(somewhat) Quick introduction to FuzzyCLIPS

(abridged version of the FuzzyCLIPS User's Guide by Bob Orchard, National Research Council Canada)

Fuzziness

Fuzziness occurs when the boundary of a piece of information is not clear-cut. For example, concepts such as *young*, *tall*, *good,* or *high* are fuzzy. There is no single quantitative value which defines the term *young*. For some people, age 25 is *young*, and for others, age 35 is *young*. In fact the concept *young* has no clean boundary. Age 1 is definitely *young* and age 100 is definitely not *young*; however, age 35 has some possibility of being *young* as well as some possibility of being *not young* and usually depends on the context in which it is being considered. The representation of this kind of information in FuzzyCLIPS is based on the concept of fuzzy set theory. Unlike classical set theory where one deals with objects whose membership to a set can be clearly described, in fuzzy set theory membership of an element to a set can be partial, i.e., an element belongs to a set with a certain grade (possibility) of membership. More formally a fuzzy set A in a universe of discourse U is characterized by a membership function

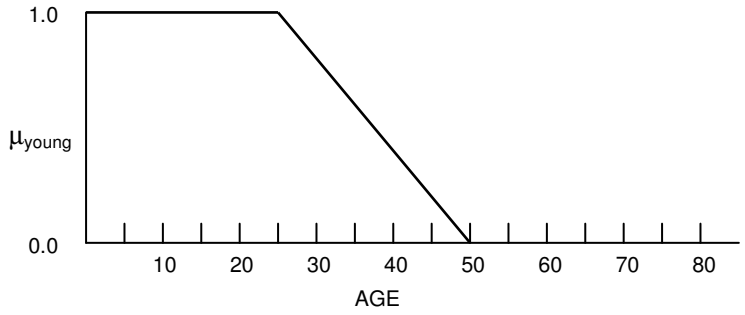
$$\mu_A : U \rightarrow [0,1] \qquad (1)$$

which associates a number $\mu_A(x)$ in the interval [0,1] with each element x of U. This number represents the grade of membership of x in the fuzzy set A. For example, the fuzzy term *young* might be defined as:

Fuzzy Term *young*

Age	Grade of Membership
25	1.0
30	0.8
35	0.6
40	0.4
45	0.2
50	0.0

Regarding equation (1), one can write: $\mu_{young}(25) = 1$, $\mu_{young}(30) = 0.8$, ... , $\mu_{young}(50) = 0$. Grade of membership values constitute a possibility distribution of the term *young*. The table can also be shown graphically:



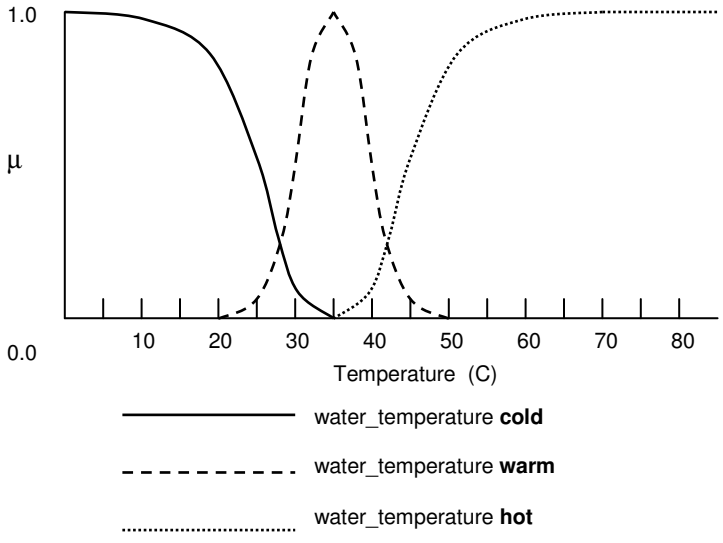
Possibility distribution of *young*

Fuzzy facts may be defined, matched as a pattern in a rule, or asserted

```
(deftemplate age ;definition of fuzzy variable 'age'
   0 120 years
  ( (young (25 1) (50 0))
    (old   (50 0) (65 1))
  )
)

(deffacts fuzzy-fact
   (age young)                    ; a fuzzy fact
)

(defrule one ; a rule that matches and asserts fuzzy facts
     (Speed_error big)
 =>
     (assert (Throttle_change small))
)
```

where young, *big* and *small* are fuzzy terms, and *age, Speed_error* and *Throttle_change* are linguistic (fuzzy) variables.

Each linguistic (fuzzy) variable has an associated fuzzy term set (called *primary terms* in FuzzyCLIPS) that is the set of values that the fuzzy variable may take. For example, the fuzzy variable *water_temperature* might have the primary term set {*cold, warm, hot*}, where each primary term represents a specific fuzzy set. The following figure illustrates the primary term values of the fuzzy variable *water_temperature*.



Primary terms of a *linguistic variable*

Uncertainty

Uncertainty occurs when one is not absolutely certain about a piece of information. The degree of uncertainty is usually represented by a crisp numerical value on a scale from 0 to 1, where a certainty factor of 1 indicates that the expert system is very certain that a fact is true, and a certainty factor of 0 indicates that it is very uncertain that a fact is true.

In FuzzyCLIPS, a fact is composed of two parts: the fact and its certainty factor. In general a FuzzyCLIPS fact takes the following form:

(fact) [CF certainty factor]

The CF acts as the delimiter between the fact and the certainty factor and [ ] indicates an optional part. For example,

```
(prediction sunny) CF 0.8
```

is a fact that indicates that the weather will be sunny with a certainty of 80%. However, if the certainty factor is omitted, as in a normal fact,

```
(prediction sunny)
```

then FuzzyCLIPS assumes that the weather will be sunny with a certainty of 100%. Certainty factors may also be associated with entire rules:

```
(defrule flight-rule
     (declare (CF 0.95)) ;declares certainty factor of the rule
     (animal type bird)
 =>
     (assert (animal can fly))
)
```

represents the hypothesis that, 95% of the time, if an animal is a bird, then it can fly. Similar to facts, if the certainty factor of a rule is not declared, it is assumed to be equal to the value 1.0.

Uncertainty and fuzziness can occur simultaneously:

```
(deffacts FuzzyAndUncertainFact
     (Speed_error more_or_less zero) CF 0.9
)

(defrule Uncertain_rule
     (declare (CF 0.8) )
     (Johns_age young)
 =>
     (assert (John goes to school))
)
```

where Speed_error and Johns_age are fuzzy variables, zero and young are fuzzy terms, more_or_less is a fuzzy term modifier and 0.9 and 0.8 are certainty factors associated with a fact and a rule respectively.

Inference Techniques

Three types of simple rules are defined: CRISP_, FUZZY_CRISP, and FUZZY_FUZZY. The naming convention is easy: the part before the underscore refers to the antecedent and the part after the underscore refers to the consequent.

CRISP_ Simple Rule

If the antecedent of the rule does not contain a fuzzy object, then the type of rule is CRISP_ regardless of whether or not a consequent contains a fuzzy fact.

Given a rule:

```
(defrule crisp-simple-rule
     (declare (CF 0.7))                  ;crisp rule certainty factor of 0.7
     (light_switch off)                  ;crisp antecedent
 =>
     (assert (illumination_level dark))  ; fuzzy consequent
)                                        ; end of rule definition
```

and given that the fact:

```
(light_switch off) CF 0.8
```

has been asserted, then the following fact will be asserted into the fact database due to the firing of the crisp-simple-rule:

```
(illumination_level dark) CF 0.56
```

where the certainty factor of the conclusion has been calculated as follows

$$CF_c = 0.7 * 0.8$$

FUZZY_CRISP Simple Rule

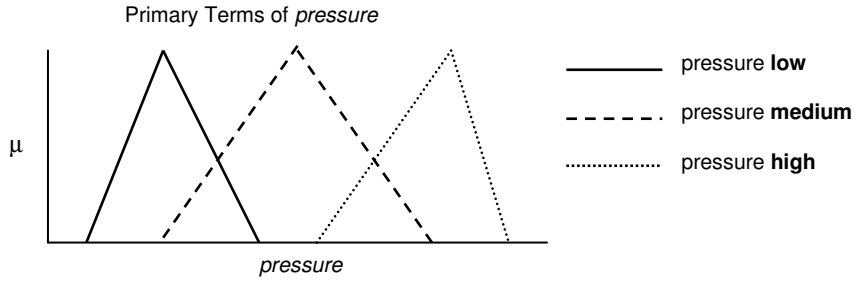
If only the antecedent contains a fuzzy fact, then the type of rule is FUZZY_CRISP.

Let us introduce some notation:

if A then C
A'

C'

For FUZZY_CRISP rules, A′ must be a fuzzy fact with the same fuzzy variable as specified in A for a match to occur and the rule to be placed on the agenda. In addition, while values of the fuzzy variables A and A′ represented by the fuzzy sets F and F′ do not have to be equal, they must overlap. For example, the fuzzy facts (*temperature high*) and (*pressure high*) do not match because the fuzzy variables *temperature* and *pressure* are not the same. However, given the fuzzy facts (*pressure low*), (*pressure medium*), and (*pressure high)*, as illustrated in the following figure, clearly (*pressure low*) and (*pressure medium*) overlap and thus match, while (*pressure low*) and (*pressure high*) do not match.



Matching of fuzzy facts

For a FUZZY_CRISP rule, the conclusion C′ is equal to C.

The CF of the consequent is

$$CF_c = CF_r * CF_f * S$$

where S is a measure of similarity between the fuzzy sets $F_{\square}$ (determined by the fuzzy pattern A) and $F'_{\square}$ of the matching fact A′). If the similarity between the fuzzy sets associated with the fuzzy pattern (A) and the matching fact (A′) is high the certainty factor of the conclusion is very close to $CF_r * CF_f$ since S will be close to 1. If the fuzzy sets are identical then S will be 1 and the certainty factor of the conclusion will equal $CF_r * CF_f$. If the match is poor then this is reflected in a lower certainty factor for the conclusion. Note also that if the fuzzy sets do not overlap then the similarity measure would be zero and the certainty factor of the conclusion would be zero as well. In this case the conclusion should not be asserted and the match would be considered to have failed and the rule would not be placed on the agenda.

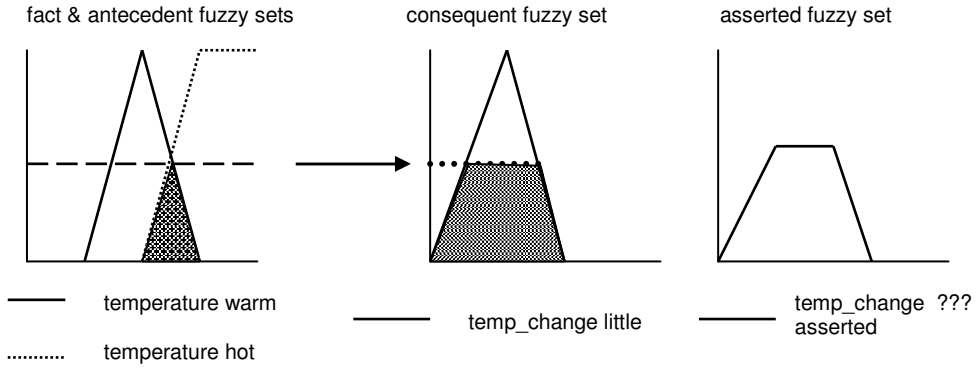
```
(defrule simple-fuzzy-crisp-rule
     (declare (CF 0.7))           ;rule has a certainty factor of 0.7
     (fuzzy-fact fact2)           ;fuzzy antecedent
 =>
     (assert (crisp-fact fact3))  ;crisp consequent
)
```

FUZZY_FUZZY Simple Rule

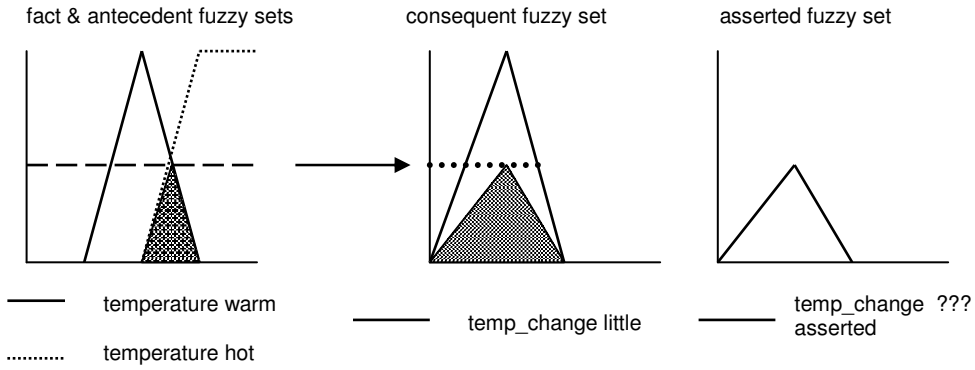
If both antecedent and consequent contain fuzzy facts, then the type of rule is FUZZY_FUZZY.

```
(defrule fuzzy-fuzzy-rule
;both antecedent and consequent are fuzzy objects
    (temperature hot)
 =>
    (assert (temp_change little))
)

(temperature warm)                ; a fact in the fact database
```

A graphical illustration of the matching of the fuzzy fact with the fuzzy pattern and the generation of the fuzzy conclusion is shown below. Note that this type of inference method is commonly referred to as **max-min** rule of inference. The conclusion set is simply *clipped* off at the z value. The subsequent figure shows the same results using a **max-prod** rule of inference. In this case the conclusion has all of its membership values scaled by the z value. The FuzzyCLIPS function set-inference-type allows the control of which method is used. The certainty factor of the conclusion is $CF_c = CF_r * CF_f$.



Compositional rule of inference (max-min)



Compositional rule of inference (max-prod)

Complex Rules

Multiple Consequents

In CLIPS, the consequent part of the rule can only contain multiple patterns (C_1, C_2 ,..., C_n) with the implicit **and** conjunction between them. They are treated as multiple rules with a single consequent. So the following rule:

if Antecedents then C_1 and C_2 and ... and C_n

is equivalent to the following rules:

if Antecedents then C_1
if Antecedents then C_2
...
if Antecedents then C_n

Multiple Antecedents

The problem of multiple patterns in the antecedent with a single assertion in the consequent needs to be considered. If the consequent assertion is not a fuzzy fact, no special treatment is needed since the conclusion will be the crisp (non-fuzzy) fact. However, if the consequent assertion is a fuzzy fact, the fuzzy value is calculated using the following basic algorithm:

If logical **and** is used, one has

if A_1 **and** A_2 then C	CF_r
A'_1	CF_{f1}
A'_2	CF_{f2}

C′	CF_c

where A'_1 and A'_2 are facts (crisp or fuzzy) that match the antecedents A_1 and A_2 respectively. In this case the fuzzy set describing the value of the fuzzy assertion in the conclusion is calculated according to the formula

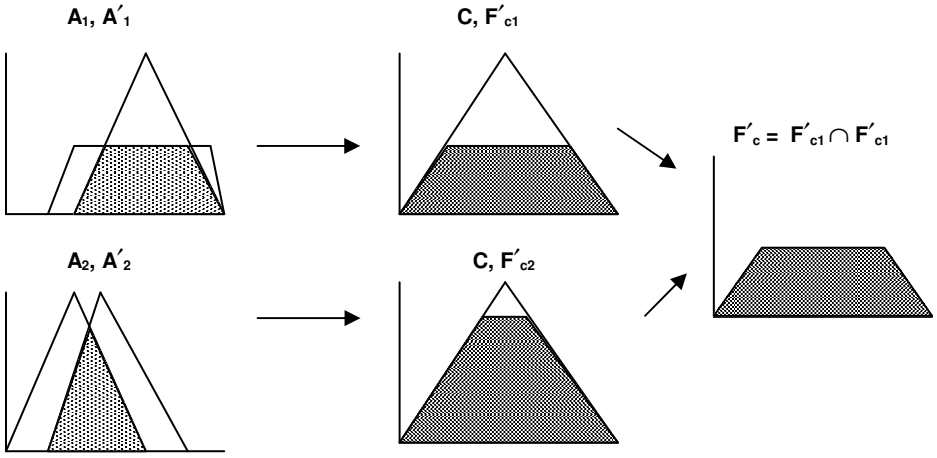
$$F'_c = F'_{c1} \cap F'_{c2}$$

where

$\cap$ denotes the intersection of two fuzzy sets
F'_{c1} is the result of fuzzy inference for the fact $A_{1'}$ and the simple rule
if A_1 then C
F'_{c2} is the result of fuzzy inference for the fact $A_{2'}$ and the simple rule
if A_2 then C

In the following figure we see the results of a rule in which both A_1 and A_2 are fuzzy patterns. Note that if both A_1 and A_2 were crisp (non-fuzzy) facts then the conclusion would just be the fuzzy fact C since we would be dealing with two CRISP_FUZZY simple rules. If one of the patterns is crisp (say A_1) and the

other is fuzzy then the conclusion is F'_{c2} since the CRISP_FUZZY simple rule would conclude C and the FUZZY_FUZZY simple rule would conclude F'_{c2}. The intersection of these two would just be F'_{c2}.



Compositional rule for multiple antecedents

The certainty factor of the conclusion is calculated according to MYCIN's model

$$CF_c = \min(CF'_{f1}, CF'_{f2}) * CF_r$$

where **min** denotes the minimum of the two numbers and

CF'_{f1} is the CF of the simple rule if A_1 then C given the matching fact A'_1
CF'_{f2} is the CF of the simple rule if A_2 then C given the matching fact A'_2

The above algorithm can be applied repeatedly to handle any combination of antecedent patterns, i.e.,

$$F'_c = F'_{c1} \cap F'_{c2} \ldots \cap F'_{cn}$$
$$CF_c = \min(CF'_{f1}, CF'_{f2}, \ldots, CF'_{fn}) * CF_r$$

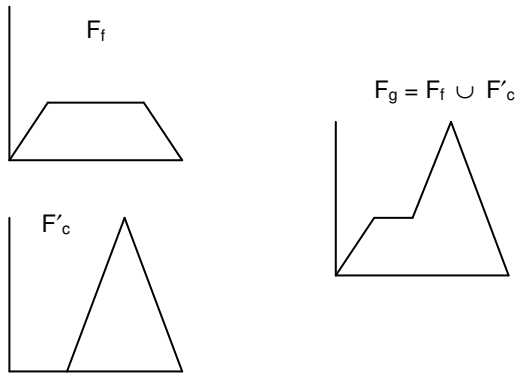
Global Contribution

In a crisp system a fact is asserted with specific values for its fields. If the fact already exists then the behavior is as if the fact was not asserted (unless fact duplication is allowed). In a crisp system there is never any need to re-assess the facts in the system - once they exist, they exist. But in a fuzzy system, refinement of a fuzzy fact may be possible. If a fuzzy fact is asserted, this fact is treated as giving contributing evidence towards the conclusion about the fuzzy variable (it contributes globally). If information about that fuzzy variable has already been asserted then this new evidence (or information) about the fuzzy variable is combined with the existing information in the fuzzy fact. In the current version of FuzzyCLIPS the new value of the fuzzy fact is calculated in accordance with the formula

$$F_g = F_f \cup F'_c$$

where

F_g is the new value of the fuzzy fact
F_f is the existing value of the fuzzy fact
F'_c is the value of the fuzzy fact to be asserted
$\cup$ denotes the union of two fuzzy sets



Union of fuzzy sets - global contribution

The uncertainties are also aggregated to form an overall uncertainty. Basically, two uncertainties are combined, using the formula $CF_g = \text{maximum}(CF_f , CF_c)$ where CF_g is the combined uncertainty, CF_f is the uncertainty of the existing fact, and CF_c is the uncertainty of the asserted fact.

As an example of the importance of the global contribution to a fuzzy fact, consider the implementation of a fuzzy logic controller. In this case the user has to ensure the firing of all rules that contribute to the control action to be performed, before any other rule (usually defuzzification) fires. This can be done by attaching a suitable salience to the rules or by separating the rules that all contribute to the same control action into a separate MODULE.

Defuzzification

The outcome of the fuzzy inference process is a fuzzy set, specifying a fuzzy distribution of a conclusion. However, in some cases, such as control applications, only a single discrete action may be applied, so a single point that reflects the best value of the set needs to be selected. This process of reducing a fuzzy set to a single point is known as *defuzzification*.

There are several possible methods, each one of which has advantages and disadvantages. A method which has been widely adopted is to take the center of gravity COG or moment) of the whole set. This has the advantage of producing smoothly varying controller output, but it is sometimes criticized as giving insufficient weight to rule consequents that agree and ought to reinforce each other. Another method concentrates on the values where the possibility distribution reaches a maximum, called the mean of maxima method. The mean of maxima (MOM) algorithm is criticized as producing less smooth controller output, but has the advantage of greater speed due to fewer floating-point calculations.

In FuzzyCLIPS, the user has the option of choosing either the COG or MOM algorithm when defuzzifying a fuzzy set.

Centre of Gravity Algorithm

The centre of gravity method may be written formally as

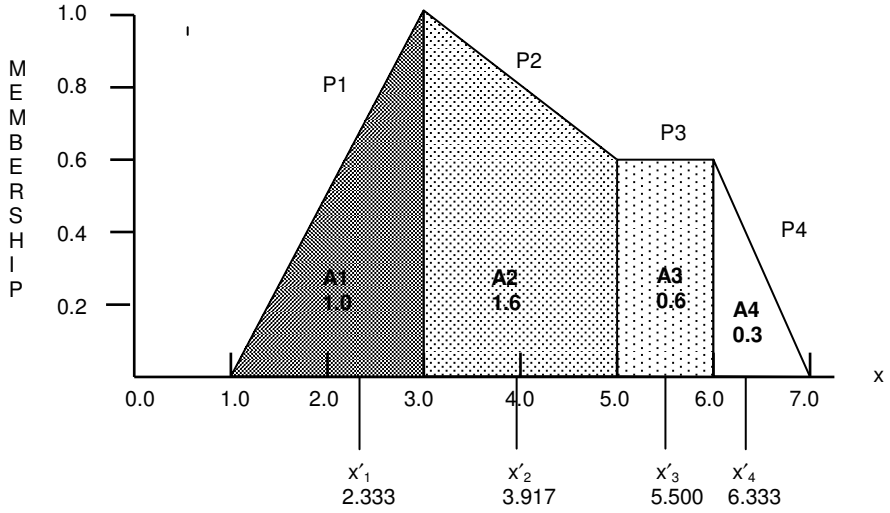
$$x' = \frac{\int_{x \in U} (x \cdot f(x)) dx}{\int_{x \in U} f(x) dx}$$

where x' is the recommended, defuzzified value, and the universe of discourse is U.

In FuzzyCLIPS, a fuzzy set is defined by a set of points that are considered to be connected by straight-line segments. The integral then reduces to a simple summation,

$$x' = \frac{\sum_{i=1}^{n} x_i' \cdot A_i}{\sum_{i=1}^{n} A_i}$$

where x_i' is the local centre of gravity, A_i is the local area of the shape underneath line segment (p_{i-1}, p_i), and n is the total number of points.



Example of COG defuzzification

For each shaded subsection in the figure above, the area and centre of gravity is calculated according to the shape identified (i.e., triangle, rectangle or trapezoid). The centre of gravity of the whole set is then determined:

$$x' = \frac{2.333 \cdot 1.0 + 3.917 \cdot 1.6 + 5.5 \cdot 0.6 + 6.333 \cdot 0.3}{1.0 + 1.6 + 0.6 + 0.3} = 3.943$$

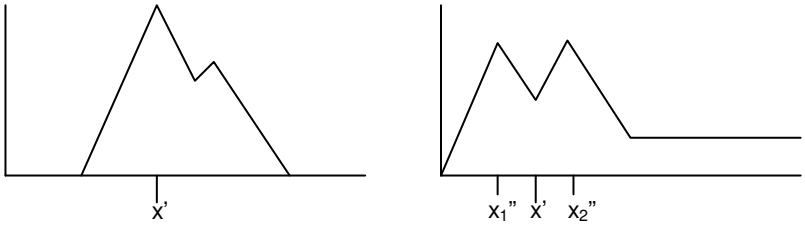
Mean of Maxima Algorithm

The MOM algorithm returns the x-coordinate (x'') of the point at which the maximum membership (y) value of the set is reached. For the fuzzy set illustrated in the previous figure, the MOM result would be 3.0.

If the maximum y value is reached at more than one point, then the average of all the x'' is taken. More formally:

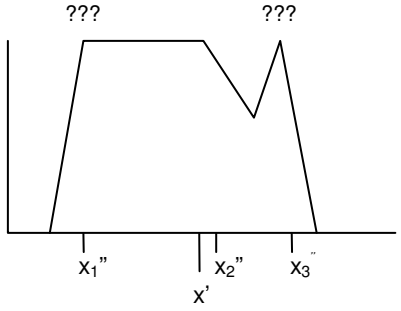
$$x' = \sum_{j=1}^{J} \frac{x''_j}{J}$$

where x''_j are the x-coordinates of all the maxima, and J is the total number of maxima.



Examples of MOM defuzzification

Note that an ambiguity occurs when the maximum value occurs along a plateau rather than just a series of individual peaks. In this case there are an infinite number of maximum points between x_1'' and x_2'' and using the average of the three points x_1'', x_2'' and x_3'' results in what may be an incorrect or perhaps an unexpected value. It is not entirely clear what the answer should be.



MOM example - Ambiguity

Defining Fuzzy Variables in Deftemplate Constructs

All fuzzy variables must be predefined before use with the deftemplate statement. The extended syntax of this construct is as follows:

```
(deftemplate <name> ["<comments>"]
   <from> <to> [<unit>]   ; universe of discourse
       (
          t1
          .
          .   ; list of primary terms
          .
          tn
       )
)
```

<name> is the identifier used for the fuzzy variable. The description of the universe of discourse consists of three elements. The <from> and <to> should be floating point numbers. They represent the beginning and end of the interval that describes the domain of the fuzzy variable (the universe of discourse). The value of <from> must be less than the value for <to>. The <unit> is a word that represents the unit of measurement (optional). The t_i are specifications for the fuzzy terms (such as hot, cold, warm) used to describe the fuzzy variable. These specifications describe the shape of the fuzzy set associated with the terms.

A primary term t_i (i=1, ..., n) has the form

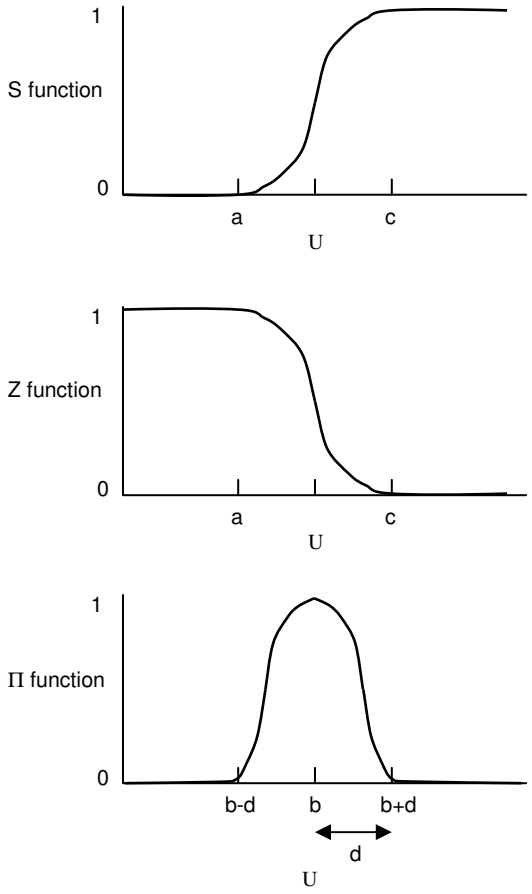
(<name> <description of fuzzy set>)

where <name> represents the name of a primary term used to describe a fuzzy concept, and <description of fuzzy set> defines a membership function of the given primary term. The membership function can be described using either:

- a singleton representation (of the form (point value), for example (3 0.5))
- a standard function representation (see next section)
- a linguistic expression that uses terms defined previously in the fuzzy deftemplate definition.

Standard Function Representation

Frequently, it is useful to describe a membership function using one of a set of standard functions S, Π, or Z. Parameters of these functions can be chosen, depending on applications. These functions look like this (note that the names are suggestive of the shape of the functions):



A standard representation of a membership function has the following format:

(S a c) | (s a c) | (Z a c) | (z a c) | (PI d b) | (pi d b)

where : a, b, c, d are numbers that represent the parameters of the respective functions.

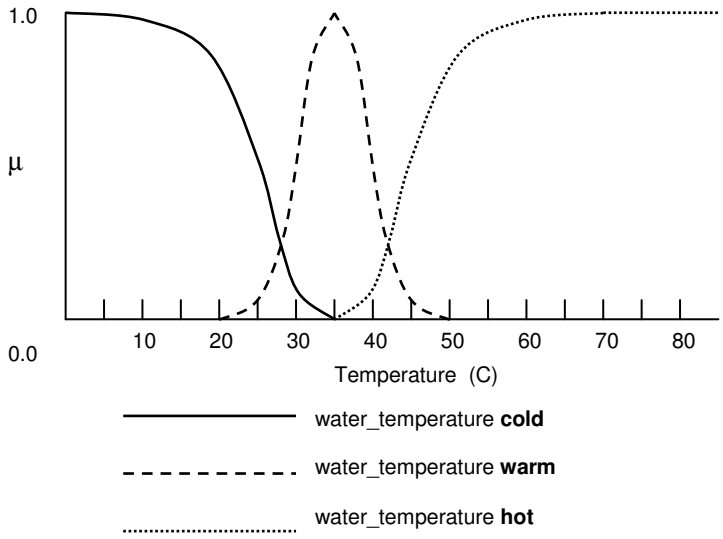
```
(deftemplate Tx "output water temperature"
     5 65 Celsius
     (  (cold (z 10 26))     ;standard set representation
        (OK (PI 2 36))
        (hot (s 37 60))
     )
 )
```

Linguistic Expression Representation

A simple example to illustrate the usage:

```
(deftemplate temperature
   0 100 C
     (  (cold (z 10 26))           ;standard set representation
        (hot (s 37 60))            ;standard set representation
        (warm not [ hot or cold ]) ; linguistic expression
     )
 )
```

Note that the term *warm* is described as being *not hot or cold*. It uses the terms *hot* and *cold* previously defined in this deftemplate to describe the warm concept. Only terms described in this deftemplate (before the term definition being defined) can be used (along with any available modifiers and the *and* and *or* operations).



Deftemplate Definitions with Fuzzy Slots (fields)

A fuzzy deftemplate describes a fuzzy variable. One may use these deftemplates to describe fuzzy facts in patterns and assert commands. For example,

```
(defrule high-temp
     (temperature hot)
=>
    (assert (move-throttle negative-big)
    (printout t "The temperature is hot")
)
```

contains a fuzzy pattern based on the fuzzy deftemplate temperature. It also contains an assertion of a fuzzy fact based on the fuzzy deftemplate for move-throttle. Facts that have as their relation name the name of a fuzzy deftemplate will be referred to as **fuzzy deftemplate facts**. These facts in essence have a single slot that holds the fuzzy value described by the linguistic expression (or fuzzy set description). It is also possible to supplement standard CLIPS deftemplate facts by having one or more of the slots be a fuzzy value slot. In these cases the slot is associated with a fuzzy deftemplate description so that the appropriate universe of discourse and terms are known. These facts with fuzzy slots are known as fuzzy facts. A fuzzy slot has the form

(fuzzy-slot> ::= (slot <slotname> (type FUZZY-VALUE <fuzzy-deftemplate-name>)

where <slotname> is the name of the slot and <fuzzy-deftemplate-name> is the name of a previously defined fuzzy deftemplate. Note that no other options are allowed in the slot specification. For example the slot cannot have multiple possible types like INTEGER and FUZZY-VALUE and the options that determine other constraints such default values or cardinality are not allowed. Fuzzy slots can only hold fuzzy values and must have a value.

```
;; assume that the fuzzy deftemplates fz-height and
;; fz-weight have already been defined
(deftemplate person
     (slot name (type SYMBOL))
     (slot height (type FUZZY-VALUE fz-height))
     (slot weight (type FUZZY-VALUE fz-weight))
)

(defrule big-person
   (person (name ?n)
           (weight heavy)
           (height tall))
  =>
   (printout t ?n " is a big person" crlf)
)
```

The use of fuzzy slots has a big payoff in many situations. Given the last example and only fuzzy deftemplate facts it would be necessary to define a fuzzy deftemplate for each person's weight and each person's height as well as rules for each person. In effect a fuzzy deftemplate acts somewhat like a type definition for the language in which a fuzzy variable type is defined.

Modifiers (Hedges)

A modifier may be used to further enhance the ability to describe our fuzzy concepts. Modifiers (very, slightly, etc.) used in phrases such as

very hot or *slightly* cold

change (modify) the shape of a fuzzy set in a way that suits the meaning of the word used. These modifiers are commonly referred to as **hedges**. FuzzyCLIPS has a set of predefined modifiers that can be used at any time to describe fuzzy concepts when fuzzy terms are described in fuzzy deftemplates, fuzzy rule patterns are written, or fuzzy facts or fuzzy slots are asserted.

Modifier Name	**Modifier Description**
not	1-y
very	y**2
somewhat	y**0.333
more-or-less	y**0.5
extremely	y**3
intensify2	(y**2) if y in [0, 0.5] 1 - 2(1-y)**2 if y in (0.5, 1]
plus	y**1.25
norm	normalizes the fuzzy set so that the maximum value of the set is scaled to be 1.0 (y = y*1.0/max-value)
slightly	intensify (norm (plus A AND not very A))

These modifiers change the shape of a fuzzy set using mathematical operations on each point of the set. In the above table the variable y represents each membership value in the fuzzy set and A represents the entire fuzzy set (i.e., y**2 squares each membership value and very A applies the very modifier to the entire set). Note that when a modifier is used it can be used in upper or lower case (NOT or not) or even a mix of cases (NoT).

These predefined modifiers are available for use at all times in fuzzy deftemplate definitions, fuzzy patterns specifications, fuzzy slot assert specifications, and in the fuzzy-modify function. Some examples below illustrate the use of the modifiers.

```
(deftemplate temp
  0 100 C
  ( (cold (Z 20 40))
    (hot  (S 60 80))
    (freezing extremely cold)
  )
```

```
)

(defrule temp-rule
  (temp not hot and not cold)
=>
  (printout t "It's such a pleasant day" crlf)
)
```

Linguistic Expressions

The use of fuzzy primary terms and modifiers together with the binary operators *and* and *or* allow us to express the problem solutions in a more natural way. These expressions are called linguistic expressions. Expressions such as

```
temperature very hot or very cold
```

are examples of expressions that could be used to describe fuzzy variable.

Note that AND has higher precedence than OR and that modifiers are basically unary operators with the highest precedence. One can control the order of the expression evaluation through the use of brackets [and]. These brackets must be separated from other items by a space due to the nature of the token parser.

Therefore,

A or B and C or D

is the same as

A or [B and C] or D

LHS Patterns

A fuzzy LHS pattern is of the form

(fuzzy-variable-name <linguistic-expr>)
or
(fuzzy-variable-name ?)
or
(fuzzy-variable-name ?<var-name>)
or
(fuzzy-variable-name ?<var-name> & <linguistic-expr>)
or
(template-name <slot-description>$^{+}$)

where
+ indicates that there are one or more of the <slot-description> entries, at least one of these is a <fuzzy-slot-description>, and a <fuzzy-slot-description> is

(fuzzy-slot-name <linguistic-expr>)
or
(fuzzy-slot-name ?)
or
(fuzzy-slot-name ?<var-name>)
or
(fuzzy-slot-name ?<var-name> & <linguistic-expr>)

The <linguistic-expr> is a fuzzy set specified by a combination of primary terms, modifiers, and the logical operators NOT and OR. A fuzzy-variable-name is the name of any fuzzy deftemplate. A template-name is the name of any non-fuzzy deftemplate. A fuzzy-slot-name is the name of a slot declared to have type FUZZY-VALUE.

The examples below show some of the ways in which fuzzy patterns might appear on the left-hand side of rules.

```
(deftemplate group           ;declaration of fuzzy variable group
  0 20 members
     ((few (3 1) (6 0)) ;primary term few
      (many (4 0) (6 1));primary term many
     )
)

(defrule simple-LHS
     (group few)             ;a simple fuzzy LHS pattern
  ...
)
```

```
 (defrule more-complex-lhs
  ?f <- (group very few or very many)
 =>
  (printout t "We are at the extreme limits of the number of"
               (get-u-units ?f) " in our club" crlf)
)


(deftemplate height
  0 8 feet
 ((short (Z 3 4.5))
  (medium (pi 0.8 5))
  (tall (S 5.5 6))
 )
)

(deftemplate person
  (slot name)
  (slot ht (type FUZZY-VALUE height))
)

(defrule quite-complex-lhs
  (person (name ?n) (ht ?h & very tall))
 =>
  (printout t ?n " is very tall, with a height of about "
             (maximum-defuzzify ?h) " "
             (get-u-units ?h) crlf )
)
```

In the last example ?h will become the fuzzy value that matches the *very tall* specification. It can then be used as an argument to various functions that can process fuzzy values, such as maximum-deffuzzify or get-u-units.

Deffacts Constructs

The syntax of the deffacts construct has been extended to allow fuzzy facts to be declared. Both fuzzy and non-fuzzy (crisp) facts can be declared in the same deffacts construct. Also certainty factors for crisp or fuzzy facts may be specified. Fuzzy fact specifications in a deffacts construct have the following form:

(deffacts <deffacts-name> [<comment>]
    <RHS-pattern>*
)

For example:

```
(deffacts groupA "some fuzzy facts"
     (my_group (1 0) (5 1) (7 0))   ;singleton description
     (your_group (z 4 8))           ;standard description
     (their_group (s (+ 1 1) 4))
     (person (name ralph) (ht tall))
)

(deffacts groupB "some fuzzy facts with certainty factors"
     (this_group (1 0) (5 1) (7 0)) CF 0.35
     (that_group (pi 2 (+ 3 4))) CF (+ .2 .3)
)
```

Assert Statements

The syntax of the assert construct has been expanded to allow fuzzy facts as arguments, and for the certainty factor of a crisp or fuzzy fact to be specified. The assert command (with a single fact asserted in an assert function call) is of the form

(assert
(<crisp fact> | fuzzy-variable-name <description of fuzzy set> |
template-name <slot-description>$^{+}$)
[CF <certainty factor> | <certainty factor expression>]
)

Some examples will illustrate the forms allowed for asserting fuzzy facts and fuzzy deftemplate facts.

```
(assert (group few))
(assert (group (1 0) (5 1) (7 0)) )
(assert (group NOT [ very few OR many ] ))
(assert (group (z 4 8))  )
(assert (person (name john) (ht extremely tall)))
(assert (person (name dan) (ht (pi 0 5.6)))
(assert (temp (24 0) (25 1) (26 0)))


(defrule assert-rule-1
     (zmin ?minval)      ;variable ?minval must be numeric
     (zmax ?maxval)      ;variable ?maxval must be numeric
  =>
     (assert
          (group (z ?minval ?maxval))
     )                   ;fuzzy set description with variables
)

(defrule assert-rule-2   ;asserts standard set with functions
     (zmin ?minval)
     (zmax ?maxval)
  =>
     (assert (group (z ?minval (+ ?maxval 2))))
)
(defrule assert-rule-3   ;asserts singleton set with functions
     (x1 ?x1val)
     (x2 ?x2val)
  =>
     (assert
          (group (?x1val 0) (?x2val 1) ((+ ?x2val 1) 0.6)
                    ((sqr ?x2val) 0))
     )
)
```

Defuzzification

A crisp value may be extracted from a fuzzy set using either the centre of gravity or mean of maxima techniques. The syntax is as follows:

(moment-defuzzify ?<*fact-var*> | integer | <fuzzy-value>) ; COG algorithm
(maximum-defuzzify ?<*fact-var*> | integer | <fuzzy-value>) ; MOM algorithm

The argument may be fact variable (?<fact-var>), which normally is bound on the LHS of a rule as described in the CLIPS Reference Manual. It may be the integer value of a fact number (i.e., the index of the fact on the fact list). It may also be a <fuzzy-value > that can be obtained (among other ways) by matching in the pattern of a fuzzy deftemplate fact or a FUZZY-VALUE slot.

These functions return a floating point number which is the result of performing the defuzzification.

Suppose that fact-1 (a fuzzy deftemplate fact) on the fact list is

```
(temperature warm)
```

and that the COG method returns a value of 28, while the MOM method returns a value of 32.5.

When the following defuzzification command is issued at the CLIPS prompt level it will return the value 28.0 and display this value at the CLIPS prompt level

```
(moment-defuzzify 1)
```

When either of the following rules are executed it will print 'Temperature is 32.5'.

```
(defrule defuzzification-1
     ?f <- (temperature ?)                  ; ? used to assure match of the
                                            ; temperature fuzzy fact
  =>
     (bind ?t (maximum-defuzzify ?f))       ; get the value
     (printout t "Temperature is " ?t crlf) ; print 32.5
)

(defrule defuzzification-2
     (temperature ?fv)            ; ?fv used to hold the fuzzy value
                                  ; of the matching fuzzy fact
     =>
     (printout t "Temperature is " (maximum-defuzzify ?fv) crlf)

)
```

Certainty Factors of Rules

The <certainty factor> is a number in the range 0 to 1.

```
(deffacts initial-facts
     (fact1) CF 0.8          ;fact with crisp CF of 0.8
)

(defrule some-other-rule
     (declare (CF 0.7))      ;a rule with CF of 0.7
     (fact1)
  =>
     (printout t "Hello!")
)
```

FuzzyCLIPS Commands and Functions

This section lists some of the commands and functions that seemed interesting to me. A longer list of FuzzyCLIPS commands and functions can be found in the full text version of this manual.

create-fuzzy-value

Syntax: (create-fuzzy-value <fuzzy-deftemplate-name> <description of fuzzy set>)

Purpose: This function allows a fuzzy value to be created. A fuzzy value is a fuzzy set that is associated with a particular fuzzy deftemplate. The fuzzy deftemplate determines the universe of discourse for the fuzzy set and the terms that can be used to describe the fuzzy set. The first argument, <fuzzy-deftemplate-name>, is the name of a fuzzy deftemplate. The remaining parts describe the fuzzy set as is done for a fuzzy slot when a fuzzy fact is asserted. This can be a linguistic expression, a singleton specification, or a standard function expression.

```
(create-fuzzy-value temp cold)
(create-fuzzy-value temp very hot or very cold)
(create-fuzzy-value temp (pi 10 20))
(create-fuzzy-value temp (s ?x (+ ?x 10)))
(create-fuzzy-value temp (10 1) (20 0))

(defrule test
  =>
    (bind ?fv (create-fuzzy-value temp cold))
    (assert (temp ?fv)) ; NOTE use of variable here!
)
```

Plotting a Fuzzy Value (plot-fuzzy-value)

Syntax: (plot-fuzzy-value <logicalName> <plot-chars> <low-limit> <high-limit> <fuzzy-value>$^{+}$)

Purpose: This function is used to plot fuzzy sets. The arguments are:
<logicalName> is any open router to direct the output to (e.g. t for the standard output)

<plot-chars> specifies the characters to be used in plotting (e.g., * or "*+.")

- for each fuzzyvalue specified a corresponding character from the string or symbol is used as the plotting symbol -- if more fuzzyvalues than symbols are specified then the last symbol is used for the remaining plots

<low-limit> is a numeric value that specifies the lowest x value to be displayed in the plot OR if it is not a numeric value then it will default to the low limit of the universe of discourse

<high-limit> is a numeric value that specifies the highest x value to be displayed in the plot OR if it is not a numeric value then it will default to the high limit of the universe of discourse

<fuzzy-value> is one of three things

- an integer that identifies a fuzzy deftemplate fact (in this case the fuzzy value from the fact is extracted and used)

- a variable with a fuzzy deftemplate fact address (in this case the fuzzy value from the fact is extracted and used)
- a variable with a fuzzy value

The + identifies that one or more <fuzzy-value> arguments may be present.

The fuzzy deftemplate associated with ALL fuzzy values to be plotted on the same plot must be the same one. This is required since the x axis must have the same meaning. The <high-limit> and <low-limit> values allow a *window* of the universe of discourse to be displayed and provides for scaling the graph in the x axis.

```
(deftemplate temp
   0 100 C
  ((cold (z 20 70))
   (hot (s 30 80))
  )
)
(plot-fuzzy-value t * nil nil (create-fuzzy-value temp hot))
Fuzzy Value: temp
Linguistic Value: hot (*)
 1.00                                      *************
 0.95                                    **
 0.90                                  **
 0.85                                *
 0.80                               *
 0.75                              *
 0.70                             *
 0.65
 0.60                            *
 0.55                           *
 0.50
 0.45                         *
 0.40                        *
 0.35                       *
 0.30
 0.25                     *
 0.20                    *
 0.15                   *
 0.10                 **
 0.05               **
 0.00******************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
Universe of Discourse:  From   0.00  to  100.00
```

fuzzy-union

Syntax: (fuzzy-union <fuzzy-value> <fuzzy-value>)

Purpose: Returns a new fuzzy value that is the union of two other fuzzy values. Both arguments must be of type FUZZY-VALUE.

```
(deftemplate temp
   0 100 C
  ((cold (z 20 70))
   (hot (s 30 80))
  )
)
(fuzzy-union (create-fuzzy-value temp cold)
             (create-fuzzy-value temp hot))
cold or hot

(plot-fuzzy-value t ".+*" nil nil
      (create-fuzzy-value temp cold)
      (create-fuzzy-value temp hot)
      (fuzzy-union (create-fuzzy-value temp cold)
                   (create-fuzzy-value temp hot))
)
Fuzzy Value: temp
Linguistic Value: cold (.),  hot (+),  [ cold ] OR [ hot ] (*)
 1.00*************                          *************
 0.95             **                      **
 0.90               **                  **
 0.85                 *                *
 0.80                  *              *
 0.75                   *            *
 0.70                    *          *
 0.65
 0.60                     *        *
 0.55                      *      *
 0.50
 0.45                       *   *
 0.40                        * *
 0.35                         +
 0.30                         *
 0.25                        + .
 0.20                       +   .
 0.15                      +     .
 0.10                    ++       ..
 0.05                  ++           ..
 0.00++++++++++++++++++               ...................
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

fuzzy-intersection

Syntax: (fuzzy-intersection <fuzzy-value> <fuzzy-value>)

Purpose: Returns a new fuzzy value that is the intersection of two other fuzzy values. Both arguments must be of type FUZZY-VALUE.

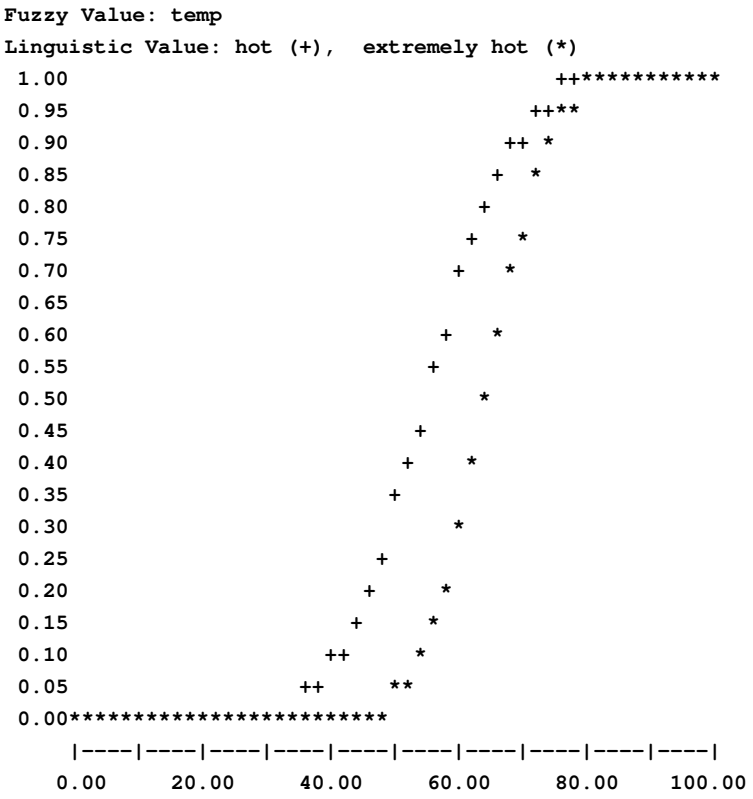
```
(fuzzy-intersection (create-fuzzy-value temp cold)
(create-fuzzy-value temp hot))
cold and hot

(plot-fuzzy-value t ".+*" nil nil
      (create-fuzzy-value temp cold)
      (create-fuzzy-value temp hot)
      (fuzzy-intersection (create-fuzzy-value temp cold)
                          (create-fuzzy-value temp hot))
)

Fuzzy Value: temp
Linguistic Value: cold (.),  hot (+),  [ cold ] AND [ hot ] (*)
 1.00.............                          ++++++++++++++
 0.95             ..                      ++
 0.90               ..                  ++
 0.85                 .                +
 0.80                  .             +
 0.75                   .           +
 0.70                    .         +
 0.65
 0.60                      .       +
 0.55                       .     +
 0.50
 0.45                         .   +
 0.40                          . +
 0.35                           +
 0.30                           *
 0.25                          * *
 0.20                         *   *
 0.15                        *     *
 0.10                      **       **
 0.05                    **           **
 0.00******************               ******************
     |----|----|----|----|----|----|----|----|----|----|
    0.00     20.00     40.00     60.00     80.00    100.00
```

fuzzy-modify

Syntax: (fuzzy-modify <fuzzy-value> <modifier>)

Purpose: Returns a new fuzzy value that is a modification of the fuzzy value argument. The modification performed is specified by the modifier argument. This modifier can be any active modifier (very, slightly, etc.).

```
(plot-fuzzy-value t "+*" nil nil
(create-fuzzy-value temp hot)
(fuzzy-modify (create-fuzzy-value temp hot) extremely))
```



(How to run a) Simple Example

```
CLIPS>(load "simplTst.clp")
Defining deftemplate: speed_error
Defining deftemplate: speed_change
Defining deffacts: my_facts
Defining defrule: speed-too-fast +j
Defining defrule: speed-ok +j
Defining defrule: get-crisp-value-and-print-rslt +j
TRUE
CLIPS> (reset)
==> f-0     (initial-fact) CF 1.00
==> f-1     (speed_error zero) CF 0.90   ;linguistic description of fuzzy set
       ( (0.0 1.0) (0.11 0.0)  )         ;singletons describe fuzzy set in detail
==> Activation 0      speed-ok: f-1
==> Activation 0      speed-too-fast: f-1
CLIPS> (run)
FIRE    1 speed-too-fast: f-1
==> f-2     (speed_change ???) CF 0.63   ;CF = 0.9 * 0.7 for fuzzy-fuzzy rule
       ( (0.1 0.0) (0.1495 0.0991)  )
==> Activation -1     get-crisp-value-and-print-rslt: f-2
FIRE    2 speed-ok: f-1
<== f-2     (speed_change ???) CF 0.63   ;retraction of fuzzy fact and ...
       ( (0.1 0.0) (0.1495 0.0991)  )
<== Activation -1     get-crisp-value-and-print-rslt: f-2
==> f-3     (speed_change ???) CF 0.63   ;reassertion as fact is modified
       ( (0.0 1.0) (0.1 0.1) (0.1333 0.06667) (0.1495 0.0991)  )
==> Activation -1     get-crisp-value-and-print-rslt: f-3
FIRE    3 get-crisp-value-and-print-rslt: f-3

Change speed by a factor of:  0.3553202565269306

3 rules fired        Run time is 0.06400000000212458 seconds.
46.87499999844391 rules per second.
3 mean number of facts (3 maximum).
1 mean number of instances (1 maximum).
1 mean number of activations (2 maximum).
CLIPS>
```

(A more detailed) Shower Example

The purpose of the shower example is to simulate the flow and temperature of water leaving a shower head as a function of time and to build a fuzzy controller to keep the flow and temperature within some required ranges.

Shower Model

1. The volumes of the pipes are zero.
2. Water mixes perfectly at point X (see Shower).
3. The water pipe is a perfect thermal insulator.
4. There is no heat transfer in the water as it travels from point X to the shower head.

As a consequence of the above model, the flow out of the shower head (Fx) at any time t is exactly equal to the flow of cold water (Fc) at time t plus the flow of hot water (Fh) at time t. Any change to a valve position (vc, vh) or to the water pressure (Ph, Pc) is immediately reflected in the flow from the shower head. The temperature of the water leaving the shower head (Tx) at time t is equal to the temperature of water at point X. See equations below.

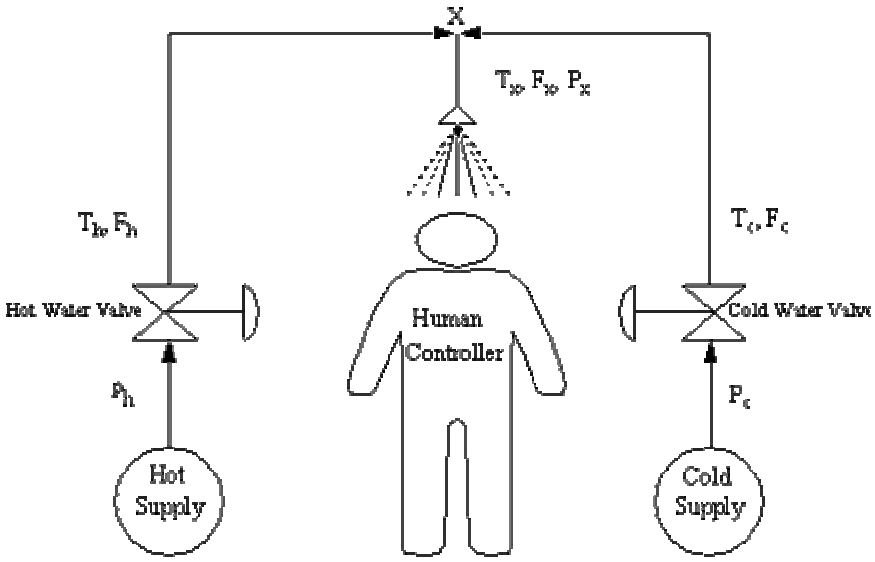
Shower Model Equations

$Tc \in [0, 35]$ °C
$Th > Tc$ $Th \in [0, 100]$ °C
$Tx = (FcTc + FhTh)/Fx$
$Fx = Fc + Fh$
$Px = 30$ kPa (atmospheric pressure)
$Fc = vc(Pc - Px)$
$Fh = vh(Ph - Px)$
$vc \in [0,1]$ - cold water valve position
$vh \in [0,1]$ - hot water valve position
Pc = f(neighbor watering lawn,clothes washing machine, etc.)
Ph = f(dish washing machine water consumption,etc.)

Changes of Pc and Ph are simulated.

Shower Control Objectives

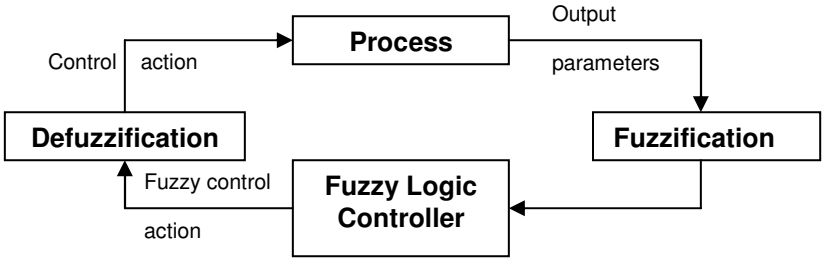
1. The temperature of the water leaving the shower head must “never” exceed 45°C. This is generally impossible.
2. The water temperature should rarely be less than 15°C.
3. The water temperature should have a mean value of 36°C and have a small variance.
4. The water flow should have a mean value of 12 L/min and have a small variance.
5. The hot and cold water valve actuators should be moved infrequently. (The control computer wants to spend more time washing than adjusting the taps).



Shower

Fuzzy Control Loop

Below we show a schematic of a control loop with a fuzzy logic controller.



Fuzzy Control Loop

Text Based Version (No Graphical Interface - shwrNOUI.clp)

Steps to follow to run fuzzy shower example

1. Start a version of FuzzyCLIPS.
2. Load the shower example (load "shwrNOUI.clp" from the fuzzy examples directory)
3. Run an example [(reset) and (run)] The program will ask for the values of parameters for temperature, pressure and valve positions. It stops when it reaches a flow between 11 and 13 L/min and water temperature between 34 and 38°C. The values of certain parameters will be printed after each set of fuzzy rules and defuzzification has taken place.
4. You will be prompted to enter further values or to quit.