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 sample's 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 limited 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))
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)