, asking it to plot “the fuzzy set big, weighted by the membership of height in tall”. That is, the fuzzy set which is the bigger triangle below, weighted by 0.58. The plotting routine shows both the weighted and unweighted fuzzy sets, to help compare with other plots.Graphical user interface

Description automatically generated with low confidence

## How It Works: Combining All the Rules

To recap, I’ve shown how the system evaluated one rule,   
if height be tall then shoe\_size := big.  
It worked out the membership fof the input height, 1.75 meters, in the fuzzy set tall. The result was 0.58, and it then multiplied big by this, squishing it down to 58% of its former size.

One of the important things, I think, is that this kind of expert system considers all the rules together. So how does this work? The entire shoe-size knowledge base is four rules:  
if height be short then shoe\_size := small.  
if height be medium then shoe\_size := average.  
if height be tall then shoe\_size := big.  
if height be very\_tall then shoe\_size := very\_big.

To evaluate this, the system just does what I described on all the rules. This gets it four weighted sets:

Graphical user interface

Description automatically generated

Two of these are zero, so in effect, the rules did not contribute. But two, for average and big, did.

Next, the system adds all the sets. This gives it a complicated shape representing each one’s contribution. Note that unlike in all my other plots, there are some points greater than 1. That doesn’t matter, because this more complicated set is only temporary.  
Graphical user interface

Description automatically generated with low confidence

As a side-issue, my labelling hasn’t worked well here. I approximated this shape by taking membership values at 100 points. As I didn’t have a name (such as “big”) for the set, the grapher used the list of points, wildly overflowing the label space. This is the kind of thing that hits user-interface designers, and that we’ll need to sort out. There’s a tension between names which tell a lot about internal content but may be too long to place easily, and names which are short symbols: easy to write and read, informative if you know what the symbol means, but otherwise opaque as to content.

## How It Works: Defuzzification and Centroids

What we now have is a set that is the sum of as many weighted sets as there are conclusions. The next and final step is that the system must “defuzzify” this. The point here is that, so far, we’ve got a distribution of shoe sizes. In effect, this summarises the advice from all the rules. But, when you saunter into your shoe shop and order a pair of trainers or Polyveldts, a distribution won’t do. You must order one specific size. So we need to convert this shape spread out along the x axis into one point thereon. This is *defuzzification*.

There are various ways to defuzzify, but with the knowledge base used here, a good method is to calculate the *centroid* of the shape. In physics, this is defined as the centre of mass of a geometric object of uniform density. The meaning for fuzzy sets is related: the centroid is the x value that, if you’re only allowed to pick one, best represents the set. It’s the centre of gravity of the available data, so to speak. Mathematically, it’s a weighted average Σₓxμ(x) / Σₓμ(x) , where x ranges over all the points in the domain, and μ is the membership function for the set being centroided.

When I do this, my code tells me a value of 9.33. That’s about halfway down the left-hand side of the second peak of the added sets, here:

Chart

Description automatically generated

So, the best prediction of shoe-size for a man 1.75 metres tall is 9.33. In a more realistic system, we’d probably want that rounded up to the nearest half-size, and displayed with a warning that it may not be an exact fit.

To recap the entire process:

1. The system looped over the knowledge base and evaluated each of the four rules. It did this by:
   1. Evaluating each rule’s condition. This involved testing membership of the input value for height in the fuzzy set named in the rule’s condition. This gives a single number μ for each rule.
   2. Weighting the fuzzy set mentioned in the rule’s conclusion. This shrinks it vertically, multiplying every y value in it by μ.
2. It then combined the resulting four weighted fuzzy sets by adding them. This gave it a single, more complicated, shape.
3. Finally, it defuzzified this shape by calculating its centroid. This gave it a single value for shoe size.

## The Need for Explanations

I’m now going to turn to explanation. Many researchers in expert systems have been keen that they should be able to justify the advice they gave. They therefore built an explanation feature into their creations. For instance, Cardiff University’s Dave Marshall has a set of lecture notes on expert systems at <https://users.cs.cf.ac.uk/Dave.Marshall/AI1/mycin.html> . In it, he writes:

EXPLANATION

This mode allows the system to explain its conclusions and its reasoning process. This ability comes from the AND/OR trees created during the production system reasoning process. As a result most expert systems can answer the following why and how questions

**why was a given fact used?**

**why was a given fact not used?**

**how was a given conclusion reached?**

**How was it that another conclusion was not reached?**

Researchers have stressed the importance of this ability. One, whom I knew personally, was Donald Michie ( <https://en.wikipedia.org/wiki/Donald_Michie> ), a codebreaker at Bletchley Park and a pioneer of British Artificial Intelligence. In their 1984 book *The Creative Computer*, Michie and his co-author Rory Johnston wrote:

Taking the opportunities [of computers] will not be easy. It will require a complete reversal of the approach traditionally followed by technology, from one intended to get the most economical use out of machinery, to one aimed at making the processes of the system clearly comprehensible to humans. For this, computers will need to think like people. Unless the computer systems of the next decade fit the ‘human window’ they will become so complex and opaque that they will be impossible to control. Loss of control leads merely to frustration as far as many applications now are concerned, but when society becomes more dependent on computers, and where such things as military warning systems, nuclear power stations and geopolitical and financial communications networks are operated by them, loss of control can lead to major crisis.

Notice these phrases: “making the processes of the system clearly comprehensible to humans”; “think like people”; “fit the ‘human window’”; “will become so complex and opaque that they will be impossible to control”. The authors clearly want systems not to be opaque.

Similarly, in “Experiments on the Mechanization of Game-Learning”, *Computer Journal* Vol. 25, 1, (1982), Michie writes:

It will not be desirable for control rooms in nuclear power stations, air traffic control centres, and the like to become polluted with uncomprehended descriptions generated by their associated computing systems.

Notice too the applications that Michie and Johston mention, which include military warning systems. There had already been many occasions when these systems almost started World War III ( <https://en.wikipedia.org/wiki/List_of_nuclear_close_calls> ), with causes as diverse as solar flares, circuit errors during power cuts, moonrise, swans, and a faulty satellite warning system. We were saved from that one by one man, Stanislav Petrov ( <https://www.bbc.co.uk/news/world-europe-24280831> ). I’m sure Michie and Johnston had these in mind.

They follow the quoted paragraph with this:

The prospect of machines becoming as capable and powerful as we describe can be daunting, even frightening. The notion of something non-human applying thought and judgement appears to encroach on what the human holds most dear: his consciousness.

I suspect that part of what they’re saying here is implied rather than stated outright. It’s the fear of something that thinks *as well as* a human, but not *like* a human. An alien mind, with thought processes we can’t understand. Making that mind explain these processes can allay that fear.

It’s now not 1984, but nearly 2024, forty years later. Computer systems are vast, ubiquitous, and unintelligible. Google won’t explain its search rankings; YouTube won’t explain its video recommendations; Twitter won’t explain why it promotes *those* tweets but not *these* tweets. There’s no point in adding to this opacity, and I hope the explanations that I’m building into your system will be a selling point. In the next section, I return to technical matters and explain how your system explains itself.

How It Works: Explanations as Graph Plots  
Your system outputs an explanation on its results page, after the main result. This is a record of the rules used, in the order they were visited. For each rule, the system records the rule’s condition and the membership value it yielded, and also notes whether the numbers in the condition were derived from other rules or from inputs. It also records the fuzzy set returned by the rule’s conclusion, and the weight it got given —which as we now know, is the same as the condition’s membership value. As well as recording results from single rules, the system stores information about how it combined them. This includes the result of adding all the weighted fuzzy sets together, and the centroid of that result.

To show these, the system depicts them as graphs. These summarise the “How it Works” sections above. There is a single graph for the condition of each rule, which shows the fuzzy set tested against an input, and the input’s degree of membership. There is another graph for the conclusion, which shows how this weights the output set. And at the end of the list of rules is a graph which plots the sum of all the output sets, and its centroid.

## Another Published Example: Pricing Goods

I’ve one more example, also from the “Animated Fuzzy Logic” paper, from page 20 onwards. It’s about pricing goods. Consider a company whose Sales Director wants its goods to be as cheap as possible to maximise sales, while the Finance Director wants them as expensive as possible, to maximise takings. They also want a healthy profit: say a 100% mark-up on the cost price. Finally, they have to consider what their competition is charging.

These four requirements can be stated as rules:

1. Our price must be low.
2. Our price must be high.
3. Our price must be around 2 × the manufacturing cost. That is, a 100% mark-up.
4. If the competition price is not very high, then our price must be around the same value as the competition price. We don’t want to start a price war.

These rules introduce several new concepts, which I’ll explain one by one. All are characteristic of fuzzy expert systems, so worth knowing.

## Reconciling Incompatible Information

The first thing that jumps out is the clash between rules 1 and 2. How can a price be low and high at the same time? In a crisp system, it couldn’t, unless the price scale was so narrow that the low and high points were the same. But a fuzzy system gets around this by having degrees of membership. So a price can be low to a certain degree, and high to a certain other degree, and the two may be compatible.

As I did with shoe-sizes, I’ve translated the pricing example into the notation used by my expert system shell, and stored it in a knowledge base. I’ll show you part of this, as it will make things more precise. Here are my fuzzy sets, endpoints, and units:  
low( price, 15 ).  
high( price, 35 ).  
scale( price, 'Pounds sterling' ).  
fuzzy\_set( price, low, down( 15, 35 ) ).  
fuzzy\_set( price, high, up( 15, 35 ) ).

The endpoints are 15 and 35, so the company is assumed to be choosing a price that lies between £15 and £35.

And there are two fuzzy sets, called low and high. Their names have no direct connection with the low( price, 15 ) and high( price, 35 ) : it’s just a coincidence of name. Here are the sets, plotted against the price scale. As before, I used the interface I wrote wherewith I can issue diagnostic commands, including plotting commands. Note that the set low slopes downwards from the left endpoint, and the set high slopes upwards to the right endpoint. You’ll have become familiar with this pattern, having seen it with the height and shoe\_size scales.

Graphical user interface

Description automatically generated

Now here are the first two rules:  
price := high. (“Our price must be high.”)  
price := low. (“Our price must be low.”)  
  
Unlike before, these are *unconditional*, meaning that here is no if … then part. So how does the system handle these? Well, in my section “How It Works: Combining All the Rules”, I explained how it would handle the *conditional* rule if height be tall then shoe\_size := big. It worked out the membership of the input height, 1.75 meters, in the fuzzy set tall. The result was 0.58. It then multiplied the fuzzy set big by this, thereby pushing it down to 58% of its former size.

In other words, the system took the fuzzy set big mentioned in the conclusion, and weighted it, giving a new fuzzy set. It does the same here, so for the first rule, it takes the fuzzy set high. The difference is that there’s no condition to set a weight, so it doesn’t weight it. Or to put it another way, it weights it by 1.

Likewise, for the second rule, the system takes the fuzzy set low and leaves it unweighted.

Next, the system has to combine these sets. In “How It Works: Combining All the Rules”, I said:

“It then adds these. This gives it a complicated shape representing the contribution of each of those weighted sets. Note that unlike in all my other plots, there are some points greater than 1. That doesn’t matter: this more complicated set is only temporary anyway.”

And indeed, something similar happens here. The authors have introduced one difference, though.

This is that when combining low with high, the system takes the *minimum* of every pair of memberships, rather than adding them. I won’t justify this in detail, but I suspect they felt that doing so would lead to less bias. Technically speaking, they’ve taken the fuzzy intersection of low and high, which is the fuzzy-logic version of finding all elements contained in both sets rather than just one. At any rate, the resulting fuzzy set is an isoceles triangle peaked at 25, midway up the price scale:

Graphical user interface

Description automatically generated with medium confidence

Finally, the system has to defuzzify this shape, converting it from a distribution of values to one value. It does this in the same way as with shoe sizes, by calculating the centroid or weighted average. This set is so symmetrical that we can do so by eye: the centroid is at 25.

## Fuzzy Numbers

Now let’s look at some other new concepts. In the last section, I explained how the system reconciles two of the rules:  
price := high.   
price := low.   
  
However, we actually have four rules:  
price := high.  
price := low.  
price := around( 2 \* 13.25 ).  
if 29.99 be not(very(high)) then price := around( 29.99 ).  
  
These translate as:

1. Our price must be high.
2. Our price must be low.
3. Our price must be around 2 × the manufacturing cost. That is, a 100% mark-up.
4. If the competition price is not very high, then our price must be around the same value as the competition price. We don’t want to start a price war.

In them, I’ve written the manufacturing cost and the competition price as constants 13.25 and 29.99. In real life, we’d ask the user to input them, but the example is shorter if I embed them in the rules.

This example introduces two new ideas. These are “fuzzy numbers” and “hedges”. A fuzzy number is a fuzzy set centered around a number, and generated by a fuzzy qualifier such as “around”, “near”, or “roughly”. Using triangular fuzzy sets is usually good enough, with the width of their base depending on the qualifier. Experiment has show that “around”, for instance, is well approximated by a triangular fuzzy set whose base is 0.5 times the width of the domain. Here is a plot of around(20) for the price domain:

Graphical user interface

Description automatically generated

## Hedges

Now that we’ve done fuzzy numbers, what are hedges? These are qualifiers such as “very”, “extremely”, “somewhat” or “slightly”, applied to fuzzy sets. Fuzzy-logic people often represent these by raising the underlying set’s membership to a power. For example, the price 29.99 has a membership 0.75 in “high”. So in “very high”, its membership is 0.752, or roughly 0.56.

Because of the powers, hedged fuzzy sets usually have curved boundaries. You can see this below with “very”.

Graphical user interface

Description automatically generated with medium confidence

## Complement or “Not”

Now I’ve done fuzzy numbers and hedges, I’m going to talk about operations on sets. Crisp set theory has an operation called “complement”. Consider the set of prices between 23 and 37 inclusive. Then its complement is all the other numbers in the domain, i.e. those ≥ 15 but < 23, and those ≤ 35 but > 27. In this diagram, the set of prices between 23 and 27 is the top set; the bottom set is its complement.

Application

Description automatically generated with low confidence

This can be extended to fuzzy sets. I explained that membership of crisp sets can be thought of as labelling every object with either 0 or 1. Something is in the set if its label is 1, out of the set if its label is 0. Complement inverts the label. For fuzzy sets, we generalise this by allowing the label to have any value *between* 0 and 1. We say that for any fuzzy set F, x’s membership in the complement of F is 1 minus x’s membership in F. Because 1-0=1, and 1-1=0, this is consistent with the crisp-set definition, which is good.

Often, complements are easy to do by eye. Thus, the “not” of an upward-sloping line is a downward-sloping one. This plot shows the complement of high in the price domain:

A picture containing graphical user interface

Description automatically generated

This is actually the same as the fuzzy set low.

If the set being complemented is a triangle, its complement will be a valley, with lines equal to 1 extending left and right to the end of the domain. Thus, the plot below shows the complement of average in the shoe\_size domain:

A picture containing graphical user interface

Description automatically generated

Because I need to improve the plotting commands, this plot doesn’t show the area each side of the valley. However, there is one, and it extends all the way to the ends of the domain. Like this:

Diagram, line chart

Description automatically generated with medium confidence

So one use of complements is to make fuzzy sets with “holes” in.

So when should we use complements? You may find crisp rules easier to think about than fuzzy ones. If a crisp rule would use a crisp complement, then ask yourself whether a fuzzy complement would make sense in the corresponding fuzzy rule. It often will.

A final note on notation. Set theorists write complement as an overbar, prime, or minus sign. The first is difficult on most keyboards, and the other two could be ambiguous. Therefore, some implementors prefer the word “not”. That’s what the “Animated Fuzzy Logic” paper used, and I’ve done the same.

## Union and Intersection: “Or” and “And”

Complementing transforms one set. There are also operators that combine two sets. Crisp set theory has two particularly fundamental ones, union and intersection. Their mathematical symbols are ∪ and ∩. As with complement, the operators can be extended to fuzzy logic.

Actually, I haven’t used them at all in the conditions of rules, so I shall skip a description of them here. But “Animated Fuzzy Logic” does use intersection to combine the *results* of the pricing rules. As I said in the section on reconciling incompatible information, I suspect they felt that doing so would cause less bias. For the purposes of this report, it suffices to say that this is an alternative to adding the sets as the shoe-size example did. It is analogous though, in that each rule generates a weighted fuzzy set, you put them all together, and you then take the centroid.

## Putting It All Together

I’ve introduced fuzzy numbers, hedges and “not”, and I now need to get back to the pricing knowledge base and how it works. Here are the rules again:

price := high.  
price := low.  
price := around( 2 \* 13.25 ).  
if 29.99 be not(very(high)) then price := around( 29.99 ).

The principle is the same as before. If a rule is conditional, the system works out its condition, getting a membership value. It then weights the fuzzy set named in the rule’s conclusion by that value. If the rule is not conditional, it works out the conclusion, but doesn’t weight it.

The system now has as many fuzzy sets as there were rules. It now combines them into one. For the shoe-size rules, this added the sets; but here, pricing rules, it does the fuzzy equivalent of taking their intersection. This gives the following strange shape:

A picture containing chart

Description automatically generated

And finally, the system takes the centroid. When I do this, I get a value of 28.6. So the price that best reconciles the advice in those rules is £28.6.

## Recap

In this report, I’ve introduced the following ideas:

**Fuzzy concepts.** I.e. concepts such as “tall”, “wet”, “profitable”. Whereas there is a sharp boundary between the concepts “odd” and “even”, there is not one between “tall” and “not tall”. Instead, there’s a gradual transition.

**The word “crisp”.**

**Crisp sets.** With a crisp set, an object is either in it, or not in it. There is no half-way stage. There are various ways to represent this: one is to label every object with 1 if it’s in the set, or 0 if it isn’t.

**Fuzzy sets.** With a fuzzy set, there are degrees of membership. We can think of this as extending the idea of labelling, so that we label every object with a number between 0 and 1. 0 still means “not in”, but we can now also have membership values like 10% in or 57.3% in.

**μ.** The lower-case Greek letter mu is conventionally used for membership of fuzzy sets. We often write μF(x) to mean the membership of x in fuzzy set F, omitting the F when it’s clear which set we’re talking about. So if I write μtall(1.75), I mean the membership of 1.75 metres in the fuzzy set “tall”.

**Depicting fuzzy sets.** We are mainly concerned with fuzzy sets of numbers, which are usually measurements. For these, membership works like an x-y plot. Given a fuzzy set F, any number x will have a degree of membership μF(x) in F. We can plot this as a graph, with μ(x) up the y axis.

**Polygonal fuzzy sets.** It turns out not to be terribly important what the exact shape of a fuzzy set is. So to economise on memory, we represent them as simple straight-line shapes — polygons — where possible. The main shapes are: a line sloping up from 0 to 1; a line sloping down from 1 to 0; a triangle peaking at 1; and a trapezoid with a flat top at 1.

**Patches, or piecewise approximation.** The shoe-size example can be regarded as a function mapping a person’s height to their shoe size. This is defined by several rules. Each rule defines a piece or “patch” of that function. Things aren’t actually quite that simple, because we combine the results from all rules, but it’s a good intuitive view.

**Domains.** When writing a knowledge base, we need to tell the system which scales, or systems of measurement, we’re using, and where their endpoints are. The latter saves resources, as the computer can ignore stuff outwith the endpoints. It also informs the grapher which x-values it should plot against.

**The anatomy of rules.** Rules can be conditional or unconditional. Conditional rules are often written with “if” followed by a condition followed by “then” followed by a conclusion. Unconditional rules lack the condition. A condition may be a simple test such as “height is tall”, or it may be several tests combined with logical operators such as “and”, “or”, and “not”. In the example here, I haven’t used these. Note that the exact notation will depend on the expert-system shell. I chose mine to be convenient to read, convenient to explain, convenient to implement, and to match examples from the papers I’m citing.

**How a single rule works.** Exact details will depend on the system, but will be roughly as I’ve described. In my system, every condition yields a membership value. So in the shoe-size example, the condition “height is tall” yields a μ of 0.58 if height is 1.75 and tall is as plotted earlier. Having evaluated a rule’s condition in this way, the system then looks at the fuzzy set named in its conclusion, and weights it by this value. So “if height is tall then shoe\_size := big” yields a fuzzy set for which each μ(x) is 0.58 times μbig(x). If the rule doesn’t have a condition, the weight is 1.

**How multiple rules work.** If there are several rules for shoe size or whatever, the system evaluates each one as above. That gives it a list of weighted fuzzy sets, as many as there are rules. It then combines these. In the shoe-size example, it combined them by adding. In the pricing example I describe in my next report, it will combine them by taking mimumum values. Different problems may need different kinds of combination. At any rate, however the fuzzy sets get combined, we end up with one, probably a rather complicated shape. We then defuzzify it to convert it to a single value.

**Defuzzification.** The process of converting a fuzzy set to a crisp value, as above. Needed because you can’t (for instance) go into a shoe shop and order a distribution of sizes: you have to buy just one. In the example here, I take the centroid.

**Centroid.** In physics, this is defined as the centre of mass of a geometric object of uniform density. The meaning for fuzzy sets is related. The centroid is the x value that, if you’re only allowed to pick one, best represents the set. It’s the centre of gravity of the available data, so to speak. Mathematically, it’s a weighted average Σₓxμ(x) / Σₓμ(x) , where x ranges over all the points in the domain, and μ is the membership function for the set being centroided. The algorithm that calculates the centroid has to decide which and how many x values to use, as there are a potentially infinite quantity. In practice, I’ve found that 100 values equally distributed over a domain give good results. As a technical aside, for “continuous” domains where any value at all between the endpoints is allowed, the formula should, strictly speaking, use integrals rather than summations. However, summation is good enough for every case that I know of.

**Explanations.** It’s important that users know why programs give the advice they do. For our expert system, we can record the rules visited, the memberships their conditions evaluated to, and so on. We can then present this to the user as a trace containing graphs. If there’s too much such information and it overwhelms the user, the final version of the system will need tools to help them navigate it.

**Reconciling incompatible pieces of information.** The pricing example showed how we can do this. It was perfectly happy with the two rules “price := low” and “price := high”.

**Fuzzy numbers.** In English, we use phrases such as “around 20”, “roughly 50”, and “60-ish”. These can be carried over into fuzzy logic as so-called fuzzy numbers: fuzzy sets derived from a crisp number and a fuzzy qualifier. The third and fourth pricing rules use these. Thus, the third rule says that — to give a decent profit — the product’s selling price should be around twice its manufacturing cost. This is easily represented by using a triangular fuzzy set centered on the crisp number. The width of the base determines the set’s fuzziness: experiment has shown, for instance, that for “around”, a base which is 0.5 times the length of the domain works well.

**Hedges.** We also use words such as “very” and “slightly”, as in “very tall”. These are called hedges. It’s been found that these can be represented by raising the membership in the underlying set to a power. Thus, “very” is often represented by squaring, so if 1.75 metres has a membership of 0.58 in “tall”, its membership of “very tall” is 0.582 or 0.34. Because of the powers, hedged fuzzy sets are usually curved. I approximate them by polygons made from around 100 points.

**Not.** The fourth pricing rule says that if the competition’s price is not very high, then our price should be around the competition price. This combines several ideas. There’s a fuzzy number: “around”. There’s a hedge: “very”. And there’s something new: the set operator “not”. In crisp logic, “not” inverts membership values, so not 1 is 0, and not 0 is 1. In fuzzy logic, it generalises this, subtracting the value from 1. So μnot(F)(x)=1-μF(x): x’s membership in not(F) is 1 minus x’s membership in F. This is consistent with crisp logic for memberships of 0 and 1. Some fuzzy sets are simple enough that you can do “not” by eye. For instance, the “not” of a line that slopes up from 0 to 1 is a line that slopes down from 1 to 0. Note that in crisp set theory, “not” is called “complement”. It’s often written as an overbar, prime, or minus sign, but all have disadvantages on the keyboard.

**Or, and.** As well as complement, crisp set theory has operators “union” and “intersection”, written in maths as ∪ and ∩. These generalise into fuzzy set theory. For two crisp sets F and G, their union is the set of objects in them both together. For two fuzzy sets, the membership of x in the union of F and G is the maximum of x’s membership in F and its membership in G. Intersection works similarly, but takes the minimum. If you have a rule that would make sense with the crisp version of union and intersection, then the fuzzy equivalents will often work too, so it’s worth trying them. In this report, we haven’t seen them in rule conditions, but we did see intersection used to combine the outputs of pricing rules. Note that because keyboards lack the symbols ∪ and ∩, you may see the operators named after their logical equivalents “or” and “and” as here, or after the programming-language equivalents || and &&.