



**Dpto. Sistemas Informáticos y Computación  
Escuela Técnica Superior de Ingeniería Informática  
UNIVERSITAT POLITÈCNICA DE VALÈNCIA**

# **Técnicas, Entornos y Aplicaciones de Inteligencia Artificial**

## **Fuzzy-CLIPS**

1.- Variables difusas. Valores difusos, funciones de pertenencia. Modificadores. ....	2
2.- Hechos Difusos .....	5
2.1.- Declaración de templates con slots difusos. ....	5
2.2.- Declaración de hechos difusos mediante expresiones ASSERT. ....	6
2.3.- Declaración de hechos iniciales: Deffacts .....	7
2.4.- Fusificación de valores Crisp .....	7
3.- Reglas difusas .....	9
4.- Inferencia difusa .....	11
5.- Defusificación .....	12
6.- Ejemplos .....	13

### **Evaluación de la Práctica**

# FuzzyCLIPS

FuzzyClips es una extensión fuzzy de CLIPS para manipular hechos y reglas difusas que ofrece un soporte para el desarrollo de sistemas expertos difusos en el entorno de CLIPS. Al tratarse de una extensión de CLIPS, utiliza una sintaxis muy parecida a la de Jess, sistema utilizado en la primera práctica.

FuzzyClips es de dominio público (desarrollado por el Inst. for Information Technology, National Research Council of Canada) y permite integrar datos y reglas difusas (imprecisas) y no difusas. FuzzyClips puede descargarse de [http://awesom.eu/~cygal/archives/2010/04/22/fuzzyclips\\_downloads/index.html](http://awesom.eu/~cygal/archives/2010/04/22/fuzzyclips_downloads/index.html).

La **instalación de FuzzyClisp** consiste simplemente en copiar en un mismo directorio los componentes: FuzzyClips.exe y CLIPSedt.exe

El presente documento es una simplificación de las funcionalidades presentes en FuzzyClips, que puede extenderse con la información en su manual.

## 1.- Variables difusas. Valores difusos, funciones de pertenencia. Modificadores.

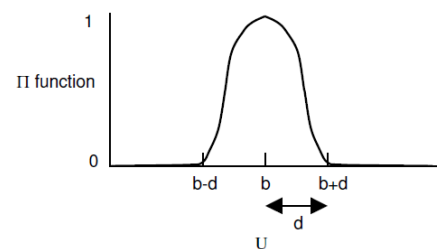
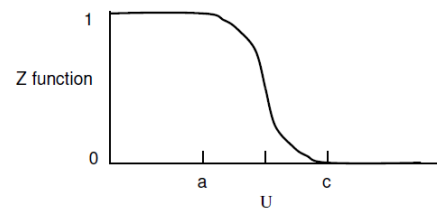
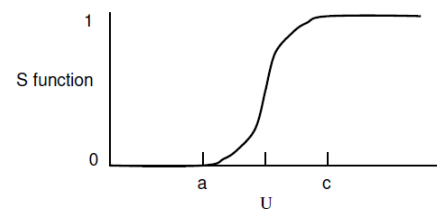
Las **variables difusas** se crean a partir de plantillas (**templates**), donde también se definen sus **valores difusos** (o valores lingüísticos) mediante las funciones de pertenencia.

En la plantilla se define:

- La variable difusa, por ej. edad, temperatura, etc.
- Universo (límites del universo de valores)
- Los valores difusos que puede tomar, por ej. alto, medio, bajo, etc.
- Las funciones de pertenencia asociados a cada uno de los valores difusos.

Las funciones de pertenencia se pueden definir de dos maneras diferentes:

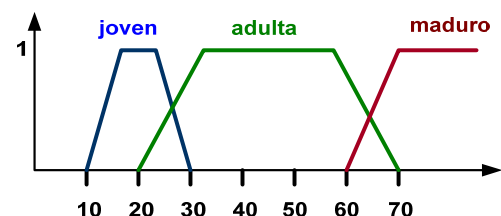
- Definiendo los puntos característicos (función de pertenencia específica). Pueden indicarse tantos puntos como sea necesario. El último valor se mantiene hasta el límite superior del universo.
- Usando unas funciones ya predefinidas: s, z, PI (Más información en el manual).



### Ejemplos:

```
(deftemplate edad ;Variable difusa
  0 120 años ;Universo
  ( (joven (10 0) (15 1) (25 1) (30 0)) ;Valores difusos
    (adulta (20 0) (30 1) (60 1) (70 0))
    (mayor (60 0) (70 1)))
```

```
(deftemplate singleton
  0 10 unit
  ( (tres (3 0) (3 1) (3 0))
    (cinco (5 0) (5 1) (5 0))))
```



(deftemplate <b>estatura</b> 0 250 cm ( (bajo (0 1) (100 1) (150 0)) (medio (100 0) (150 1) (170 1) (180 0)) (alto (170 0) (180 1))))	(deftemplate <b>temperatura</b> 30 50 grados ( (bajo (35 1) (37 0)) (medio (35 0) (36 1) (37 0)) (alto (36 0) (37 1))))
--	--

### Dibujo gráfico de una función de pertenencia

Permite dibujar la función de pertenencia de un valor difuso:

**(plot-fuzzy-value <salida> <chars> <low> <high> {<fuzzy-value>})**

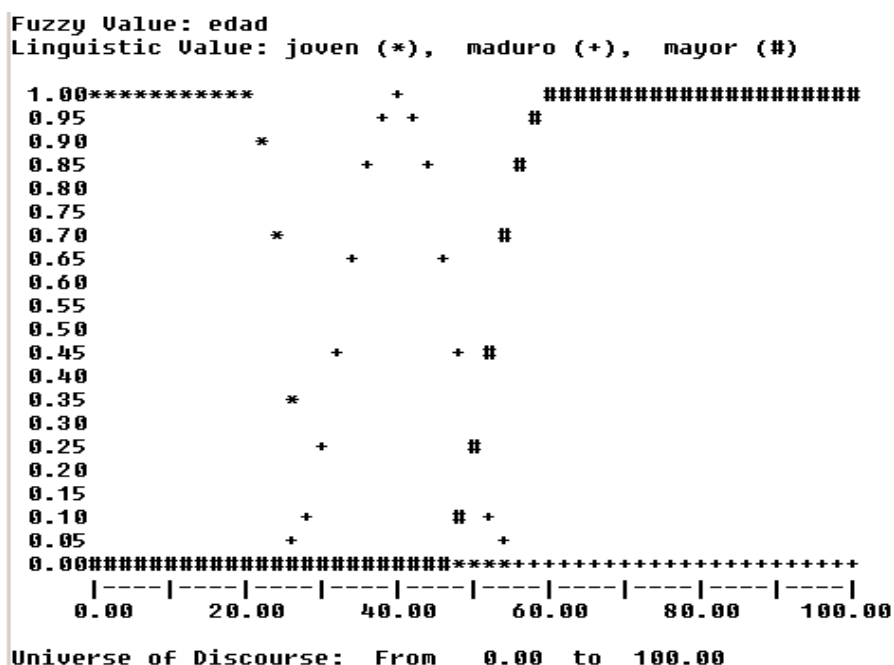
<salida>: indicación de la salida (t para la salida estándar)  
 <chars>: caracteres para usar en el dibujo (típicamente \*, +)  
 <low>, <high>: límites inferior/superior del dibujo  
 <fuzzy-value>: expresión que indica el valor a dibujar.  
 Típicamente: (create-fuzzy-value variable valor)

#### Ejemplo:

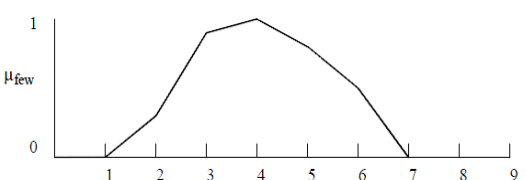
Asumido que se define: (deftemplate edad 0 100 años  
 ((joven (z 20 30))  
 (maduro (PI 15 40))  
 (mayor (s 45 60))))

Mediante la expresión: (plot-fuzzy-value t "\*" 0 100 ;Solo en consola!  
 (create-fuzzy-value edad joven)  
 (create-fuzzy-value edad maduro)  
 (create-fuzzy-value edad mayor))

Se obtendría:



## Otros ejemplos:

<pre>(deftemplate ejemplo 0 10 unit   ((medio (0 0)(1 0.1)(2 0.3)(3 0.5)(4 0.7)     (5 0.9)(6 1)(7 0.8)(8 0.5)(9 0.3)     (10 0))   (singleton (9 0)(9 1)(9 0))))</pre>	<p>Que puede visualizarse mediante la expresión:</p> <pre>(plot-fuzzy-value t "*" 0 10   (create-fuzzy-value ejemplo medio)   (create-fuzzy-value ejemplo singleton))</pre>
<pre>(deftemplate amount ;a linguistic variable declaration   0 9 ;universe of discourse limits (no units)   ( ;start of primary term declaration, 'few'     (few (1 0) (2 0.3) (3 0.9) (4 1) (5 0.8) (6 0.5) (7 0)))     ;end of primary term declarations   ) ;end of fuzzy deftemplate</pre>	<pre>(plot-fuzzy-value t "*" 0 10   (create-fuzzy-value amount few))</pre> 

## Modificadores Lingüísticos

Los modificadores lingüísticos modifican la forma de un conjunto difuso previamente definido. Por ejemplo:

$$\text{very}(y) = y^2 \qquad \text{more-or-less}(y) = \sqrt{y}$$

FuzzyClips provee los siguientes principales modificadores lingüísticos:

not	$1-y$	very	$y^2$	somewhat	$y^{1/3}$
more-or-less	$\sqrt{y}$	extremely	$y^3$	plus	$y^{1.25}$

norm: normaliza el conjunto difuso, tal que se escala para que el máximo valor sea 1.

Por ejemplo, podemos tener:

<pre>(deftemplate temperatura 0 80 grados   ((frio (z 10 25))   (congelado <b>plus</b> frio)))</pre>	<p>Puede visualizarse mediante:</p> <pre>(plot-fuzzy-value t "*" 0 80   (create-fuzzy-value temperatura frio)   (create-fuzzy-value temperatura congelado))</pre>
--	---

Los operadores lingüísticos se pueden combinar con expresiones lógicas:

<pre>(deftemplate temperatura 0 80 grados   ((frio (z 10 25))   (calor (s 30 40))   (templado not [ frio or calor ])))</pre>	<p>Cuyas valores pueden visualizarse con:</p> <pre>(plot-fuzzy-value t "*+#" 0 80   (create-fuzzy-value temperatura frio)   (create-fuzzy-value temperatura templado)   (create-fuzzy-value temperatura calor))</pre>
--	---

Además se pueden añadir nuevos **modificadores lingüísticos** mediante funciones.

```
(deffunction dem-fun(?p)
  (if (> ?p 0.5) then 1.0
    else 0.0))
```

Mediante esta línea se añadiría un nuevo modificador **muy-int** que usa la función **dem-fun**.

```
(add-fuzzy-modifier muy-int dem-fun)
```

### Ejemplo:

(deftemplate cantidad 0 100 ( (poco (z 10 50)) (muypoco muy-int poco)))	Cuyos valores pueden visualizarse con: (plot-fuzzy-value t "+" 0 50 (create-fuzzy-value cantidad poco) (create-fuzzy-value cantidad muypoco))
---	--

Si queremos quitar un modificador lingüístico previamente definido: (remove-fuzzy-modifier muy-int)

## 2.- Hechos Difusos

Un hecho difuso es una expresión de la forma:

(*variable-difusa* *valor-difuso*),  
(*variable-difusa* *modificador valor-difuso*), o  
(*variable-difusa* *expresión-lógica*),

Donde 'variable-difusa' debe haber sido definida previamente e incluir, entre sus valores, el valor-difuso que se aserta.

Las expresiones lógicas combinan valores difusos mediante expresiones **AND** (mínimo de los valores de las funciones de pertenencia), **OR** (máximo de los valores de las funciones de pertenencia) y **NOT** (1- función de pertenencia). **NOTA:** el operador AND tiene más prioridad que el OR.

### Ejemplo

Una vez definidas las variables difusas 'edad' y 'estatura' anteriores, podemos tener como hechos difusos:

(edad adulta) (estatura very bajo), (edad adulta OR joven), etc.

Estos hechos pueden ser asertados mediante expresiones assert:

(assert (edad adulta)) (assert (estatura very bajo)) (assert (edad adulta OR joven))

O inicializados mediante expresiones deffacts:

(defacts ejemplo  
(edad adulta) (estatura very bajo))

**Nota.** No se deben definir hechos difusos **no estructurados** con más de un hecho difuso o mezclándolos con hechos Crisp. Por ejemplo, no se admite:

(juan edad adulta)  
(juan edad adulta estatura bajo)  
(persona nombre adan edad adulta estatura alto)

Esta información la asumiría como información Crisp, es decir, no difusa.

### 2.1- Declaración de templates con slots difusos.

Una template (hecho estructurado) puede contener slots con valores difusos y slots con valores Crisp. Una template se define de la forma:

(deftemplate <nombre-template>  
    <crisp-slot>+ | <fuzzy-slot>+)

Los slots difusos corresponden a variables difusas previamente definidas y se declaran de la forma:

---

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

Por ejemplo, asumiendo ya definidas las variables difusas 'edad' y 'temperatura', podemos declarar el template:

```
(deftemplate persona
  (slot nombre (type SYMBOL))
  (slot edad-difusa (type FUZZY-VALUE edad))
  (slot estatura-difusa (type FUZZY-VALUE estatura))
  (slot vive (type SYMBOL)))
```

## 2.2.- Declaración de hechos difusos mediante expresiones ASSERT.

Una vez definidas las variables y valores difusos, se pueden definir los hechos difusos mediante expresiones **assert**:

```
(assert ( <crisp-fact> |
         fuzzy-variable-name <description of fuzzy set> |
         template-name <crisp-slot-description>+
         (fuzzy-slot-name <description of fuzzy set>)* ))
```

donde <description of fuzzy set> puede ser un <valor difuso> definido para la variable o slot difuso, o bien el par <modificador> <valor difuso>, o bien una expresión lógica difusa:

<description of fuzzy set> := <valor difuso> | < modificador> <valor difuso> | <fuzzy-logical-expresion>

### Ejemplos:

Supongamos definidas las variables y sus valores difusos:

(deftemplate <b>edad</b> 0 100 años ( ( <b>joven</b> (10 0) (15 1) (25 1) (30 0)) ( <b>adulta</b> (20 0) (30 1) (60 1) (70 0)) ( <b>madura</b> (60 0) (70 1))))	(deftemplate <b>estatura</b> 0 250 cm (( <b>bajo</b> (0 1) (100 1) (150 0)) ( <b>medio</b> (100 0) (150 1) (170 1) (180 0)) ( <b>alto</b> (170 0) (180 1))))
--	---

Podemos declarar hechos difusos de la forma:

```
(assert (edad adulta))
(assert (estatura very alto))
```

Y definiendo el siguiente template:

```
(deftemplate persona
  (slot nombre (type SYMBOL))
  (slot edad-difusa (type FUZZY-VALUE edad))
  (slot estatura-difusa (type FUZZY-VALUE estatura))
  (slot vive (type SYMBOL)))
```

**;Nota: Tipos en mayúscula!!**

Podemos declarar el hecho:

```
(assert (persona (nombre david) (edad-difusa joven) (estatura-difusa very alto) (vive Valencia)))
```

Asimismo, asumida definida la variable difusa:

---

```
(deftemplate grupo 0 20 miembros ;declaración de la variable grupo
  ((pocos (3 1) (6 0)) ;valor primario pocos
   (muchos (4 0) (6 1)))) ;valor primario muchos
```

Podemos asertar hechos como:

```
(assert (grupo pocos))
(assert (grupo (1 0) (5 1) (7 0)) ) ;Nuevo valor
(assert (grupo NOT [ very pocos OR muchos ] )) ;es necesario dejar un espacio junto a [ y ]
(assert (grupo (z 4 8)))
```

**Nota:** Los slots difusos deben tener siempre un valor en la declaración del template. **No deben** dejarse sin valor.

### 2.3.- Declaración de hechos iniciales: Deffacts

Las expresiones deffacts utilizan las expresiones assert definidas anteriormente. De esta manera, podemos declarar un conjunto de hechos iniciales crisp y fuzzy que se inicializarían con '(reset)'. Por ejemplo:

```
(deffacts ejemplo1
  (edad adulta)
  (estatura very alto))

(deffacts ejemplo2
  (persona (nombre adan) (edad-difusa joven) (estatura-difusa very alto) (vive Valencia))
  (persona (nombre eva) (edad-difusa joven) (estatura-difusa alto) (vive Valencia))
  (persona (nombre moises) (edad-difusa madura) (estatura-difusa medio)))
```

Podemos, posteriormente, añadir nuevos hechos mediante expresiones assert:

```
(assert (persona (nombre juan) (edad-difusa adulta) (estatura-difusa medio) (vive Madrid)))
```

### 2.4.- Fusificación de valores Crisp

Hemos visto cómo se puede asertar información difusa mediante expresiones assert, pero hay casos en los que la información de partida es un valor Crisp, es decir un valor concreto no difuso. Por ejemplo, imaginemos que sabemos que la edad es de 35 años y la altura de 172 cm y queremos asertar dicha información. Para ello podemos utilizar la siguiente función *fuzzify* que viene como ejemplo con el manual de usuario de FuzzyClips (ejemplo llamado fuzzify.clp):

```
(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)) ))
            )))
```

---

Esta función se puede invocar de la forma "(fuzzify edad 35 0.1)" que asertará automáticamente: (assert-string "(edad (34.9 0) (35 1) (35.1 0))"). De esta forma se asertará información sobre la edad con un valor de certidumbre correspondiente al rango que se haya definido en la invocación de la función. NOTA: si queremos definir un valor singleton, será tan sencillo como invocar "(fuzzify edad 35 0)".

**Nota:** La utilización de la función fuzzify requiere utilizar variables difusas de forma aislada, y no como slots difusos de una template.

## 2.5.- Lectura de valores críps o difusos por consola

A menudo es útil introducir al sistema hechos que correspondan a valores leídos por consola, mediante la expresión (read), y asertar dichos valores, como hechos del sistema. Esto permite una mayor interacción con el usuario.

Sin embargo, fuzzy-clips **no permite asertar valores difusos leídos directamente desde consola**. Los valores difusos necesitan ser explícitamente asertados mediante expresiones assert o deffacts.

Para poder introducir valores difusos, a partir de valores críps o de símbolos (por ejemplo: alto, bajos, etc..) leídos por consola podemos utilizar la función assert-string.

Por ejemplo, asumamos definida la variable difusa edad y estatura:

(deftemplate <b>edad</b> 0 100 años ( ( <b>joven</b> (10 0) (15 1) (25 1) (30 0)) ( <b>adulta</b> (20 0) (30 1) (60 1) (70 0)) ( <b>madura</b> (60 0) (70 1))))	(deftemplate <b>estatura</b> 0 250 cm (( <b>bajo</b> (0 1) (100 1) (150 0)) ( <b>medio</b> (100 0) (150 1) (170 1) (180 0)) ( <b>alto</b> (170 0) (180 1))))
--	---

Podemos utilizar una regla de la forma siguiente para asertar como valor difuso el mismo símbolo (que representa un valor fuzzy) leído por consola.

```
(defrule leerconsola
  (initial-fact)
=>
  (printout t "Introduzca la edad: joven, adulta, madura" crlf)
  (bind ?Redad (read))
  (assert-string (format nil "(edad %s)" ?Redad)) )
```

Esta regla también puede codificarse como una función.

Por otra parte, podemos usar la función anterior **fuzzify** para asertar un valor difuso en función de un valor críps leído:

```
(defrule leerconsola
  (initial-fact)
=>
  (printout t "Introduzca la edad en annos" crlf)
  (bind ?Redad (read))
  (fuzzify edad ?Redad 0.1))
```

Y también, en caso de querer asertar la fusificación *singleton* de un valor críps:

```
(defrule leerconsola
  (initial-fact)
=>
  (printout t "Introduzca la edad en annos" crlf)
  (bind ?Redad (read))
  (assert (edad (?Redad 0) (?Redad 1) (?Redad 0))))
```



---

### 3.- Reglas difusas

#### Antecedentes

Los antecedentes de una regla difusa representan condiciones sobre hechos difusos o hechos crisp. Las condiciones fuzzy son de la forma:

```
(fuzzy-variable-name <linguistic-expr>) | (fuzzy-variable-name ?<var-name>) | (fuzzy-variable-name ? ) |  
(fuzzy-variable-name ?<var-name> & <linguistic-expr>) | (template-name <fuzzy-slot-description>+)
```

donde, <fuzzy-slot-description><sup>+</sup>) puede ser:

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

Donde <linguistic-expr> puede representar un valor difuso, con modificadores y/o expresiones lógicas (and/or).

Además, se admite el conjunto de patrones sobre valores crisp y condiciones <test>.

#### Consecuentes

El consecuente de una regla difusa utiliza las expresiones assert previamente vistas, así como el resto de funciones Clips.

#### EJEMPLO-1:

Supongamos la siguiente declaración de variable difusa:

```
(deftemplate tanque 0 80 litros  
  ((bajo (10 1)(30 0))  
   (medio (20 0)(35 1)(45 1)(60 0))  
   (alto (50 0)(70 1))))
```

Y asertado el hecho simple: (assert (tanque plus alto))

Podemos escribir una regla de la forma:

```
(defrule danger  
  (tanque extremely alto)  
  =>  
  (printout t "El tanque puede desbordarse. PELIGRO!"crlf)  
  (assert (alarma)))
```

Después de ejecutar (mediante "(run)") se obtendría la conclusión alarma (CF=0.85), en la medida en que alto cumple (0.85) la condición de extremadamente alto.

#### EJEMPLO-2:

Supongamos la siguiente declaración de variable difusa altura:

```
(deftemplate altura  
  0 8 pies  
  ( (baja (Z 3 4.5))  
    (media (pi 0.8 5))  
    (alta (S 5.5 6)) ))
```

Y el siguiente template, que incluye slots difusos y crisp:

```
(deftemplate persona
  (slot nombre)
  (slot ht (type FUZZY-VALUE altura)))
```

Podemos escribir la siguiente regla:

```
(defrule example
  (persona (nombre ?n) (ht ?h & very alta))
=>
  (printout t ?n " es muy alta, con una altura aproximada de "
    (maximum-defuzzify ?h) " " (get-u-units ?h) crlf ))
```

Esta regla se activaría ante la aserción de un hecho como: (assert (persona (nombre Adan) (ht alta)))

### Notas importantes:

- La expresión (ht ?h & very alta) puede ser similar a (ht very alta), pero no obtendríamos el valor en ?h para utilizarlo en la parte derecha.
- No es posible utilizar la función 'test' con valores difusos. Es decir:

```
(defrule example
  (persona (nombre ?n) (ht ?h))
  (test (eq ?h alta)))
```

**no haría** matching difuso.

- La función (get-u-units ?h) devuelve un *string* que corresponde a las unidades en las que se ha definido el valor difuso ?h

También:

<b><i>Función</i></b>	<b><i>Devuelve</i></b>
(get-u-units <fact-index>)	Unidades del conjunto
(get-u-from <fact-index>)	Valor inferior universo
(get-u-to <fact-index>)	Valor superior universo
(get-u <fact-index>)	Rango del universo

- Para introducir valores Crisp, y que puedan hacer matching con valores difusos, es necesario utilizar la función *fuzzify* explicada en el apartado 2.4 o definir el valor Crisp como un valor difuso (de tipo singleton en la propia definición de la variable difusa).

**Ejemplo:** Conocemos que una persona tiene 25 años, y tenemos definidos los conjuntos difusos:

```
(deftemplate edad 0 100 años
  ( (joven (10 0) (15 1) (25 1) (30 0))
    (veinticinco (25 0) (25 1) (25 0))
    (adulta (20 0) (30 1) (60 1) (70 0))
    (madura (60 0) (70 1))))
;definimos el singleton, edad 25.

(deftemplate estatura 0 250 cm
  ( (bajo (0 1) (100 1) (150 0))
    (medio (100 0) (150 1) (170 1) (180 0))
    (alto (170 0) (180 1))))
```

Ahora tenemos el hecho inicial

```
(defacts ejemplo (edad veinticinco))
```

Y podemos tener la regla:

```

(defrule edades
  (edad adulta)
  =>
  (printout t "Los adultos son altos" crlf)
  (assert (estatura alto)))

```

La premisa (edad adulta) hará matching con el hecho difuso asertado (edad veinticinco). Análogamente, podíamos NO haber definido el singleton veinticinco y haber usado la función "(fuzzify edad 25 0)" del apartado 2.4. En el primer caso hemos definido un valor singleton que estará siempre disponible en la variable difusa edad, mientras que en el segundo caso simplemente asertamos información Crisp que está siendo fusificada.

## 4.- Inferencia difusa

FuzzyClips utiliza el **modus ponens difuso** para obtener las consecuencias inferidas. Se puede elegir entre dos reglas composicionales, Max-min y Max-prod:

```
(set-fuzzy-inference-type <tipo>)
```

que son las dos reglas más utilizadas en el razonamiento difuso.

**EJEMPLO.** Inferencia difusa sobre el control de la temperatura de una habitación.

Primero definimos las variables difusas.

**ENTRADA:** TEMPERATURA.

```

(deftemplate Temp 5 50 Celsius
  ((frio (z 10 20))
   (templado (pi 5 25))
   (calor (s 30 40))))

```

**SALIDA:** APERTURA DE LA VALVULA

```

(deftemplate valvula 0 90
  grados_de_apertura
  ((poco (z 10 30))
   (medio (pi 30 45))
   (mucho (s 70 80)))

```

Definimos las reglas:

<pre> (defrule hace_frio   (Temp frio)   =&gt;   (assert (valvula mucho))) </pre>	<pre> (defrule temperatura_buena   (Temp templado)   =&gt;   (assert (valvula medio))) </pre>	<pre> (defrule hace_calor   (Temp calor)   =&gt;   (assert (valvula poco))) </pre>
---	---	--

La regla del comienzo del programa:

```

(defrule start
  (initial-fact)
  =>
  (printout t "Hola mundo!" crlf)
  (assert (Temp (10 0)(12 0.5)(15 1)(18 0.5)(23 0))))

(defrule print
  (declare (salience 1000)) ;prioridad alta
  (Temp ?t)
  =>
  ;;(plot-fuzzy-value t "" ?t)
  (plot-fuzzy-value t "+.#." 5 65 ?t
    (create-fuzzy-value Temp frio)
    (create-fuzzy-value Temp templado)
    (create-fuzzy-value Temp calor)))

```

Cuando hemos obtenido un conjunto difuso que define la apertura de la válvula, defuzzificamos para obtener un valor numérico (un ángulo) con el que girar la válvula, concretamente 66.85° como se puede observar en el nuevo hecho asertado.

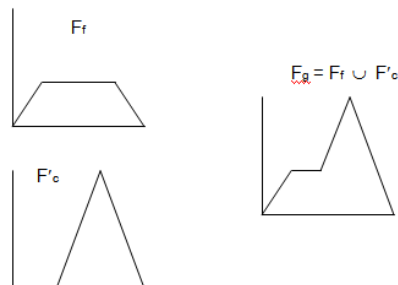
```
(defrule defuzzificar
  (valvula ?val)
  =>
  (plot-fuzzy-value t "." 0 90 ?val)
  (assert (crisp valvula (moment-defuzzify ?val))))
```

### Nota sobre duplicación de hechos difusos

En Clips estándar si se aserta un hecho que ya existe y no se permite duplicación de hechos, es como si no se asertara realmente. De esta forma, las reglas no se vuelven ejecutar sobre un mismo hecho.

Sin embargo, en un sistema difuso, el refinamiento de un hecho difuso puede ser posible:

- a) Si se aserta un nuevo valor difuso a un slot, distinto al existente, se combinan ambos valores considerando una combinación OR ( $F_g = F_f \cup F'_c$ ):



Por ejemplo, si asertamos

```
(assert (Temp frio))
(assert (Temp templado))
```

Se supone que se añade más información, y resulta el hecho: (Temp [frio] OR [templado])

- b) De esta forma, una regla previamente ejecutada sobre este hecho, volverá a ejecutarse con la nueva información.

Por ejemplo, si se aserta “(assert (Temp frio))” y se lanza la regla (hace\_frio), y posteriormente, se aserta “(assert (Temp templado))”, se puede ejecutar de nuevo la regla, con la nueva información (en la medida en que (Temp templado) satisfaga la premisa de la regla (Temp frio).

- c) Si se aserta un hecho estructurado difuso igual a uno que ya existe, se considera un nuevo hecho difuso, por lo que una regla previamente aplicada volverá a aplicarse. Esto puede dar lugar a un **bucle infinito**. Por ello, esto debe controlarse mediante la aserción de algún slot que impida ejecutarse la regla (ver ejemplos finales).

## 5.- Defusificación

Para realizar la defusificación de un conjunto difuso a un valor numérico, se dispone de dos funciones:

(moment-defuzzify ?fuzzy-value | ?variable-fuzzy | integer), que aplica el algoritmo del centro de gravedad.

(maximun-defuzzify ?fuzzy-value | ?variable-fuzzy | integer), que aplica la media de máximos.

El valor integer corresponde a la identificación de un valor en la base de hechos (identificado en la parte izquierda de una regla).

---

## Ejemplo

Declaramos la variable difusa y las reglas:

```
(deftemplate edad 0 100 años
  ( (joven (10 0) (15 1) (25 1) (30 0))
    (adulta (20 0) (30 0.7) (40 1) (60 0.7) (70 0))
    (madura (60 0) (70 1))))

(defrule fuzzy1
  ?f <- (edad ?)
  => (bind ?e (maximum-defuzzify ?f))
  (printout t "En fuzzy1, edad es " ?e crlf))

(defrule fuzzy2
  ?f <- (edad ?)
  => (bind ?e (moment-defuzzify ?f))
  (printout t " En fuzzy2, edad es " ?e crlf))
```

Asertamos el hecho: (assert (edad adulta)), y tras lanzar la ejecución (run), obtenemos:

```
FuzzyClips> (run)
FuzzyClips> En fuzzy1, edad es 40.0
FuzzyClips> En fuzzy2, edad es 44.76923076923077
```

Debe notarse que, si por el contrario hubiéramos especificado: (assert (edad not adulta)), obtendríamos de acuerdo a los métodos de defusificación:

```
FuzzyClips> (reset)
FuzzyClips> (assert (edad not adulta))
FuzzyClips> (run)
FuzzyClips> En fuzzy1, edad es 47.5
FuzzyClips> En fuzzy2, edad es 52.51851851852
```

También:

```
FuzzyClips> (reset)
FuzzyClips> (assert (edad madura))
FuzzyClips> (run)
FuzzyClips> En fuzzy1, edad es 85.0
FuzzyClips> En fuzzy2, edad es 82.38095238095238
```

## 6.- Ejemplos

### EJEMPLO-1.

Definamos un sistema que caracteriza la aptitud de una persona para jugar al baloncesto en función de su edad, su salud y su estatura.

```
(deftemplate edad 0 100 años ; definición de la variable fuzzy edad
  ( (joven (10 0) (15 1) (25 1) (30 0))
    (adulta (20 0) (30 1) (60 1) (70 0))
    (madura (60 0) (70 1))))

(deftemplate estatura 0 250 cm ; definición de la variable fuzzy estatura
  ((bajo (0 1) (100 1) (150 0))
    (medio (100 0) (150 1) (170 1) (180 0))
    (alto (170 0) (180 1))))

(deftemplate aptitud 0 10 unidades ; aptitud para jugar al baloncesto
  ((baja (0 1) (5 0))
    (media (3 0) (4 1) (6 1) (10 0))
    (alta (5 0) (10 1))))
```

---

Definimos una regla de clasificación y una para defusificar

```
(defrule ejemplo ;asignar_aptitud_jugar_baloncesto
  (edad joven)
  (estatura alto)
  =>
  (assert (aptitud alta)))

(defrule defusificar
  ?f <- (aptitud ?)
  => (bind ?e (maximum-defuzzify ?f))
      (printout t "Aptitud es " ?e crlf))
```

Ahora creamos una persona de ejemplo:

```
(deffacts fuzzy-fact
  (edad adulta)
  (estatura very alto))
```

Si ejecutamos, obtendremos:

```
FuzzyClisp> Aptitud es 9.1666
```

Nótese que el nuevo hecho obtenido (ventana Facts) es: (aptitud ???), que está indicando que su aptitud es un valor difuso no previamente definido. Este nuevo valor difuso (en la medida en que hacen más o menos matching las premisas) es obtenido mediante la inferencia de los datos. Su defusificación obtiene el valor 9.16.

Nótese también que FuzzyClips representa, en los hechos obtenidos, la función de pertenencia del valor difuso inferido.

## **EJEMPLO-2.**

Similar al ejemplo anterior utilizando templates.

```
(deftemplate edad 0 100 años ; definición de la variable fuzzy "edad"
  ( (joven (10 0) (15 1) (25 1) (30 0))
    (adulta (20 0) (30 1) (60 1) (70 0))
    (madura (60 0) (70 1))))

(deftemplate estatura 0 250 cm
  ((bajo (0 1) (100 1) (150 0))
   (medio (100 0) (150 1) (170 1) (180 0))
   (alto (170 0) (180 1))))

(deftemplate aptitud 0 10 unidades
  ((baja (0 1) (5 0))
   (media (3 0) (4 1) (6 1) (10 0))
   (alta (5 0) (10 1))))

(deftemplate persona
  (slot nombre (type SYMBOL))
  (slot edad (type FUZZY-VALUE edad))
  (slot estatura (type FUZZY-VALUE estatura))
  (slot aptitud (type FUZZY-VALUE aptitud) )
  (slot evaluado (type SYMBOL)))

(deffacts fuzzy-fact
  (persona (nombre adan) (edad adulta) (estatura alto) (aptitud baja) (evaluado no))
  (persona (nombre eva) (edad joven) (estatura medio) (aptitud baja) (evaluado no))
  (persona (nombre moises) (edad madura) (estatura alto) (aptitud baja) (evaluado no))
  (persona (nombre david) (edad joven) (estatura alto) (aptitud baja)(evaluado no))
  (persona (nombre samuel) (edad not joven) (estatura more-or-less alto ) (aptitud
baja)(evaluado no))
  (persona (nombre salome) (edad very joven) (estatura extremely alto ) (aptitud
baja)(evaluado no)))
```

---

```

(defrule ejemplo ;asignar_aptitud_jugar_baloncesto
  ?f <- (persona (nombre ?n) (edad joven) (estatura alto) (evaluado no) )
=>
  (retract ?f)
  (assert (persona (nombre ?n) (edad joven) (estatura alto) (aptitud alta) (evaluado si))))

(defrule defusificar
  (persona (nombre ?nombre) (edad joven) (estatura alto) (aptitud ?f) (evaluado si))
=> (bind ?e (maximum-defuzzify ?f))
  (printout t "nombre " ?nombre " tiene una aptitud de " ?e crlf))

```

Podemos obtener:

```

FuzzyClisp> nombre salome tiene una aptitud de 10.0
FuzzyClisp> nombre samuel tiene una aptitud de 8.75
FuzzyClisp> nombre david tiene una aptitud de 10.0
FuzzyClisp> nombre eva tiene una aptitud de 8.75
FuzzyClisp> nombre adan tiene una aptitud de 9.166

```

Nótese que:

- Es necesario incluir un control mediante el slot 'evaluado' para evitar que la regla se lance sucesivamente sobre hechos duplicados de una instancia ya existente.
- La regla no se aplica sobre 'Moises' al no satisfacerse la premisa (los valores edad=madura y edad=joven no hacen matching). Por el contrario, la regla sí se aplica sobre 'Samuel' porque aun siendo "not joven" puede ser adulto.

---

## Evaluación de la Práctica

### OBJETIVOS

El objetivo de la práctica es modelar un problema difuso y aplicar las técnicas de razonamiento difuso utilizando FuzzyClips.

La **realización de esta práctica** consiste en resolver uno de los ejercicios propuestos seguidamente. Las propuestas de dominio son orientativas, **debiendo completarse** a partir del conocimiento descrito. Además, el desarrollo de la práctica está abierto a cualquier **aportación propia** del alumno, pudiendo extender e incorporar cualquier criterio/conocimiento que quiera añadir a fin de que el sistema incorpore más conocimiento. Estas aportaciones se valorarán positivamente en la nota del proyecto.

Se deberá entregar una **memoria** en la que se indique cómo se ha modelado el problema (con el código implementado), con una explicación de las variables difusas, reglas diseñadas, así como pruebas de su ejecución, conclusiones y crítica.

Existe un **plazo de entrega** para dicha memoria.

### Elegid UNA de las siguientes propuestas

#### Propuesta 1. Control termostato difuso

Implementar el controlador difuso de un ventilador, tal que controle la velocidad de un ventilador a partir de la temperatura y humedad de la habitación.

La tabla para determinar la **velocidad** en función de la temperatura y la humedad es:

Temperatura/Humedad	seco	húmedo	mojado
frío	medio	alto	alto
templado	bajo	medio	alto
caliente	medio	alto	alto

Definir adecuadamente las variables difusas: temperatura, humedad, y velocidad. Mediante un razonamiento difuso, obtener valores de velocidad para diferentes escenarios en la habitación. Se recuerda que no se pueden leer valores difusos por consola (ver punto 2.5). La definición de las funciones de pertenencia y el método de defusificación es elección del alumno.

#### Propuesta 2. Control bomba de calor

Controlar la temperatura de una bomba de calor mediante un sistema de control difuso. En función de la temperatura ambiente actual y la temperatura previamente medida, se deberá determinar la temperatura de la bomba de calor.

Se deben modelar las variables difusas: temperatura actual (baja, media o alta), temperatura anterior (baja, media o alta) y la temperatura de salida fijada por el controlador: máximo frío (MF), frío (F), apagado (Off), calor (C) y máximo calor (MC).

Obtener un valor de la temperatura de la bomba, dada por una inferencia difusa, en función de la variación de la temperatura ambiente (dependiente de la temperatura anterior y la actual).



Temperatura Actual /Temperatura Anterior	alta	media	baja
alta	MF	MF	MF
media	F	Off	C
baja	MC	MC	MC

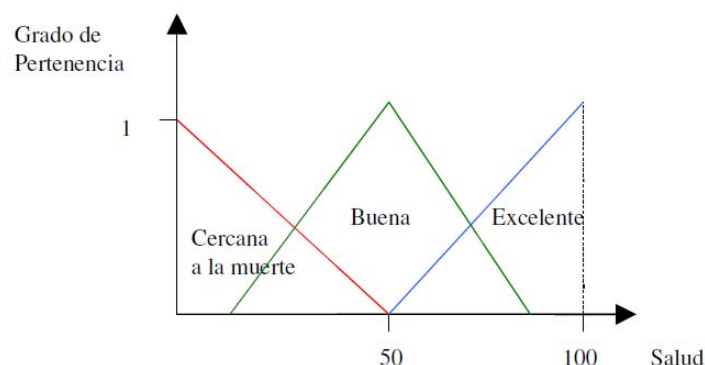


Se recuerda que no se pueden leer valores difusos por consola (ver punto 2.5). La definición de las funciones de pertenencia y el método de defusificación es elección del alumno.

### **Propuesta 3. Estrategia en juego de ataque**

El escenario a modelar consiste en una batalla donde, dependiendo de la **salud del jugador principal y la de su enemigo**, el jugador principal deberá adoptar una **estrategia** en el siguiente asalto. Es decir, el sistema pedirá a través de la consola la salud actual del jugador principal y luego la salud del enemigo, y devolverá, también a través de la consola, la estrategia que debe seguir el jugador actual para obtener en el siguiente asalto el mejor resultado posible.

La salud de los jugadores es un valor difuso, asumiendo que su universo está entre 0 (no tiene fuerza y está cercano a la muerte), y 100 (indicando que el jugador está en plenitud de fuerza). La siguiente imagen representa las funciones de pertenencia:



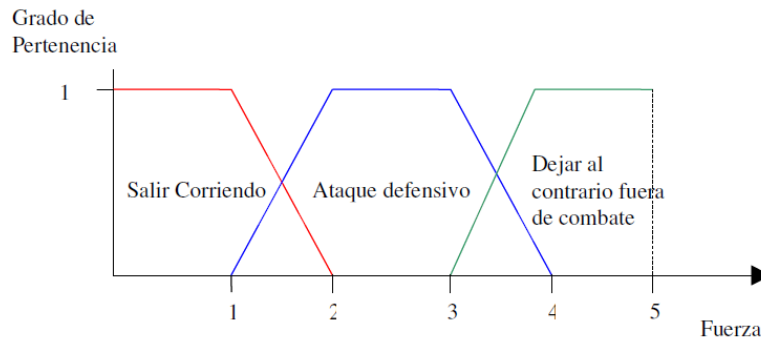
**NOTA:** la información de entrada podrá solicitarse de forma difusa (valores: cercana-a-la-muerte, buena o excelente) o como valores Crisp (rango: 0..100). En el segundo caso se recomienda utilizar la función para fusificar vista en el apartado 2.4.

Las estrategias que se plantean son: **Salir corriendo, Ataque defensivo y Dejar al contrario fuera de combate.**

Un experto en el juego nos indica que la forma de tomar las decisiones sobre la estrategia a seguir es:

- Cuando el jugador principal se encuentra cercano a la muerte y el enemigo tiene una salud buena o excelente la mejor estrategia que este jugador puede tomar, es salir corriendo.
- Cuando el enemigo se encuentra cercano a la muerte y el jugador principal tiene una salud buena o excelente, entonces se debe seguir la estrategia que deja al contrario fuera de combate.
- Esta estrategia también se puede seguir cuando el enemigo tiene una buena salud, pero la del jugador principal es una salud excelente.
- En cualquier otro caso se recomienda realizar ataques defensivos.

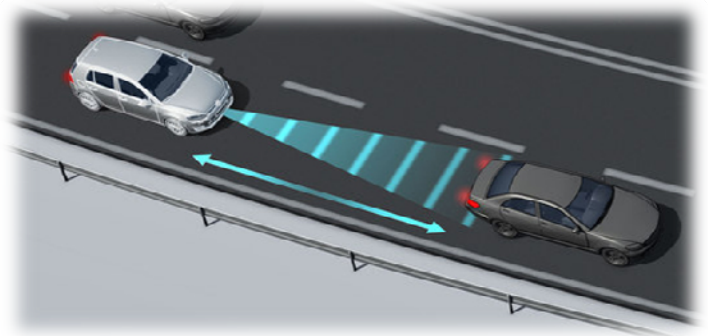
La estrategia a realizar implica la fuerza que se deba aplicar en la siguiente jugada. La fuerza tiene un valor entre 0 y 5, y se clasifica de la siguiente forma:



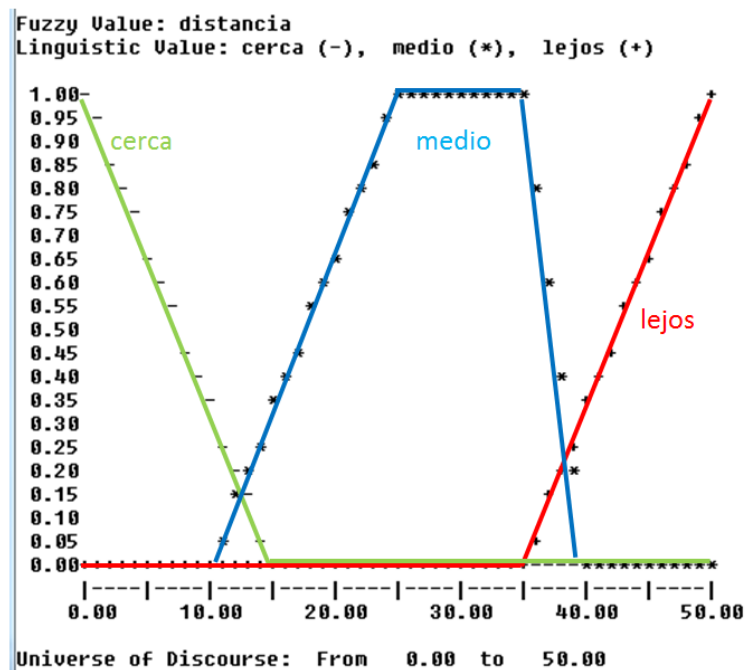
Implementar un sistema difuso que, dependiendo de la salud nuestra y la de nuestro enemigo, decida la estrategia a seguir (fuerza a aplicar) en el siguiente asalto.

#### Ejemplo 4. Control de cruceo de un vehículo

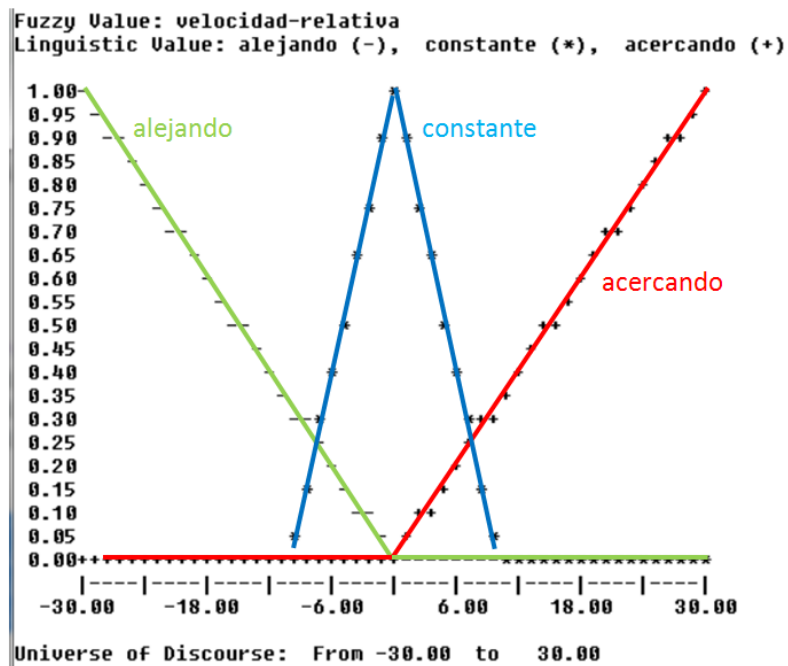
Queremos diseñar un sistema difuso para controlar la velocidad de cruceo de un vehículo con el objetivo de mantener una distancia de seguridad con el vehículo que le precede.



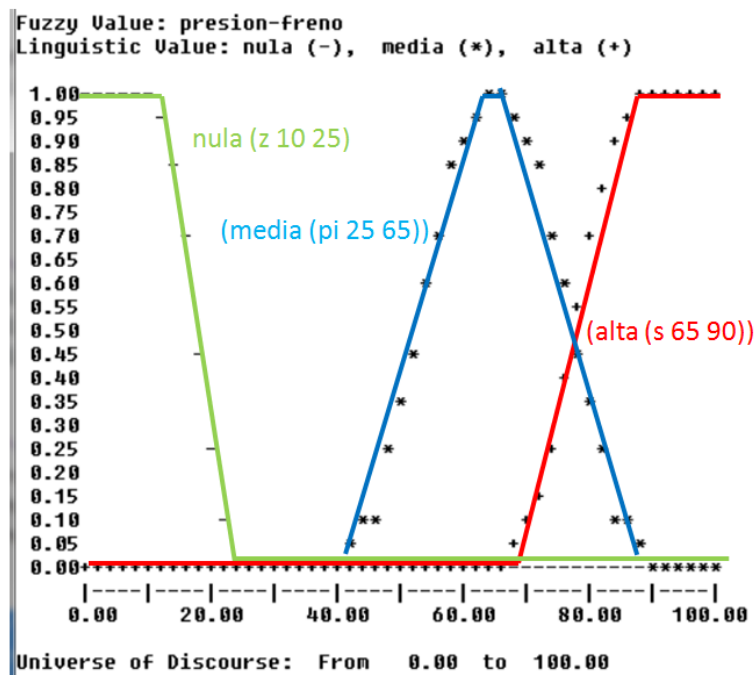
El vehículo dispone de dos sensores de entrada. El primero mide la distancia que separa el vehículo con el que le precede en un rango de 0 a 50 metros, asumiendo los siguientes valores: cerca, medio y lejos. El grado de pertenencia aproximado es:



El segundo sensor mide la velocidad relativa entre los dos coches, calculada en base a la diferencia de velocidades entre nuestro vehículo y el de delante (en un rango -30..30 km/h). Es decir, si es un valor cercano a -30 km/h quiere decir que nuestro vehículo se está alejando del de delante. Si la velocidad relativa es cercana a 0 km/h estamos manteniendo una distancia constante. Si, por el contrario, la velocidad relativa es cercana a 30 km/h quiere decir que nos estamos acercando al vehículo de delante. El grado de pertenencia aproximado es:



En función de la distancia y velocidad relativa, el vehículo debe aplicar una determinada presión de freno (nula, media o alta) en un valor de 0-100%. El grado de pertenencia aproximado es:



Los ingenieros han analizado distintas configuraciones y han definido el siguiente mapa de frenado automático en función de la distancia y velocidad relativa:

Distancia vs. Velocidad relativa	Alejando	Constante	Acercando
Cerca	Nula	Media	Alta
Medio	Nula	Nula	Media
Lejos	Nula	Nula	Media

---

Se pide obtener el valor de presión de freno en porcentaje, dado por una inferencia difusa, en función de los valores de la distancia y velocidad relativa medidos en metros y km/h, respectivamente. Para ello será necesario fusificar dichos valores Crisp de entrada utilizando la función fuzzify dada en el apartado 2.4. Se recuerda que no se pueden leer valores difusos por consola (ver punto 2.5). La elección del método de defusificación es elección del alumno.

### **Otras Propuestas**

A partir de los ejemplos vistos en clase, diseñar un sistema difuso que obtenga resultados similares. Por ejemplo: péndulo invertido, duración de un coche sin averías en función de su precio, etc.