A Fuzzy Logic Primer

Peter Elsea

Fuzzy logic is a mathematical pursuit and an engineering approach. In mathematics, fuzzy logic has been highly controversial, with raging battles (in academic journals, where the weapon of choice is the sharp retort) about the validity of the discipline at all, its appropriateness versus probability theory, and the personal habits of its proponents.

It's mostly about the name. Fuzzy logic is not logic that is fuzzy, it is a logic that deals with fuzzy issues. And when you get right down to it, nearly every real situation involves fuzzy issues. How fast is too fast?

It's also that fuzzy admits that there is such a thing as "partially true." Classically trained logicians really hate that. The arguments are fading out though, especially since it is easy to prove that classical logic is a special case of fuzzy logic.

As an engineering discipline, fuzzy is catching on just about everywhere, although it first flowered in Japan. In the US it was ignored until the mid '90s. It is mostly used in control systems, things like thermostats that adjust for the time of day and angle of the sun, or autofocus in a video camera. Fuzzy is also a good way to make expert systems, because it presents a very straightforward way to encode problems and make decisions.

Review of Crisp Sets

You should be familiar with the set, a collection of things that have some common characteristic. An example would be the set of all people in this classroom wearing coats. Another set might be the set of all those in the classroom wearing scarves. The two sets are distinct. However, each member of each set is also a person, so the set of all persons in the classroom includes both sets. The set of people wearing coats is a subset of the set of persons in the classroom.

Another subset of the set of people in the classroom is the set of those with both coats and scarves. These people belong to two sets, and constitute the intersection of the sets.

Yet another way to group people is those wearing either scarves or coats. This is the union of the two sets.

Now consider the set of all persons in the classroom wearing tennis shoes. There is also the set of those not wearing tennis shoes. Obviously, the two sets are mutually exclusive. Mutually exclusive sets are complements of each other. If we have a set of all people in the classroom wearing hats, there is a complementary set of those not wearing hats.

You've seen diagrams with circles to represent all of this.

I prefer to look at it slightly differently, from the point of view of one person. It can look like table 1.

membership->	√		√	
People ->	W/hats	W/tennies	W/coat	W/scarf

Table 1.

If I have four things that could describe one potential set member[1], I can make a list of the descriptions and just check the ones that apply. Someone wearing a hat and coat, but not tennis shoes or a scarf would get a list like this. The check mark indicates they are a member of the particular set. Being mathematically inclined, I'd probably use a 1 to indicate a check and a 0 to represent no check.

Ralph	1	0	1	0
Cindy	0	0	0	1
George	1	1	1	0
Sonia	1	1	1	1
People ->	W/hats	W/tennies	W/coat	W/scarf

Table 2

At this point, the sets become useful, because we can arrange them in a table, and make observations about how many folks wear scarves and tennis shoes together. See table 2.

FUZZY SETS

Fuzzy logic was invented to consider collections that are not either/or. For instance, the set of all tall persons in the classroom. Well, what do I mean by tall? In the ordinary logic system I have to pick an arbitrary height, say 5' 6.0000", and consider everyone above that to be tall. That means some people could jump from the tall to the not tall set by combing their hair. Anyone who has received a B+ grade understands the awkwardness that can arise from sharp and arbitrary divisions.

For a vague description like tall, it makes more sense to use a fuzzy set. A fuzzy set allows partial membership. It works like this:

Looking at the domain of height, we find a value we consider definitely tall, like 6'. We assign a person 6' tall or taller a membership of 1. We also find a height that is clearly not tall (say 5'), and give that a zero. We fill in the spots between these points along a straight line or some sort of curve, resulting in a membership table like 3:

Tall	0	.02	.08	.2	.3	.4	.5	.6	.7	.8	.92	.97	1
height	5'0	5'1	5'2	5'3	5'4	5'5	5'6	5'7	5'8	5'9	5'10	5'11	6'

Table 3

Now we can describe someone as having 0.7 membership in the set of tall people. That may seem odd, but if you pronounce it "fairly tall", it makes sense.

[1] The set of everything that could possibly be a member of any of the sets I am talking about is called the Universe.

We can assume a simple relationship between the numbers shown. That is, a person 5 feet six and a half inches in height has a membership in {Tall} of 0.55. Everyone above 6'0" has a membership of 1 and everyone shorter than 5'0" has a membership of 0.

We can also create a fuzzy set for "heavy". I'll let you fill in the numbers in table 4.

heavy weight	0	.1	.2	.3	.4	.5	.6	.7	.8	.9	1

Table 4.

Anyway, we can now describe someone in terms of weight and height with a bit of precision, as in table 5.

Jack	.8	.2	.2	.8
George	.4	.7	.4	.7
	heavy	tall	H and T	H or T

Table 5.

Fuzzy Union and Intersection

If Jack has a membership in {heavy} of 0.8 and in {tall} of 0.2, his membership in the set {heavy and tall} is 0.2. How is that? Membership in the intersection of two fuzzy sets is equal to the minimum membership in either. (Remember, all people who are heavy and tall are members of both sets, and that is the intersection.) Likewise, membership in the union of two fuzzy sets is equal to maximum of the membership in either.

If George has a membership in heavy of 0.4 and in tall of 0.7, his membership in {heavy and tall} is 0.4 and membership in {heavy or tall} is 0.7.

What's the point? Well, we now have a mathematical way of making the statement "If a person is heavy and tall, he is strong". How strong? If we set things up right, Jack's membership in {strong} is 0.2 and George's membership in {strong} is 0.4.

Of course height and weight alone don't determine strength. We need to know more about Jack and George. George goes to the gym five times a week, and Jack is a university teacher. So let's change the statement to "If a person is heavy or tall and works out, he is strong". Assigning a membership in {works out} of .9 for George and 0.1 for Jack (he walks to class), we get:

George: {strong} = {{0.4 heavy} or {0.7 tall}} and {0.9 works out} = 0.7
Jack: {strong} = {{0.8 heavy} or {0.2 tall}} and {0.1 works out} = 0.1

You can see that the fact that Jack does not work out limits his strength, whereas George can probably ease up a bit. You can also see that it should not be much effort to modify the formula to include other factors like health and nutrition. The ability to easily add factors is one of the best features of the fuzzy system.

Fuzzy Complements

So what is Jack's membership in the fuzzy set of short people? People who are short are not tall, and not implies the complement set. The complement of a fuzzy set is found by subtracting the membership from 1.0. Thus Jack's membership in {short} is 0.8, and George's is 0.3. Here's a diagram that explores the concept:

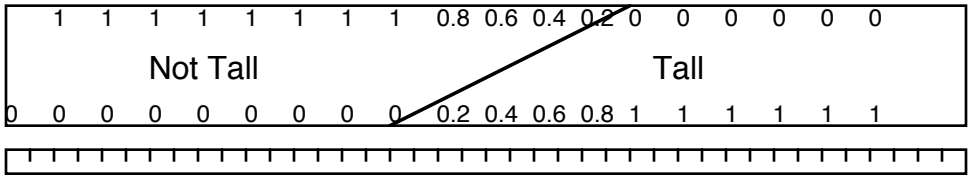

fig. 1

Fuzzy Numbers

Now we get to Fuzzy numbers. These are a way to express the concept of "around 7". These are sets too, along the domain of real numbers with a membership peak at the official value and falling membership nearby.

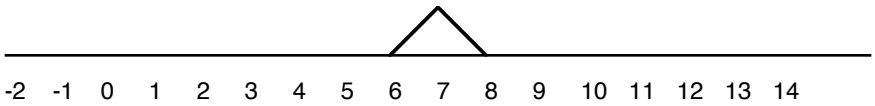

fig 2.

(Drawing shapes like this is a popular way of describing fuzzy sets. The domain is shown across the bottom, and the set is a triangle, trapezoid, or some kind of curve. The highest membership in the set is always 1.)

There are similar concepts for "nearly 7" or "just above 7". Some people do math with fuzzy numbers, but we'll just use them as input to reasoning systems.

Fuzzy Logic

Monotonic reasoning

The simplest reasoning system is one where there is a pretty direct relationship between one factor and another. For instance, you could easily implement the rule "if the car is going too slowly, apply some extra gas." In this case, you would have two fuzzy sets, one representing "too slowly", another representing " extra gas".

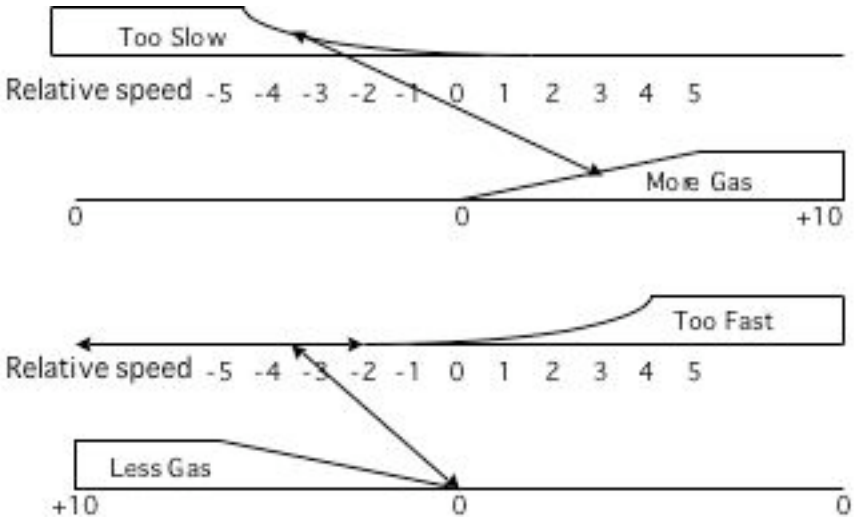

Figure 3

You can see from figure 3 how the two sets would relate. There would be some starting gas setting, probably associated with the speed limit. (Let's assume the target speed is the 0 mark on the set too slow.) As the speed dropped into the too slow range, the membership value for the speed would determine how much extra gas to add. The extra amount of gas is found by reading the domain value for {more gas} at the membership value obtained from {too slow}. The relationship is direct, but it doesn't have to be simple. Note that the concept of too slow has a curved membership function. Fuzzy software interpolates between defined points on a curve, so these things are easy to tweak.

Of course, speeding up is only half the story. Imagine a similar system that implements the rule "if the car is going too fast, let up on the gas." Combining the rules is easy. One asks for more gas under certain circumstances, the other asks for less gas under different circumstances. We just add the results together. The interesting thing is that this works even if the zones for too fast and too slow overlap!

Complex Predicate

Of course, some times you have to shift gears to go faster. You would control the transmission with a group of rules like "If the speed is too slow and the engine rpm is too high, shift up". The word "and" in that statement tells you what to do -- we need to be

satisfying both criteria to get a result, so we are asking for the intersection. That will be the lowest membership value found for a particular speed and RPM. (In practice, you will have a different "too slow" set for each gear.)

Multiple Factors

Even more interesting is the concept of the target speed.
Here are some factors to consider:

- Speed limit (a fuzzy number if I ever saw one)
- Being late
- Weather
- Presence of police

We can codify this into a set of rules

- Drive at a speed near the speed limit.
- If late, speed up
- If road is wet, slow down
- If police have been seen, slow down.

Figure 4 shows how this works. For each rule, we have a predicate and a consequent set. For each, we clip the consequent set to the membership we get out of the predicate. The clipped sets are shown as shaded. Note that "around the speed limit" is always true, so the whole consequent is taken.

From the clipped sets, we generate a composite solution set. The sets may be added, or superimposed as is done in figure 4.

Once we have a solution set, we must "defuzzify" it to extract a single value. There are may ways of doing this, but the most common is to find the centroid or center of gravity of the solution set.

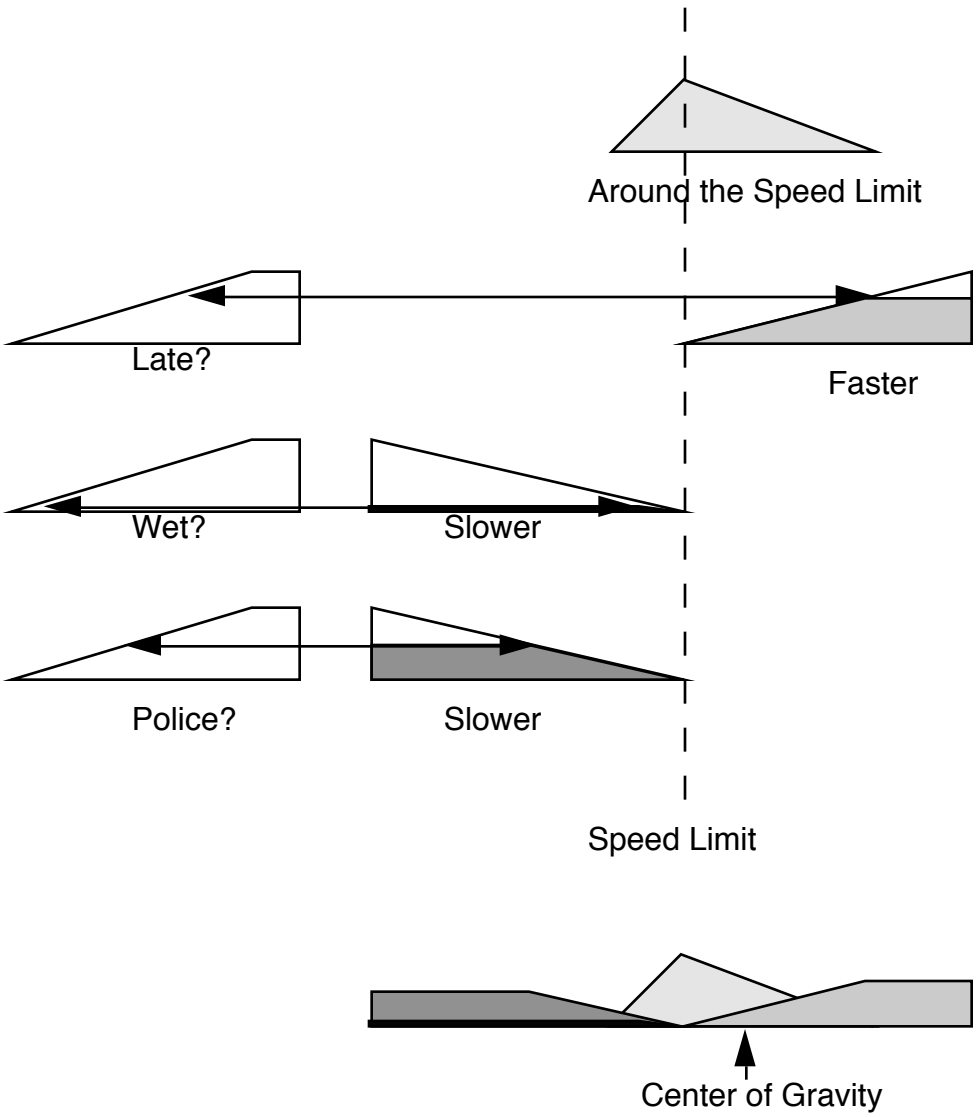

Figure 4. Predicates on the left, consequents on the right.

This gives us a single value to use for target speed.

Effectiveness

These techniques and a few others can be used to give some kind of result in nearly any situation. The question is, are these the right results. The following points have a bearing:

- A fuzzy system is defined by its rules. If a problem is well understood, a fuzzy model is relatively easy to construct. If a problem is only partially understood, a fuzzy approximation is very easy to expand and refine.
- The sets need to be appropriate. The sets usually need a fair amount of adjustment to tune the system. In some cases, fuzzy systems can learn automatically. They are sometimes combined with neural nets, for instance.

- A working system needs to be monitored. After it is working well rules should be selectively removed to evaluate their contribution. Generally you want only enough rules to get the job done.
- On the other hand, the system needs to be tested at the limits of stability, to ensure that it will not break catastrophically.

The best place to find rules is to look at working models. Fuzzy excels at creating expert systems, where experienced operators are available to describe what they do. This is the point of using fuzzy logic in music: we can be our own experts and build our own systems that work the way we want them to.

Fuzzy Pattern Matching

Fuzzy logic is very useful in matching real observations to ideal data. A classic example is music transcription, where a performer will seldom play precisely on the beat or produce sixteenth notes that are exactly half the time of eighths. To resolve these ambiguities, count the number of note onsets within an estimated beat. For any number of onsets, there is a limited number of ideal patterns.

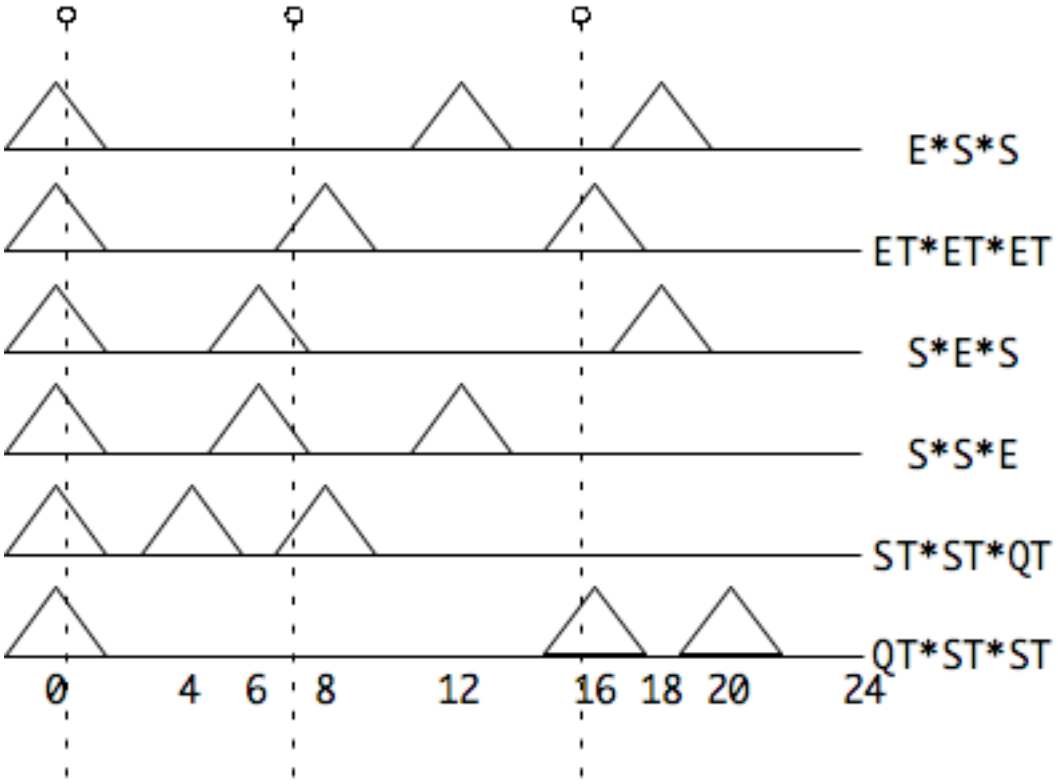

Figure 5.

In figure 5, the sets for various ideal patterns of three onsets within a beat are shown as fuzzy sets. The time is given in ticks (1/24th of a quarter note) across the bottom. The captured note on times are show by the dotted line. To assign a pattern, simply add up the membership values of the three onsets in each set-- these are determined by the points where the dotted lines cross (or miss) a triangle. The highest membership wins.

Resources

I have a long paper on fuzzy logic and music at
ftp://arts.ucsc.edu/pub/ems/FUZZY/Fuzzy_Logic_Tutorial.pdf

Cox, Earl; Fuzzy Logic for Business and Industry : Charles River Media; 1995;

A search on Google for fuzzy logic will give about 200,000 hits
Searching for fuzzy logic music gets my paper and some band in Canada. "Fuzzy Logic" brings up some good tutorials.

Fuzzy Operations in Lisp

The file fuzzy.lisp contains a library of basic fuzzy operations. To demonstrate these, I'll use the following fuzzy sets:

```
(defVar TEST1 '(0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0))
(defVar TEST2 '(1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0))
(defVar TEST3 '(0 0 0 0.2 0.4 0.6 1.0 0.6 0.4 0.2 0.0 ))
(defVar TEST4 '(0 0.2 0.4 0.6 1.0 0.6 0.4 0.2 0.0 0 0))
```

Fuzzy Complement

The fuzzy complement of a set is built by subtracting all members from 1.0.
The function fz-complement does this.

```
? (fz-complement   test1)
(1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.19 0.099 0.0)
```

Fuzzy Intersection

The fuzzy intersection of tow sets is produced by taking the maximum membership from either set a each point. This is produced by fz-intersection.

```
? (fz-intersection   test1   test2)
(0.0 0.1 0.2 0.3 0.4 0.5 0.4 0.3 0.2 0.1 0.0)
```

Fuzzy Union

The fuzzy union of two sets is built from the lowest memberships at each point. The function is fz-union.

```
? (fz-union   test1   test2)
(1.0 0.9 0.8 0.7 0.6 0.5 0.6 0.7 0.8 0.9 1.0)
```

Clipping

Clipping a set reduces any memberships above the clip value to the clip value. This is done by fz-clip.

```
? (fz-clip  test1  0.5)
(0.0 0.1 0.2 0.3 0.4 0.5 0.5 0.5 0.5 0.5 0.5)
```

Adding Lists

Adding lists is not a traditional fuzzy operation, but I often find it useful. Add-lists only operates on two lists at a time.

```
? (add-lists  '(1  2  3)  '(4  5  6))
(5 7 9)
```

Normalization

Normalized lists have their values adjusted so the greatest membership is1.0. It is often necessary after adding lists.

```
? (fz-normalize-list  '(0  1  2  3  4  5  6))
(0.0 0.166 0.33 0.5 0.66 0.833 1.0)
```

Crisp sets

Occasionally, we want to convert a fuzzy set into a traditional ordered set (known to fuzzy aficionados as crisp sets).

```
? (fz-crisp-up  '(0  0.2  0  1))
(0 1 0 1)
```

Bounded Addition

It is often useful to accumulate data into a list by adding a small increment to members corresponding to a samples value, building a histogram on the fly. (It's a good way to keep track of notes played to determine key, for instance). It's important to keep values limits to 1 so that reducing the value is responsive. Bounded-add-to-n adds a value to a specified member only if that member is less than 1.0. The function includes a setf, so the input list is modified.

```
? (defvar  testlist  '(0  0  0  0  0  0  0  0  0  0  0  0))
testlist
? (bounded-addto-n  testlist  4  0.3)
(0 0 0 0 0.3 0 0 0 0 0 0 0)
? (bounded-addto-n  testlist  4  0.3)
(0 0 0 0 0.6 0 0 0 0 0 0 0)
? (bounded-addto-n  testlist  4  0.3)
(0 0 0 0 0.89 0 0 0 0 0 0 0)
? (bounded-addto-n  testlist  4  0.3)
(0 0 0 0 1 0 0 0 0 0 0 0)
? testlist
(0 0 0 0 1 0 0 0 0 0 0 0)
```

Bounded subtraction

Bounded subtraction is the complement of bounded addition, limited to 0. When we detect a note is unlikely to be in a key (for instance after the note below and above are played) we reduce its value in the histogram. this also includes a setf

```
? (bounded-subfrom-n  testlist  4  0.3)
(0 0 0 0 0.7 0 0 0 0 0 0 0)
? (bounded-subfrom-n  testlist  4  0.3)
(0 0 0 0 0.39 0 0 0 0 0 0 0)
? (bounded-subfrom-n  testlist  4  0.3)
(0 0 0 0 0.099 0 0 0 0 0 0 0)
? (bounded-subfrom-n  testlist  4  0.3)
(0 0 0 0 0 0 0 0 0 0 0 0)
? testlist
(0 0 0 0 0 0 0 0 0 0 0 0)
```

A reminder of the contents of some test lists:

```
? test1
(0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0)
? test2
(1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0)
```

Weighting

Weighting is often necessary to adjust the importance of rules in fuzzy operations. It consists of multiplying each member by a scaling factor.

```
? (fz-weight  test1  0.5)
(0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5)
```

Inference

Fuzzy inference is done by finding the index of a specified value in a fuzzy set. Since the set is assumed to be continuous, it is appropriate to return a fractional index if the target falls between two listed values. Fz-find performs this function. If the value is not within the list, fz-find returns nil.

```
? (fz-find  test1  0.5)
5.0
? (fz-find  test2  0.55)
4.49
? (fz-find  test1  2.5)
nil
```

We also need to be able to look between the values to find the interpolated membership at a fractional index. this can be done with the fz-membership function.

```
? (fz-membership  test1  7)
0.7
? (fz-membership  test1  7.5)
0.75
? (fz-membership  test1  11)
1.0
```

Fz-rule performs the fuzzy inference operation. Given a value, a predicate set and a consequent set, fz-rule returns the consequent set weighted by the membership of predicate at value. In the usual example, this is a way of inferring "if he is tall he is heavy".

```
? (fz-rule  1  test1  test2)
(0.1 0.09 0.08 0.069 0.06 0.05 0.04 0.03 0.02 0.01 0.0)
? (fz-rule  7  test1  test2)
(0.7 0.63 0.559 0.489 0.42 0.35 0.279 0.21 0.139 0.069 0.0)
```

In some circumstances, it is more appropriate to clip the consequent set to the predicate membership of value. Fz-clip-rule performs this variation.

```
? (fz-clip-rule  1  test1  test2)
(0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0)
? (fz-clip-rule  7  test1  test2)
(0.7 0.7 0.7 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0)
```

To complete the inference several of these rules are executed and the resultant sets added or unioned. A single result is extracted from this composite by the fz-centrioid function, which returns the index of the median membership.

```
? test3
(0 0 0 0.2 0.4 0.6 1.0 0.6 0.4 0.2 0.0)
? (fz-centroid  test3)
6.0
? test4
(0 0.2 0.4 0.6 1.0 0.6 0.4 0.2 0.0 0 0)
? (fz-union  test3  test4)
(0 0.2 0.4 0.6 1.0 0.6 1.0 0.6 0.4 0.2 0)
? (fz-centroid  test4)
4.0
? (fz-centroid  (fz-union  test3  test4))
4.999999999999999
```

Building sets

We often need to build sets with arbitrary membership functions in them. Here are some lisp functions to construct sets.

```
(defun MAKE-FLAT (howmany value)
"Returns a list with all members set to value"

? (make-flat  12  0.5)
(0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5)

(defun MAKE-RAMP (howmany increment &optional (index 0))
"Returns a list of howmany values increasing by increment"

? (make-ramp  12  1/12)
(0.0 0.083 0.16 0.25 0.33 0.416 0.5 0.583 0.66 0.75 0.833 0.9166)

(defun FZ-MAKE-LINEAR-UP (howmany lastzero firstone)
"fuzzy set of arg1 elements ramping from 0 at lastzero to 1 at firstone "

? (fz-make-linear-up  12  4  7)
(0.0 0.0 0.0 0.0 0.0 0.33 0.66 1.0 1.0 1.0 1.0 1.0)

(defun FZ-MAKE-LINEAR-DOWN (howmany lastone firstzero)
"fuzzy set of howmany elements ramping from 1 at lastone to 0 at firstzero"
```

```
? (fz-make-linear-down  12  4  7)
(1.0 1.0 1.0 1.0 1.0 0.66 0.33 0.0 0.0 0.0 0.0 0.0)
```

(defun **FZ-MAKE-TRAPEZOID** (*howmany lastzero firstone lastone firstzero*)
"fuzzy set of *howmany* elements ramping from 0 at *lastzero* to 1 at *firstone* and back at *lastone* to *firstzero*"

```
? (fz-make-trapezoid  12  2  5  7  10)
(0.0 0.0 0.0 0.33 0.66 1.0 1.0 1.0 0.66 0.33 0.0 0.0)
```

(defun **FZ-MAKE-NUMBER** (*howmany number width*)
"fuzzy set of *howmany* elements triangular with point at *number* and width of *width*"

```
? (fz-make-number  12  5  3)
(0.0 0.0 0.0 0.0 0.0 0.66 1.0 0.66 0.0 0.0 0.0 0.0 0.0 0.0)
```