Problems and Exercises

Lab 1

Blood Cell Classification case example

Experts are very good at classifying blood cells by looking at a blood sample through a microscope. But when they try to explain how they do the classification, they usually come up with a set of ill-defined, incomplete rules. A set of such rules is given as an illustration in figure 1. The rules, though, are not difficult to implement as a fuzzy system. The five features—nucleus exists, segmented or granular cell, darkness of the nucleus, size of the nucleus, and size of granules—are used as input variables in the realization described here. Fuzzy values are defined for the input fuzzy variables, for example: "yes," "maybe," "no." The output variables are the blood cell classes. Their fuzzy values are defined as certainty degrees for the object to be classified class "yes," "likely," "possibly," "maybe," and "not." Membership functions of singleton type for representing those fuzzy values are shown on the screenshot in figure 2. The three membership functions for representing the fuzzy variable nucleus-exists and the fuzzification result for a particular input data, as well as the classification result, are graphically and numerically presented there.

Create fuzzy variables and rules into MATLAB. Test sample.

RULE 1:
IF Nucleus might exist and cell might be segmented and granular and Nucleus is dark,
THEN the cell is unlikely to be an Erythrocyte and the cell is possibly a Lymphocyte.

RULE 2:
IF Nucleus might exist and cell might be segmented and granular and Nucleus is light,
THEN the cell is unlikely to be an Erythrocyte and the cell is unlikely to be a Lymphocyte.

RULE 3:
IF Nucleus might exist and cell is not segmented and granular and Nucleus is light,
THEN the cell may be an Erythrocyte and the cell is unlikely to be a Lymphocyte
And the cell is likely to be a Monocyte.

RULE 4:
IF Nucleus might exist and cell is not segmented and granular and Nucleus is dark,
THEN the cell is unlikely to be an Erythrocyte and the cell is unlikely to be a Monocyte
And the cell is likely to be a Lymphocyte.

RULE 5:
IF Nucleus exist and cell might be segmented and granular and Nucleus is light,
THEN the cell is likely to be a Monocyte.

RULE 6:
IF Nucleus exist and cell might be segmented and granular and Nucleus is dark,
THEN the cell is likely to be a Lymphocyte.

RULE 7:
IF Nucleus exist and cell is not segmented and granular and Nucleus is dark,
THEN the cell is a Lymphocyte.

RULE 8:
IF Nucleus exist and cell is not segmented and granular and Nucleus is light,
THEN the cell is a Monocyte.

Figure 1. A set of expert fuzzy rules for Blood Cell Classification Problem

RULE 9:
IF Nucleus is large and granules are large and Nucleus is medium dark,
THEN the cell is likely to be a Basophil.

RULE 10:
IF Nucleus is large and granules are large and Nucleus is light,
THEN the cell is a Basophil.

RULE 11:
IF Nucleus is large and granules are medium and Nucleus is light,
THEN the cell is likely to be a Basophil.

RULE 12:
IF Nucleus is large and granules are medium and Nucleus is medium dark,
THEN the cell is possibly a Basophil.

RULE 13:
IF Nucleus is medium and granules are large and Nucleus is medium dark,
THEN the cell is possibly a Basophil and the cell is possibly an Eosinophil.

RULE 14:
IF Nucleus is medium and granules are large and Nucleus is light,
THEN the cell is a Basophil.

RULE 15:
IF Nucleus is medium and granules are large and Nucleus is light,
THEN the cell is likely to be a Basophil.

RULE 16:
IF Nucleus is medium and granules are medium and Nucleus is dark,
THEN the cell is likely to be a Eosinophil.

RULE 17:
IF Nucleus is medium and granules are large and Nucleus is medium dark,
THEN the cell is likely to be a Basophil and the cell is possibly an Eosinophil.

RULE 18:
IF Nucleus is medium and granules are medium and Nucleus is dark,
THEN the cell is possibly an Eosinophil and the cell is may be a Neutrophil.

RULE 19:
IF Nucleus is small and granules are large and Nucleus is dark,
THEN the cell is an Eosinophil.

RULE 20:
IF Nucleus is small and granules are large and Nucleus is medium dark,
THEN the cell is likely to be an Eosinophil.

RULE 21:
IF Nucleus is small and granules are medium and Nucleus is medium dark,
THEN the cell is possibly an Eosinophil and the cell is likely to be a Neutrophil.

RULE 22:
IF Nucleus is small and granules are medium and Nucleus is medium dark,
THEN the cell is possibly an Eosinophil and the cell is may be a Neutrophil.

Figure 1. A set of expert fuzzy rules for Blood Cell Classification Problem (continued)

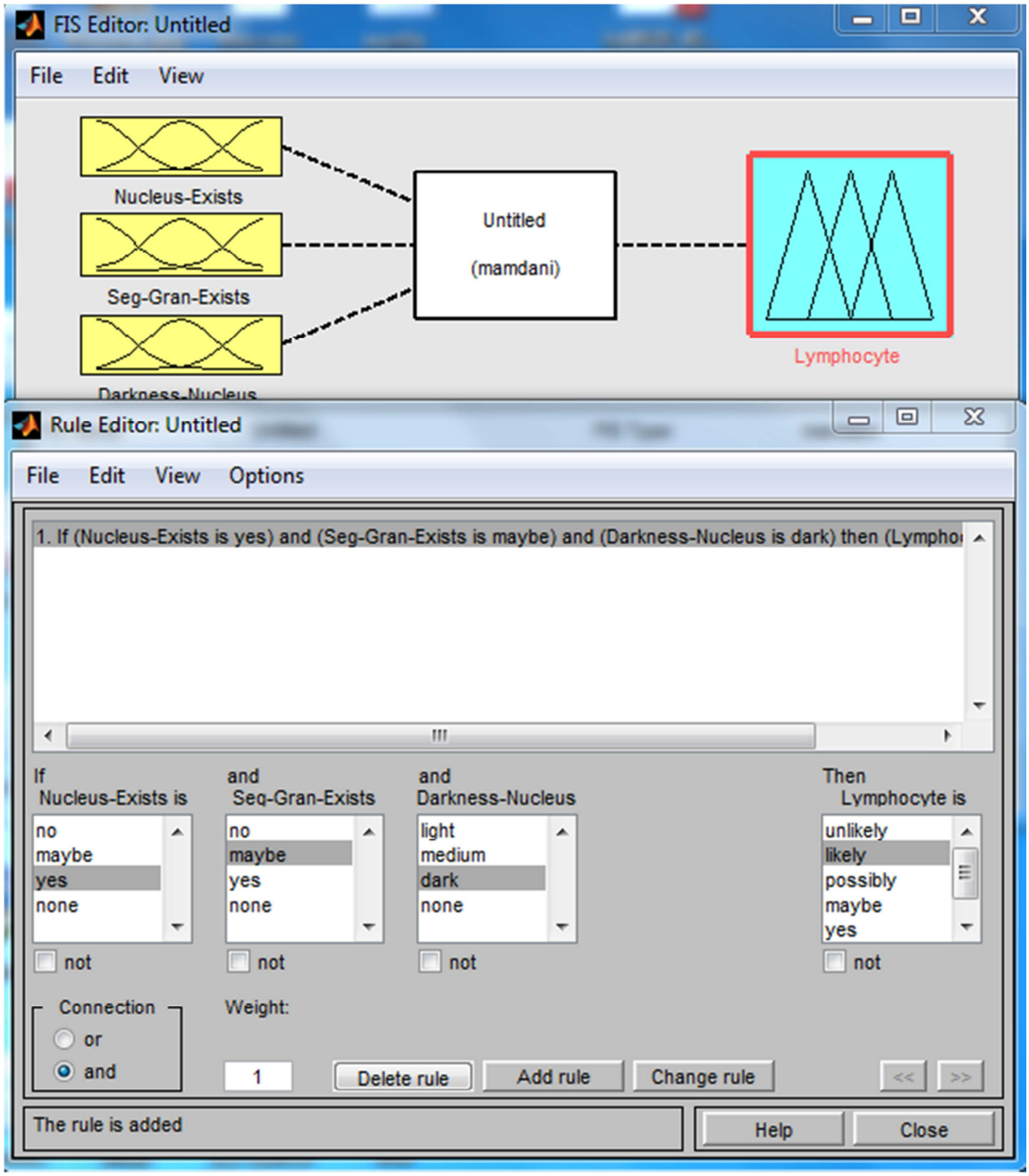
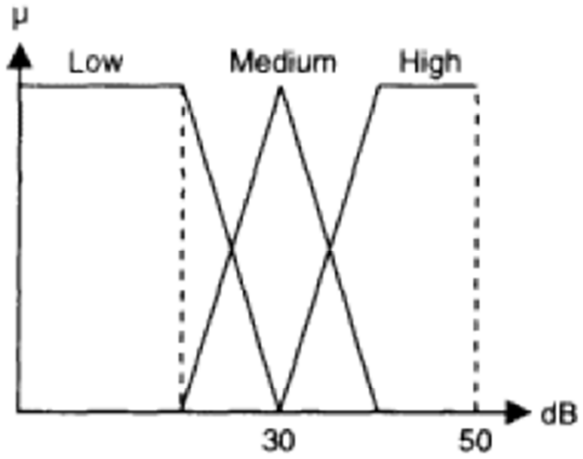


Figure 2. Fuzzy variables of the Blood Cells Classification problem, one output variable (Lymphocyte) and Rule Editor in MATLAB.

Lab 2
Recognition of musical notes

Instead of the intervals used there for representing variations of the energy of the signal in different frequency bands, fuzzy quantization can be applied and fuzzy rules can be articulated based on the exact interval rules (figure 3).
Create fuzzy variables and rules into MATLAB. Test sample.



(a)

IF 0-1000 is Medium AND
1000-2000 is Medium AND
2000-3000 is Low THEN
The note is Middle C

IF 0-1000 is High AND
1000-2000 is Medium AND
2000-3000 is Low THEN
The note is D above Middle C

IF 0-1000 is High AND
1000-2000 is Low AND
2000-3000 is Low THEN
The note is E above Middle C

IF 0-1000 is Medium AND
1000-2000 is Medium AND
2000-3000 is Medium THEN
The note is F above Middle C

IF 0-1000 is Low AND
1000-2000 is Medium AND
2000-3000 is Medium AND
3000-4000 is Low THEN
The note is G above Middle C

IF 0-1000 is Low AND
1000-2000 is Medium AND

2000-3000 is Medium AND
3000-4000 is Medium THEN
The note is A above Middle C

IF 0-1000 is Low AND
1000-2000 is Medium AND
2000-3000 is Low THEN
The note is B above Middle C

IF 0-1000 is High AND
1000-2000 is High AND
2000-3000 is High THEN
The note is C above Middle C

(b)

Figure 3. (a) Membership functions for the Musical Recognition Problem;
(b) some fuzzy rules for classification of a musical signal.

Labs 3-9

FuzzyCLIPS Commands and Functions

A number of commands supplied with CLIPS can be used to look at fuzzy facts and templates just as one looks at standard facts (see commands facts, ppdeftemplate, list-deftemplates, undeftemplate, ppdeffacts, list-deffacts, undeffacts). The following commands or functions add capability to access components of fuzzy facts, control thresholds for rule firings, set precision for fuzzy set display, set the fuzzy inference type, set a threshold for fuzzy pattern matching, create and manipulate fuzzy values, etc.

Lab 3

1.1.1 Accessing the Universe of Discourse (get-u, get-u-from, get-u-to, get-u-units)

Command: get-u

Syntax: (get-u ?<fact-var>) or

(get-u <integer>) or

(get-u <fuzzy-template-name>) or

(get-u <fuzzy-value>)

where

?<fact-var> is a fact variable, normally bound on the LHS of a rule
<integer> is the number of a fact on the fact list (constant or expression)
<fuzzy-template-name> is a symbol that represents the name of a fuzzy deftemplate
<fuzzy-value> is an element of type FUZZY-VALUE

Purpose: Returns a string of the form: "<from> - <to> <units>", the limits of the universe of discourse and the units (if specified). If no units have been declared in the deftemplate statement, then the function returns "<from> - <to>".

Example 35

```
(get-u ?t)        ;; ?t is bound to the temp fuzzy fact
(get-u 2)         ;; 2 is a fact index
(get-u temp)      ;; temp is the name of the fuzzy
deftemplate


(defrule test
  (temp ?fv)      ;; ?fv is a fuzzy value
 =>
  (printout t (get-u ?fv))
```

)

Command: get-u-from

Syntax: (get-u-from ?<fact-var>) or
(get-u-from <integer>) or
(get-u-from <fuzzy-template-name>) or
(get-u-from <fuzzy-value>)

Purpose: Returns the lower limit of the universe of discourse in floating point format.

Command: get-u-to

Syntax: (get-u-to ?<fact-var>) or
(get-u-to <integer>) or
(get-u-to <fuzzy-template-name>) or
(get-u-to <fuzzy-value>)

Purpose: Returns the upper limit of the universe of discourse in floating point format.

Example 36

```
;; Deffunction fuzzify
;;
;; Inputs:   ?fztemplate  - name of a fuzzy deftemplate
;;           ?value       - float value to be fuzzified
;;           ?delta       - precision of the value
;;
;; Asserts a fuzzy fact for the fuzzy deftemplate. The
fuzzy set
;; is a triangular shape centered on the value provided
with zero
;; possibility at value+delta and value-delta. Note that
it
;; checks bounds of the universe of discourse to
generate a fuzzy
;; set that does not have values outside of the universe
range.

(deffunction fuzzify (?fztemplate ?value ?delta)

  (bind ?low (get-u-from ?fztemplate))
```

```
  (bind ?hi  (get-u-to   ?fztemplate))

  (if (<= ?value ?low)
    then
      (assert-string
         (format nil "(%s (%g 1.0) (%g 0.0))"
                 ?fztemplate ?low ?delta))
    else
      (if (>= ?value ?hi)
        then
          (assert-string
             (format nil "(%s (%g 0.0) (%g 1.0))"
                     ?fztemplate (- ?hi ?delta) ?hi))
        else
          (assert-string
             (format nil "(%s (%g 0.0) (%g 1.0) (%g
0.0))"
                     ?fztemplate (max ?low (- ?value
?delta))
                     ?value (min ?hi (+ ?value ?delta))
))))
)
```

As an example the following uses the fuzzify function:

```
(deftemplate temp
   0 100 Degrees-F
   ( (warm (30 0) (60 1) (90 0))
   )
)

(defrule test
   (temp warm)
 =>
   (bind ?x (assert (dummy)))
   (printout t “Certainty Factor = “ (get-cf ?x) crlf)
   (retract ?x)
)
```

Asserting a fuzzified fact with

```
(fuzzify temp 50 0.001)
```

and firing the rule 'test' will result in the output:

Certainty Factor = 0.66667

Command: get-u-units

Syntax: (get-u-units ?<fact-var>) or
(get-u-units <integer>) or
(get-u-units <fuzzy-template-name>) or
(get-u-units <fuzzy-value>)

Purpose: Returns the units of the universe of discourse in string format. If no units have been specified, then the empty string " " is returned.

1.1.2 Accessing the Fuzzy Set (get-fs, get-fs-x, get-fs-y, get-fs-length, get-fs-lv, get-fs-value)

Command: get-fs

Syntax: (get-fs ?<fact-var>) or
(get-fs <integer>) or
(get-fs <fuzzy-value>)

Purpose: Returns the entire fuzzy set in singleton representation, in string format.

Command: get-fs-length

Syntax: (get-fs-length ?<fact-var>) or
(get-fs-length <integer>) or
(get-fs-length <fuzzy-value>)

Purpose: Returns the number of pairs in a fuzzy set description as an integer.

Command: get-fs-x

Syntax: (get-fs-x ?<fact-var> <i>) or
(get-fs-x <integer> <i>) or
(get-fs-x <fuzzy-value> <i>)

where <i> is an integer, variable, or function expression.

Purpose: Returns the x-coordinate of the ith pair in the fuzzy set, where the pairs are numbered left to right from 0 to (n-1) and n is the total number of pairs in the set. If the expression <*i*> evaluates to a non-integer value, then it is truncated to the nearest integer. The x-coordinate is returned as a floating point value.

Command: get-fs-y

Syntax: (get-fs-y ?<fact-var> <i>) or

(get-fs-y <integer> <i>) or

(get-fs-y <fuzzy-value> <i>)

Purpose: Returns the y-coordinate of the ith pair in the fuzzy set as a floating point value.

Example 37

Suppose the fuzzy fact

(temperature ???)

was fact-2 on the fact list (the ??? indicates that the fuzzy set
is not expressible using terms and modifiers for that fuzzy
variable; it may have been defined using a singleton description
or modified by the compositional rule of inference). Suppose
also that it has a fuzzy set consisting of the following three
singletons

((0 0) (25 1) (40 0)).

Then (get-fs-x 2 1) at the command line would return 25.
In a rule one could do the following:

```
(defrule print-last-coordinate
    ?f <- (temperature ?)
    =>
    (bind ?n (get-fs-length ?f))
    (bind ?x (get-fs-x ?f (- ?n 1)))
```

```
    (bind ?y (get-fs-y ?f (- ?n 1)))
    (printout t “The last point in the set is:” ?x “,”
?y crlf)
)

    OR

(defrule print-last-coordinate
    (temperature ?fv)
    =>
    (bind ?n (get-fs-length ?fv))
    (bind ?x (get-fs-x ?fv (- ?n 1)))
    (bind ?y (get-fs-y ?fv (- ?n 1)))
    (printout t “The last point in the set is:” ?x “,”
?y crlf)
)
```

Command: get-fs-lv

Syntax: (get-fs-lv ?<fact-var>) or
(get-fs-lv <integer>) or
(get-fs-lv <fuzzy-value>)

Purpose: Returns the returns the linguistic value associated with the fuzzy set.

Example 38

```
if (temp very hot) is asserted and the variable
?fuzzyfact is assigned to this fact then the function
call

    (get-fs-lv ?fuzzyfact)

would return the string "very hot"
```

Command: get-fs-value

Syntax: (get-fs-value ?<fact-var> <number>) *or*
(get-fs-value <integer> <number>) *or*
(get-fs-value <fuzzy-value> <number>)

Purpose: Returns the returns the value of the fuzzy set at the specified x value (<number>). The <number> is a value that must lie between the lower and upper limits of the universe of discourse for the fuzzy set.

Example 39

Suppose we have defined the following

```
(deftemplate temp
  0 100 C
  (
     ...
    (OK (30 0) (60 1) (90 0))
     ...
  )
)
```

and we assert the fact (temp OK). Then if we bind that fact to a variable ?fact we could call

```
(get-fs-value ?fact 50.0)
```

and it would return 0.6666667

1.1.3 Accessing the Certainty Factor (get-cf)

Command: get-cf

Syntax: (get-cf ?<fact-var>)

(get-cf <integer>) or

Purpose: Returns the certainty factor of a fact as a floating point number.

Example 40

Suppose a fact on the fact list is *(temperature cold) CF 0.7.*

```
(defrule print-cf-rule
   ?f <- (temperature ?)
 =>
  (printout t "The certainty of the temperature
measurement is: "
    (get-cf ?f))
)
```

Then the above rule will print "The certainty of the temperature measurement is: 0.7".

Lab 4

1.1.4 Enabling and Disabling Certainty Factor Calculations in Rules (enable-cf-rule-calculation, disable-cf-rule-calculation)

These commands control the certainty factors are calculated for facts asserted within rule executions. If the calculation is disabled the facts are assigned the certainty specified in the fact assertion (or 1.0 if not specified). If enabled then the calculations for certainty factors of facts described in sections **Error! Reference source not found.** and **Error! Reference source not found.** are applied.

Command: enable-cf-rule-calculation

Syntax: (enable-cf-rule-calculation)

Purpose: After executing this command any facts asserted in a rule will use the appropriate calculations to determine the certainty value for the fact.

Command: disable-cf-rule-calculation

Syntax: (disable-cf-rule-calculation)

Purpose: After executing this command any facts asserted in a rule will always have the certainty value specified for the fact and if none is specified it will have a value of 1.0. The certainty value of the rule or the matched facts will not be considered.

1.1.5 Accessing the Threshold Certainty Factor (threshold, get-threshold)

Command: set-threshold[1]

Syntax: (set-threshold <NUMBER>*)*

Purpose: Sets threshold certainty factor to the value of <NUMBER>. <NUMBER> must evaluate to a floating value between 0.0 and 1.0. By default the threshold value is 0.0.

Command: get-threshold

Syntax: (get-threshold)

Purpose: Returns the floating point value of the threshold certainty factor if threshold capability is ON. If it is OFF, then a value of 0.0 is returned.

---

1 For compatibility with previous versions of FuzzyCLIPS *threshold* is also accepted.

1.1.6 Setting the Rule CF Evaluation Behaviour (set-CF-evaluation, set-CF-evaluation)

Command: set-CF-evaluation

Syntax: (set-CF-evaluation <value>)

Purpose: Sets the behavior for evaluating the CF of rules to <value>. Value must be one of when-defined (default) or when-activated. This is similar to the set-salience-evaluation function of CLIPS. The value when-defined forces the certainty factor of the rule to be evaluated at the time of rule definition (compilation). The value when-activated forces the certainty factor of the rule to be defined at the time of rule definition and when the rule is activated (added to the agenda).

Command: get-CF-evaluation

Syntax: (get-CF-evaluation)

Purpose: Returns the current setting of the behavior for evaluating the CF of rules. Return value is either when-defined or when-activated (similar to the get-salience-evaluation function).

Lab 5

1.1.7 Controlling the Fuzzy Set Display Precision (set-fuzzy-display-precision, get-fuzzy-display-precision)

Command: set-fuzzy-display-precision

Syntax: (set-fuzzy-display-precision <integer>)

Purpose: When fuzzy facts are displayed the fuzzy set values are displayed in floating point format. This function allows the number of significant digits displayed after the decimal point to be set. The <integer> argument is an integer value between 2 and 16. If it is less than 2 it is set to 2 and if it is greater than 16 it is set to 16. The default value is 4. Note that *clear* will not reset this value to 4.

Example 41

```
(set-fuzzy-display-precision 16)
(facts)
   f-0  (speed_error more_or_less large_positive) CF
1.00
        ( (0.0 0.0)
          (0.1 0.3162277660168379) (0.2
0.4472135954999579)
```

```
          (0.3 0.5477225575051661) (0.35
0.5916079783099616)
          (0.4 0.6324555320336759) (0.5
0.7071067811865476)
          (0.6 0.7745966692414834) (0.7
0.8366600265340756)
          (0.8 0.8944271909999159) (0.9
0.9486832980505138)
          (1.0 1.0)  )

(set-fuzzy-display-precision 2)
(facts)
   f-0  (speed_error more_or_less large_positive) CF
1.00
        ( (0.0 0.0)
          (0.1 0.32) (0.2 0.45)
          (0.3 0.55) (0.35 0.59)
          (0.4 0.63) (0.5 0.71)
          (0.6 0.77) (0.7 0.84)
          (0.8 0.89) (0.9 0.95)
          (1.0 1.0)  )
```

Command: get-fuzzy-display-precision

Syntax: (get-fuzzy-display-precision)

Purpose: Returns an integer value that is the current display precision.

1.1.8 Controlling the Fuzzy Inference Method (set-fuzzy-inference-type, get-fuzzy-inference-type)

Command: set-fuzzy-inference-type

Syntax: (set-fuzzy-inference-type <inf-type>)

Purpose: Sets the current inference type to one of *max-min* or *max-prod*. The default is *max-min*. The effect of this is described in more detail in Section 4.3.1.3. Note that *clear* will not reset this value to *max-min*.

Command: get-fuzzy-inference-type

Syntax: (get-fuzzy-inference-type)

Purpose: Returns a symbol that is one of *max-min* or *max-prod* indicating the current inference type.

1.1.9 Setting the Fuzzy Pattern Matching Threshold (set-fuzzy-inference-type, get-fuzzy-inference-type)

Command: set-alpha-value

Syntax: (set-alpha-value <alpha-val>)

Purpose: When fuzzy slots are matched to fuzzy patterns on the LHS of rules, the match is considered to be successful if there is any overlap (intersection) between the two fuzzy sets involved in the matching. This function allows the match to succeed only if the maximum of the intersection set has a membership value greater than or equal to this threshold. The default alpha-value is 0.0. When the alpha-value is 0.0 the maximum of the intersection set must be greater than 0.0. Note that a *clear* does not reset the alpha-value to 0.0.

Example 42

```
(deftemplate temp
  0 100 C
  ( (low (10 1) (50 0))
    (ok  (20 0) (50 1) (80 0))
    (high (50 0) (90 1))
  )
)
(defrule test-alpha
   (temp low)
  =>
   (printout t "Rule fired ****" crlf)
)

(set-alpha-value 0.0)
(assert (temp (pi 0 30)))
(run) ; rule should fire with alpha 0.0
Rule fired ****
(retract *)
(set-alpha-value 0.5)
(assert (temp (pi 0 30)))
(run) ; rule should fire with alpha 0.5
Rule fired ****
(retract *)
(set-alpha-value 0.55)
(assert (temp (pi 0 30)))
(run) ; rule should NOT fire with alpha 0.55
;; no output here -- match was not successful
```

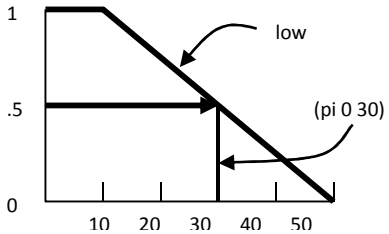


Figure 1

Command: get-alpha-value

Syntax: (get-alpha-value)

Purpose: Returns a floating point value which is the current alpha-value.

Lab 6

1.1.10 Fuzzy Value Predicate Function (fuzzvaluep)

Command: fuzzyvaluep

Syntax: (fuzzyvaluep <arg>)

Purpose: This function returns TRUE if the argument is of type FUZZY-VALUE, otherwise it will return FALSE.

Example 43

```
(fuzzyvaluep 45.6)
FALSE
(fuzzyvaluep “string”)
FALSE
(fuzzyvaluep (create-fuzzy-value temp cold))
TRUE

(defrule check-fuzzyvaluep
  (temp ?fv & cold)
 =>
  (fuzzyvaluep ?fv)
)
(assert (temp cold))
(run)
TRUE
```

1.1.11 Creating and Operating on FUZZY-VALUEs (create-fuzzy-value, fuzzy-union, fuzzy-intersection, fuzzy-modify)

Command: 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 (see Section **Error! Reference source not found.**).

Example 44

```
(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!
)
```

Command: 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.

Example 45

```
(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
```

Command: fuzzy-intersection

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

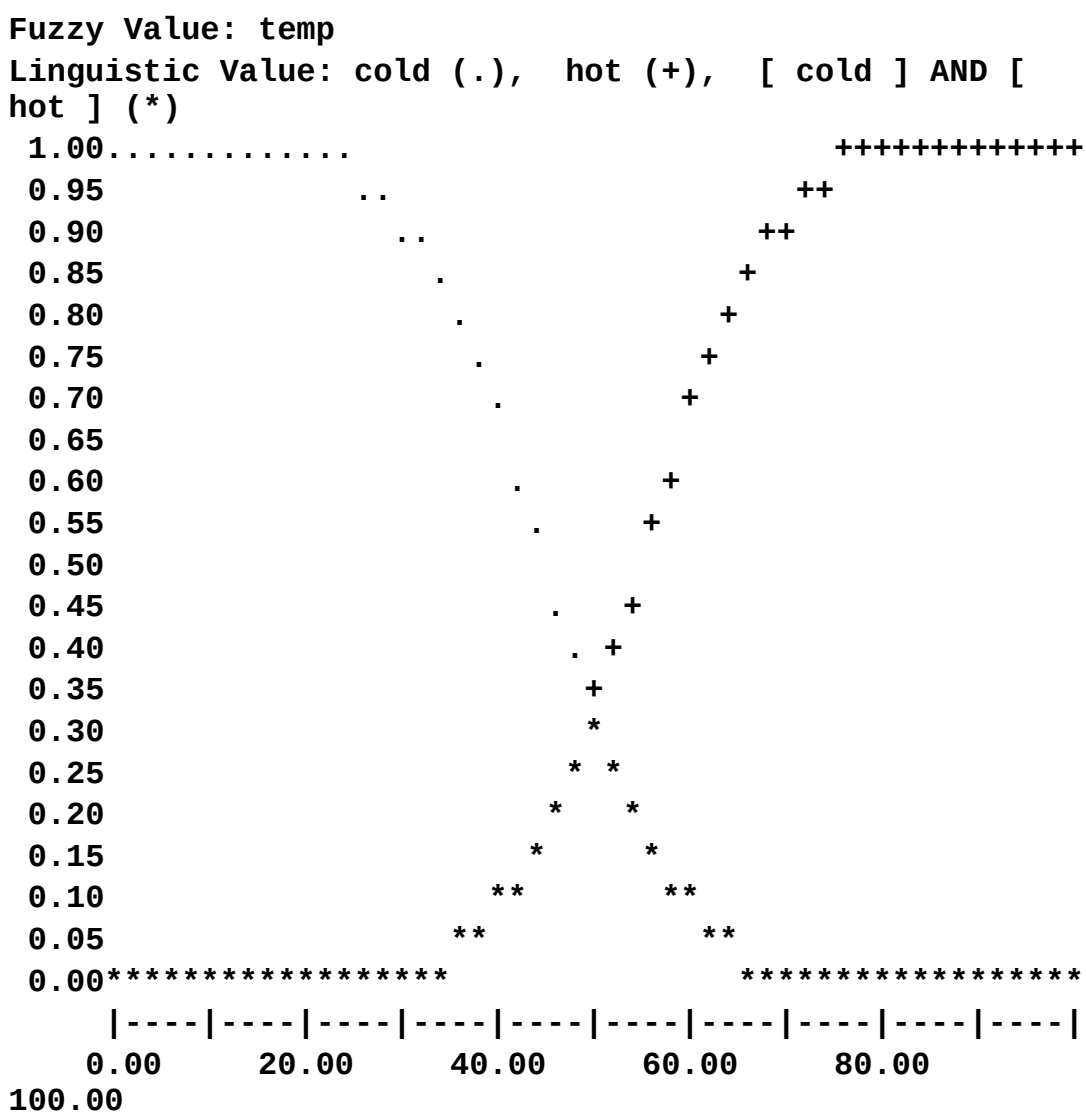
Purpose: Returns a new fuzzy value that is the intersection of two other fuzzy values. Both arguments must be of type FUZZY-VALUE.

Example 46

```
(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))
)
```



Command: 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.).

Example 47

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

Fuzzy Value: temp
Linguistic Value: hot (+),  extremely 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
```

1.1.12 Accessing a Fuzzy Slot in a Fact (get-fuzzy-slot)

Command: get-fuzzy-slot

Syntax: (get-fuzzy-slot ?<fact-var> [<slot-name>])

(get-fuzzy-slot <integer> [<slot-name>])

Purpose: This function will retrieve the fuzzy value associated with a fuzzy slot in a fact. The first argument can be a variable that is associated with a fact address or an integer that is the fact number for a fact. If the fact is a fuzzy deftemplate fact (one whose relation name is a fuzzy deftemplate name) then the second argument is not needed since the only slot for the fact is the fuzzy value. If the fact is a standard deftemplate fact with fuzzy slots, then the second argument is a symbol that identifies the slot to access. (Note that the

slot of fuzzy deftemplate facts is always name ‘GenericFuzzySlot’ and it could be accessed using that name.)

Example 48

```
(defrule test1-get-fuzzy-slot
  ?f <- (temp hot)
 =>
  (plot-fuzzy-value t * nil nil (get-fuzzy-slot ?f))
)

(defrule test2-get-fuzzy-slot
  ?f <- (system (name sysA) (t-outflow hot))
 =>
  (plot-fuzzy-value t * nil nil (get-fuzzy-slot ?f t-
outflow))
)
```

Lab 7

1.1.13 Displaying a Fuzzy Value in a Format Function

This is not a function or command but is an addition to CLIPS to allow the formatting of fuzzy values in a format function. The specifier %F is used.

Example 49

```
(deftemplate temp
   0 100 C
 ((cold (z 20 50)))
)
(assert (temp cold))
<Fact-0>
(format t "Value is '%F'%n" (get-fuzzy-slot 0))
Value is 'cold'
```

1.1.14 Plotting a Fuzzy Value (plot-fuzzy-value)

Command: 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.

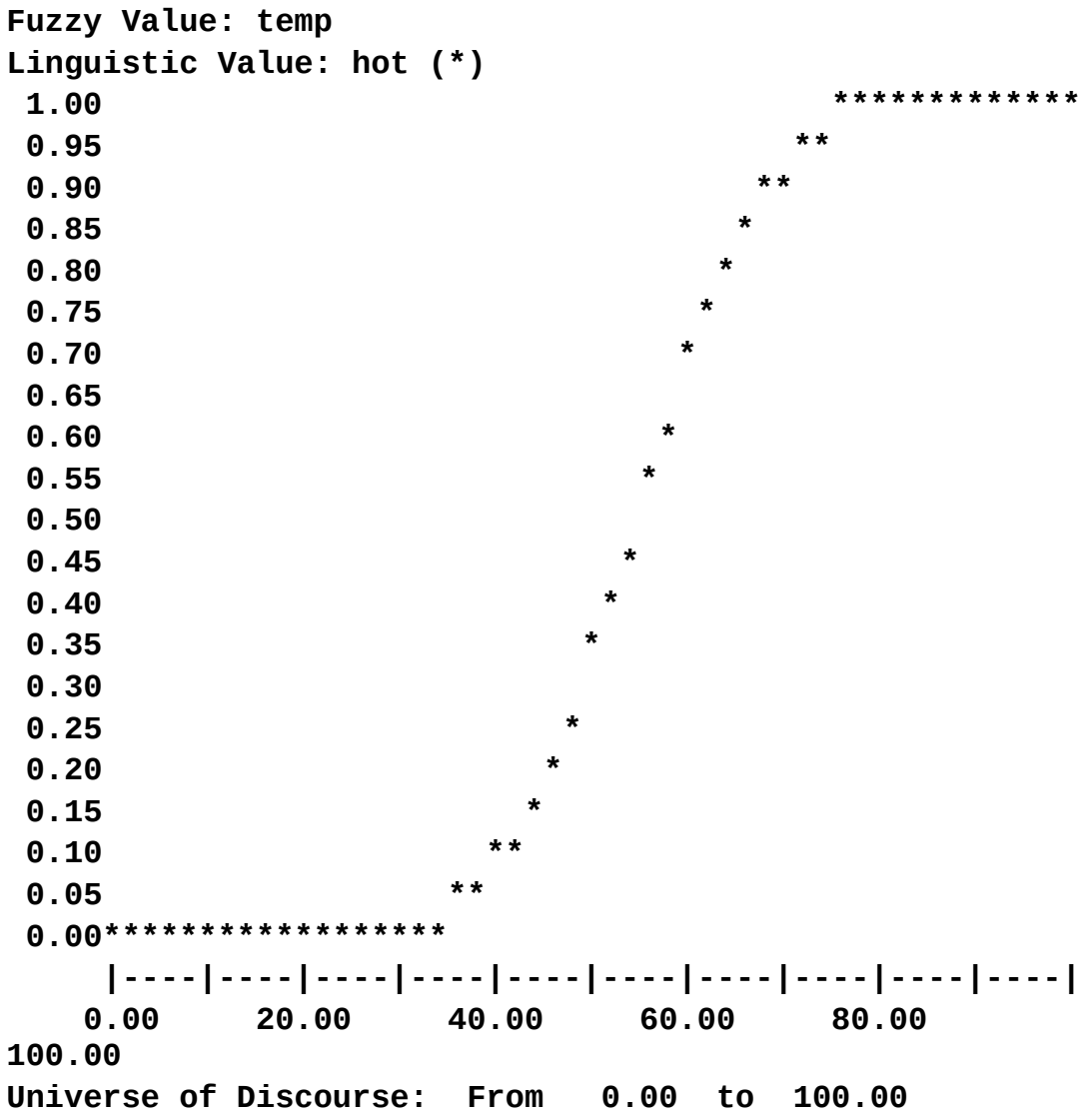
The fuzzy deftemplate associated with ALL fuzzy values to be plotted on the same plot must be the same one. This is required since the x axis must have the same meaning.

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.

Example 50

```
(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

```
(plot-fuzzy-value t * 20 70  (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***************
     |----|----|----|----|----|----|----|----|----|----|
   20.00     30.00     40.00     50.00     60.00
70.00
Universe of Discourse:  From   0.00  to  100.00
```

Example 51

```
(deftemplate system
    (slot name)
    (slot t-outflow (type FUZZY-VALUE temp))
)

(assert (system (name sysA) (t-outflow not hot)))
<Fact-1>

(plot-fuzzy-value t "+*" nil nil
    (create-fuzzy-value temp hot)
    (get-fuzzy-slot 1 t-outflow)
```

```
)

Fuzzy Value: temp
Linguistic Value: hot (+),  not 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
```

1.1.15 Controlling the Result of Defuzzification

The defuzzifcation functions are defined to return values that depend on the fuzzy value that they are provided with. The moment-defuzzify function (see section **Error! Reference source not found.**) is undefined in the case of a fuzzy set with area equal to zero. This can occur when the fuzzy set is a crisp value or the fuzzy set is flat at membership value of 0.0. In these cases, by default, the function will return the midpoint of the universe of discourse. To allow a user to detect these situations, a special function is provided.

Command: is-defuzzify-value-valid

Syntax: (is-defuzzify-value-valid)

Purpose: This function is used to check if the value returned by the last defuzzify function is valid or not. It returns TRUE if a valid value was be returned or

FALSE if not. In the current implementation only the moment-defuzzify function can return an invalid default value.

If no defuzzify functions have yet been called, the return value is TRUE.

Normally one will use this function immediately after the defuzzify function is called. For example:

Example 52

```
(defrule defuzzify-temperature
   ?f <- (temperature ?)
 =>
   (bind ?temperature-value (moment-defuzzify ?f))
   (if (is-defuzzify-value-valid)
      then
         ... do something with the defuzzified value
      else
         ... perhaps use the maximum-defuzzify function
   )
)
```

Lab 8

Simple FuzzyCLIPS Example

This lab describes the output of watching facts and rules fire for the example found in the file simplTst.clp. Please note that this was not intended to be a real example of the use of FuzzyCLIPS. For that see the other examples included in the distribution.

```
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>
```

Lab 9
Decision-making on stock market trading

Figure 9.1 shows a part of a program written in FuzzyCLIPS for decision-making on future investments. Three fuzzy input variables are used (economic situation, political situation, trend of the predicted value) for which fuzzy values are defined by using standard membership functions.
Create a decision make program in FuzzyCLIPS and test it.

```
;;; A part ofa FuzzyCLIPS program for stock market prediction

;; a single rule for decision making – a fuzzy value is inferred by it;
(defrule hybrid_system_step3
  (declare (CF 0.9))
  ?step3 <- (step3)
  (political_climate good)
  (economic_climate good)
  (predicted_value up)
  =>
  (assert (shares buy) CF 0.9)
  (assert (end_run))
  (retract ?step3))

;; a single rule for defuzzification and final decision making;
(defrule final
  (end_run)
  ?f <- (shares ?)
=>
  (printout t crlf "THE DECISION OBTAINED BY THE SYSTEM IS: " crlf (get-fs ?f) crlf   " with a
degree of certainty of: " (get-cf ?f) crlf "On the scale of 0-9 levels, where 0 means definitely
"sell", 5 means "hold", and 9 means definitely "buy", the suggested action is " (moment-
defuzzify ?f))
```

Figure 9.1. A part of a program written in FuzzyCLIPS for decision-making on future investment in stock market.

Lab 10
Control of a two-stage inverted pendulum

Inverted Pendulum Problem is shown graphically in figure 10.1. Fuzzy Controller 1 takes data from the upper stage of the two-stage pendulum. It has two inputs—angle A2 and angular velocity V2. The output of it called "Stage 1" is used as an input variable to Fuzzy Controller 2, which takes two more parameters—angle A1 and angular velocity V1 from the lower stage—and actually controls the current to the motor by producing an output value for the control variable "Motor." For each output value of the variable "Stage 1," a separate rule map is used to represent the rules for Fuzzy Controller 2. The rule map for Stage 1 = ZR is given in figure 10.2 with the rule map of the rules for Fuzzy Controller 1. In case of stage 1 = ZR the two maps are identical, but not in the other cases.
Define fuzzy variables and rules into MATLAB or FuzzyCLIPS. Test them.

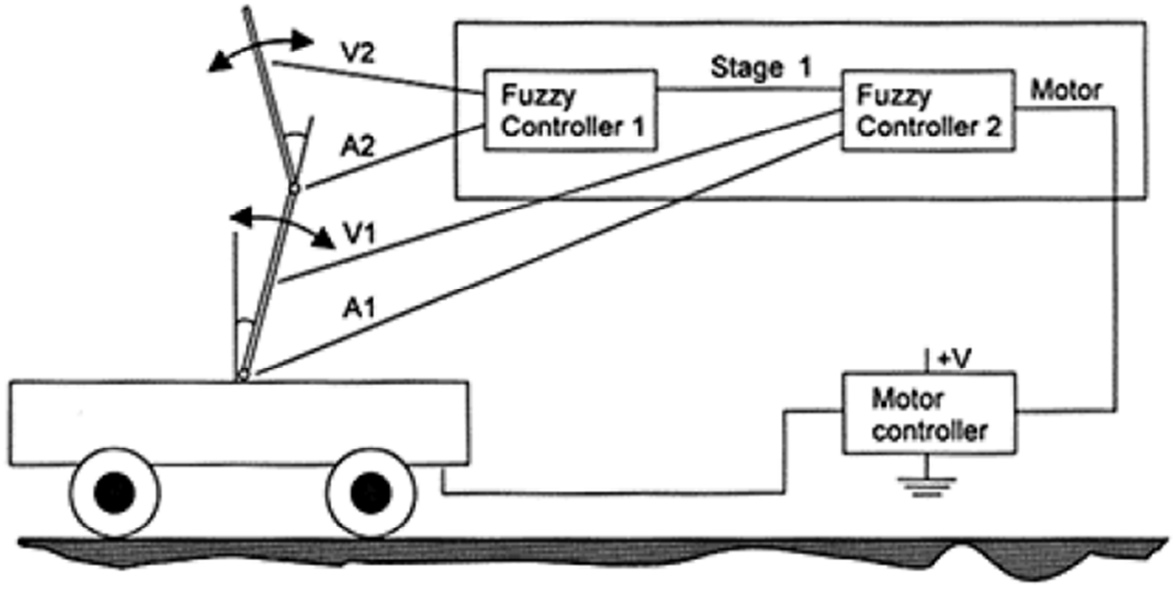


Figure 10.1 A sketch of the two-stages Inverted Pendulum Problem.

Controller One (Output: Stage 1)

Angle 2 \ Velocity 2	NL	NM	ZR	PM	PL
NL	NL	NL	NL	NM	NS
NM	NL	NL	NM	NS	PS
NS	NL	NM	NS	PS	PM
PS	NM	NS	PS	PM	PL
PM	NS	PS	PM	PL	PL
PL	PS	PM	PL	PL	PL

Controller Two (Output: Motor)
Stage 1 = ZR Zero

Angle 1 \ Velocity 1	NL	NM	ZR	PM	PL
NL	NL	NL	NL	NM	NS
NM	NL	NL	NM	NS	PS
NS	NL	NM	NS	PS	PM
PS	NM	NS	PS	PM	PL
PM	NS	PS	PM	PL	PL
PL	PS	PM	PL	PL	PL

Figure 10.2 Two maps of fuzzy rules for the two-stages Inverted Pendulum Problem. The map for controller one and one map only of fuzzy rules for the controller two are shown.

Lab 11
Fuzzy system for the Car Monitoring Problem

Figure 11.1 shows a fuzzy quantization of the parameters of the task. The parameter "brakes' response" is represented here on the universe of time with three linguistic labels—"quick," "normal," and "slow." The state of the cooling system is represented by three linguistic fuzzy values—"underheating," "normal," and "overheating." The status of the temperature gauge is represented by two labels—"OK" and "damaged"—on the universe of grades of sensitivity of the gauge. The variable "temperature" is represented by the labels "low," "normal," and "high." The conclusion "stop the car," can be represented as a single-valued membership function representing the certainty of the advice. A chain fuzzy inference is performed in this case. When the system is realized by the use of the centroid defuzzification inference method, even slight matching of the conditions in the fuzzy rules by the current status of the car will cause a message to be communicated to the driver. The fuzzy rules are:

Rule 1: IF Brakes'_response is Slow, THEN Message is Stop_the_car
Rule 2: IF Cooling_status is Overheating, THEN Message is Stop_the_car
Rule 3: IF Temperature is High and Gauge_status is OK, THEN Cooling_status is Overheating

Define fuzzy variables and rules into MATLAB or FuzzyCLIPS. Test them.

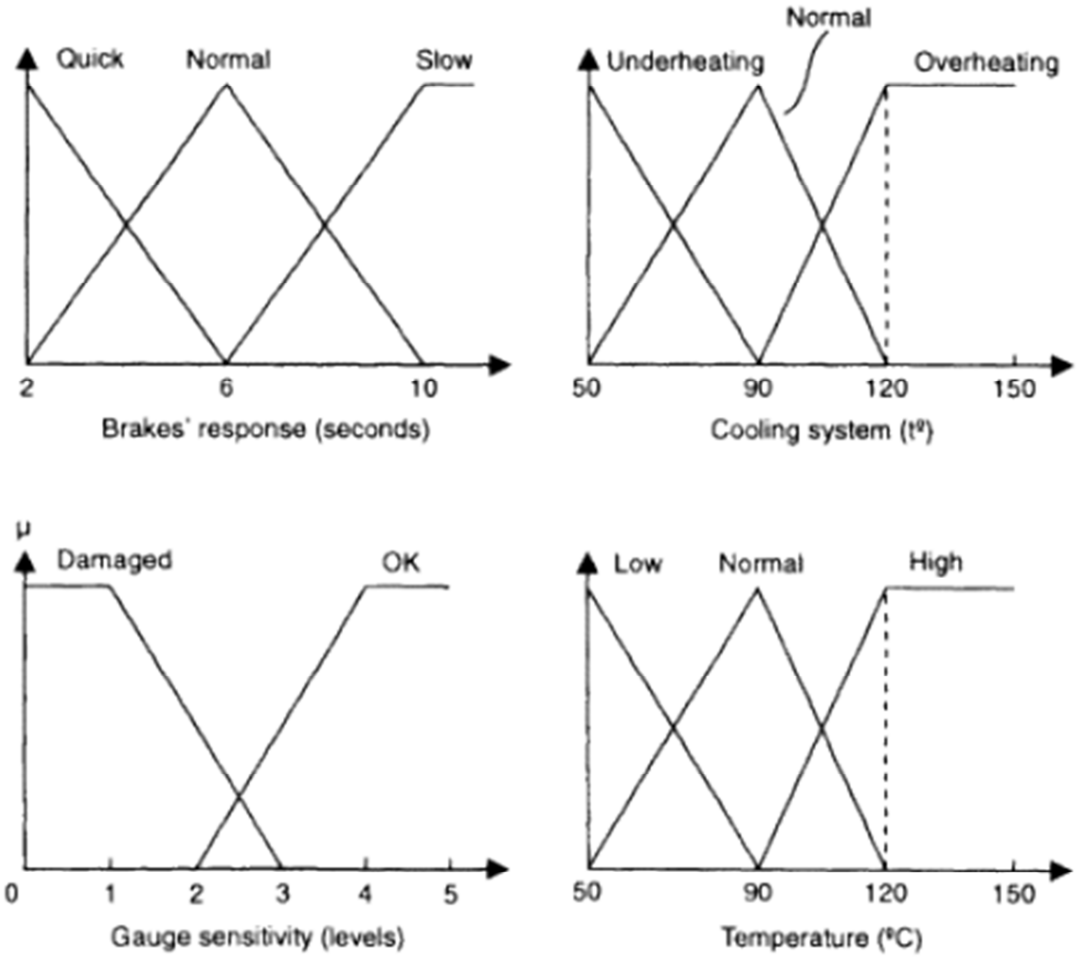


Figure 11.1 Membership functions of linguistic values for the Car Monitoring Problem.

Lab 12
A fuzzy diagnostic system for the Medical Diagnosis

A fuzzy diagnostic system for the Medical Diagnosis case example is explained here. The rules vaguely articulated there are now presented in a more precise form as fuzzy rules, shown in figure 12.1. M1, M2, M3, and M4 are manifestations (symptoms), defined with their linguistic values, and D1, D2, D3, and D4 are disorders, defined with linguistically represented confidence factors CF1, CF2, CF3, and CF4. In the same figure, another representation of the same rules after substituting the fuzzy quantifiers with its corresponding typical numerical values, as given in Chen (1988), is shown. A table of corresponding fuzzy linguistic labels, numerical intervals, and typical single values for the frequency of appearance and strength of symptoms is given in figure 12.2.
Define fuzzy variables and rules into MATLAB or FuzzyCLIPS. Test them.

Fuzzy Rules:
Rule 1: IF(M1 is Always AND M2 is Weak AND M3 is No AND M4 is No)
THEN D is D1 (CF is very strong);

Rule 2: IF(M1 is No AND M2 is Always AND M3 is Weak AND M4 is No)
THEN D is D2 (CF is very strong);

Rule 3: IF(M1 is More_or_less_weak AND M2 is No AND M3 is Always AND M4 is No)
THEN D is D3 (CF is very strong);

Rule 4: IF(M1 is Weak AND M2 is More_or_less_weak AND M3 is Always AND M4 is No)
THEN D is D4 (CF is very strong);

Use of typical values instead of fuzzy values:
Rule 1: IF(M1=1.0, M2=0.2, M3=0.0, M4=0.0) THEN D1 (CF=0.99);

Rule 2: IF(M1=0.0, M2=1.0, M3=0.2, M4=0.0) THEN D2 (CF=0.99);

Rule 3: IF(M1=0.3, M2=0.0, M3=1.0, M4=0.0) THEN D3 (CF=0.99);

Rule 4: IF(M1=0.2, M2=0.3, M3=1.0, M4=0.0) THEN D4 (CF=0.99);

Figure 12.1 Two sets of similar rules for the Medical Diagnosis Problem.

Fuzzy label	Numerical interval	Typical value
Always	[1.00, 1.00]	1.0
Very strong	[0.95, 0.99]	0.99
Strong	[0.80, 0.94]	0.9
More or less strong	[0.95, 0.79]	0.7
Medium	[0.45, 0.64]	0.5
More or less weak	[0.30, 0.44]	0.3
Weak	[0.10, 0.29]	0.2
Very weak	[0.01, 0.09]	0.05
No	[0.00,0.00]	0.0

Figure 12.2 Corresponding linguistic values, numerical intervals, and typical exact values for representing frequency of appearance and strength of symptoms.

Lab 13
The Investment Adviser Problem

Here, a solution to this problem is presented. It can be compared with its solution as a production system. A set of fuzzy rules for solving the problem are shown in figure 13.1. Standard triangular membership functions are specified for the fuzzy values. The system is written in the syntax of FLIE.
Define fuzzy variables and rules into MATLAB or FuzzyCLIPS. Test them.

MEMBERSHIPS
INPUT amount: {0, 10, 25, 40, 55, 70, 100}
- Very_small: {1, 1, 0, 0, 0, 0, 0}
- Small: {0, 0, 1, 0, 0, 0, 0}
- Average: {0, 0, 0, 1, 0, 0, 0}
- Large: {0, 0, 0, 0, 1, 0, 0}
- Very_large {0, 0, 0, 0, 0, 1, 1}

INPUT risk: {0, 0.1, 25, 40, 55, 70, 100}
- no: {1, 0, 0, 0, 0, 0, 0}
- very_low: {1, 1, 0, 0, 0, 0, 0}
- low: {0, 0, 1, 0, 0, 0, 0}
- medium: {0, 0, 0, 1, 0, 0, 0}
- high: {0, 0, 0, 0, 1, 0, 0}
- Very_high {0, 0, 0, 0, 0, 1, 1}

INPUT period: {0, 5, 12, 24, 36, 48, 60}
- short: {1, 1, 0.6, 0, 0, 0, 0}
- medium: {0, 0, 0, 1, 1, 0, 0}
- long: {0, 0, 0, 0, 0, 1, 1}

INPUT income: {0, 10, 15, 25, 36, 40, 46, 64, 67, 100}
- poor: {1, 1, 0.8, 0.1, 0, 0, 0, 0, 0, 0}
- good: {0, 0, 0, 0, 1, 1, 1, 0, 0, 0}
- excellent: {0, 0, 0, 0, 0, 0, 0.2, 0.9, 1, 1}

OUTPUT investment: {0, 20, 40, 60, 80, 100}
- none: {1, 0, 0, 0, 0, 0}
- very_small: {0, 1, 0, 0, 0, 0}
- small: {0, 0, 1, 0, 0, 0}
- average: {0, 0, 0, 1, 0, 0}
- large: {0, 0, 0, 0, 1, 0}
- all: {0, 0, 0, 0, 0, 1}

RULES
If <amount is average> and <risk is high> and <period is short> and <income is good>
Then <investment is very_small>
else
If <amount is small> and <risk is no> and <period is short> and <income is poor>
Then <investment is all>
else
If <amount is small> and <risk is high> and <period is medium> and <income is good>

Then <investment is very_small>
else
If <amount is average> and <risk is high> and <period is long> and <income is good>
Then <investment is none>
else
If <amount is very_large> and <risk is no> and <period is medium> and <income is excellent>
Then <investment is all>
else
If <amount is average> and <risk is high> and <period is short> and <income is poor>
Then <investment is small>
else

Figure 13.1 A fuzzy system for solving the Investment Adviser Problem written in the syntax of FLIE. The membership functions are chosen as triangular, defined by their centers and uniformly distributed over the whole universe. The output variable represents how much should be invested in bank. The rest is meant to be invested in shares.