



**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
1.1.- Modificadores Lingüísticos	3
2.- Hechos Difusos.	4
2.1.- Declaración de hechos iniciales: Deffacts	4
2.2.- Declaración de hechos difusos mediante expresiones ASSERT.	4
2.3.- Fusificación de valores Crisp	6
2.4.- Lectura de valores Crisp o valores difusos por consola.....	6
3.- Reglas difusas.	7
4.- Inferencia difusa.	9
5.- Defusificación.	10
Anexo. Hechos estructurados. Templates.	12

FuzzyCLIPS

FuzzyClips es la extensión fuzzy de CLIPS para manipular hechos y reglas difusas, ofreciendo el soporte para el desarrollo de sistemas difusos en el entorno de CLIPS. 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 Poliformat.

La **instalación de FuzzyClips** 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.

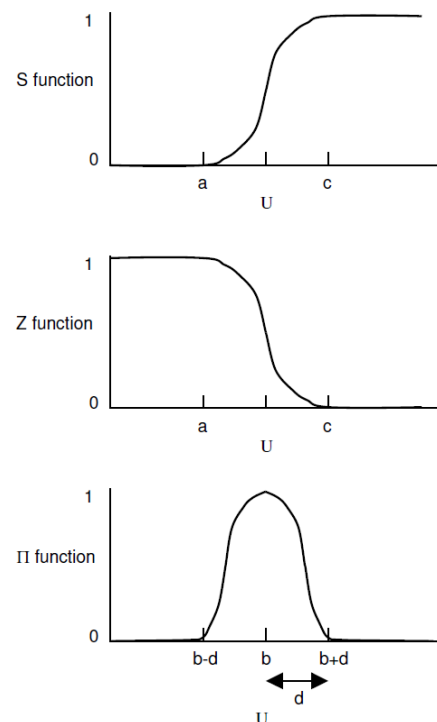
Las **variables difusas** se crean a partir de plantillas (**templates**), donde también se define su **Universo**, sus **valores difusos** (o valores lingüísticos) mediante las correspondientes **funciones de pertenencia**.

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.

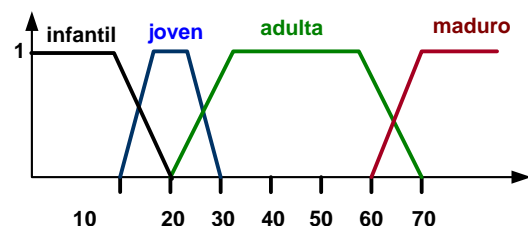
Las funciones de pertenencia se pueden definir de dos maneras diferentes:

- a) 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.
- b) Usando unas **funciones ya predefinidas**: s, z, pi (Más información en el manual).



Ejemplos:

```
(deftemplate edad ;Variable difusa
  0 120 años ;Universo
  ( (infantil (12 1) (20 0)) ;Valores difusos
    (joven (10 0) (15 1) (25 1) (30 0))
    (adulta (20 0) (30 1) (60 1) (70 0))
    (mayor (60 0) (70 1))))
```



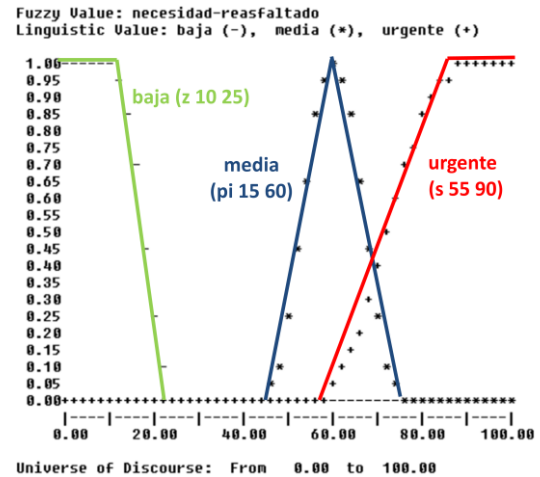
```
(deftemplate singleton
  0 10 unit
  ( (tres (3 0) (3 1) (3 0))
    (cinco (5 0) (5 1) (5 0))))
```



```
(deftemplate 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 temperatura 30 50 grados
  ( (bajo (35 1) (37 0))
    (medio (35 0) (36 1) (37 0))
    (alto (36 0) (37 1))))
```

```
(deftemplate necesidad-reasfaltado 0 100 unidades
  ( (baja (z 10 25))
    (media (pi 15 60))
    (urgente (s 55 90))))
```



1.1.- Modificadores Lingüísticos

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

$$\text{very}(y) = y^2$$

$$\text{more-or-less}(y) = \sqrt{y}$$

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	$y^{1.25}$

Por ejemplo, podemos tener:

```
(deftemplate estatura 0 250 cm
  ( (bajo (0 1) (100 1) (150 0))
    (muybajito extremely bajo)
    (medio (100 0) (150 1) (170 1) (180 0))
    (alto (170 0) (180 1))
    (muy-alto very alto)))
```

```
(deftemplate temperatura 0 80 grados
  ( (frio (z 10 25))
    (congelado plus frio)))
```

Los operadores lingüísticos también se pueden combinar con expresiones lógicas:

```
(deftemplate temperatura 0 80 grados
  ((frio (z 10 25))
   (calor (s 30 40))
   (templado not [ frio or calor ]))) ;hay que dejar un espacio después/antes de "[ ]"!!
```

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-difusa),

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.

a) Estos hechos pueden ser asertados mediante expresiones assert:

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

b) O inicializados mediante expresiones deffacts:

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

2.1.- Declaración de hechos iniciales: Deffacts

La expresión **deffacts** permiten declarar un conjunto de hechos iniciales crisp y fuzzy que se inicializarían con '(reset)'. Por ejemplo:

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

Podemos, posteriormente, añadir nuevos hechos mediante expresiones assert.

2.2.- Declaración de hechos difusos mediante expresiones ASSERT.

Se pueden **asertar nuevos hechos difusos** mediante expresiones **assert**:

(assert (<crisp-fact> |
 fuzzy-variable-name <description of fuzzy set>))

donde <description of fuzzy set> puede ser un <valor difuso> definido para la variable, 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 edad 0 100 años ((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))))
--	---

Podemos declarar **hechos difusos** de la forma:

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

Asimismo, asumida definida la variable difusa:

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

IMPORTANTE

No se deben declarar o asertar hechos **no estructurados** que contengan hechos difusos o mezclados con hechos Crisp.

Por ejemplo, si asertamos:

```
(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.

2.3.- 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 a variables difusas. 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-difusa 35 0.1)` y asertará el valor fusificado en la variable difusa *edad* mediante la expresión:

```
(assert-string "(edad-difusa (34.9 0) (35 1) (35.1 0))").
```

El rango de la función de pertenencia del valor asertado se establece en la llamada a la función. Particularmente, si queremos definir un valor *singleton*, será tan sencillo como invocar “(fuzzify edad 35 0)”.

Nota: Fuzzy-clips puede dar problemas si el valor crisp coincide con alguno de los extremos del intervalo definido para la variable difusa.

2.4.- Lectura de valores Crisp o valores 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 que sean leídos directamente desde consola**. Los valores difusos necesitan ser explícitamente asertados mediante expresiones assert o deffacts.

Por ejemplo, si tenemos definidas las variables difusas *edad* y *estatura*:

(deftemplate edad 0 100 años	(deftemplate estatura 0 250 cm
(joven (10 0) (15 1) (25 1) (30 0))	(bajo (0 1) (100 1) (150 0))
(adulto (20 0) (30 1) (60 1) (70 0))	(medio (100 0) (150 1) (170 1) (180 0))
(madura (60 0) (70 1)))	(alto (170 0) (180 1)))

Podríamos asertar desde consola: (assert (edad adulto)). Pero no podríamos asertar un valor ‘adulto’ leyendo dicho valor en consola. Para poder introducir valores difusos, a partir de valores crisp o de símbolos (por ejemplo: alto, bajos, etc.) leídos por consola podemos utilizar la función *assert-string*.

➤ **Lectura de símbolos (como valores fuzzy):**

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

```
(defrule leerconsola ;lee de consola un valor difuso y lo aserta
  (initial-fact)
=>
  (printout t "Introduzca la edad: joven, adulta, madura" crlf)
  (bind ?Redad (read))
  (assert-string (format nil "(edad %s)" ?Redad)) )
```

Esta regla también podría codificarse como una función para lectura de valores y su aserción fuzzy.

➤ **Lectura de valor crisp y aserción de su valor fusificado:**

Podemos usar la función **fuzzify** para asertar un valor difuso en función de un valor crisp leído:

```
(defrule leerconsola ;lee de consola un valor crisp y aserta su valor fusificado
  (initial-fact)
=>
  (printout t "Introduzca la edad en anyos" crlf)
  (bind ?Redad (read))
  (fuzzify edad ?Redad 0.1))
```

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

```
(defrule leerconsola ;lee de consola valor crisp y aserta su valor fuzzy singleton
  (initial-fact)
=>
  (printout t "Introduzca la edad en anyos" 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>)

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: (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.

Notas importantes:

- No es posible utilizar en una regla la función ‘test’ con valores difusos. Es decir:

```
(defrule example
  (tanque ?h)
  (test (eq ?h alto))
```

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:

Función	Devuelve
(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 2.3 o definir el valor Crisp como un valor difuso (de tipo singleton en la propia definición de la variable difusa).

EJEMPLO-2:

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: (deffacts ejemplo (edad veinticinco))

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 2.3. 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.

SALIDA: APERTURA DE LA VALVULA

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

```
(deftemplate valvula 0 90 grados-apertura
  ((poco (z 10 30))
   (medio (pi 30 45))
   (mucho (s 70 80))))
```

Definimos las reglas:

```
(defrule hace_frio
  (Temp frio)
  =>
  (assert (valvula mucho)))
```

```
(defrule temperatura_buena
  (Temp templado)
  =>
  (assert (valvula medio)))
```

```
(defrule hace_calor
  (Temp calor)
  =>
  (assert (valvula poco)))
```

Tras los hechos iniciales:

```
(deffacts ejemplo
  (Temp very templado))
```

Lanzamos la ejecución (run), lo que asertará los valores correspondientes a la variable difusa válvula.

```
f-0      {initial-fact} CF 1.00
f-1      {Temp very templado} CF 1.00
f-2      {valvula medio} CF 1.00 1.25 0.01563 (21.56 0.04785) (21.88 0.09766)
```

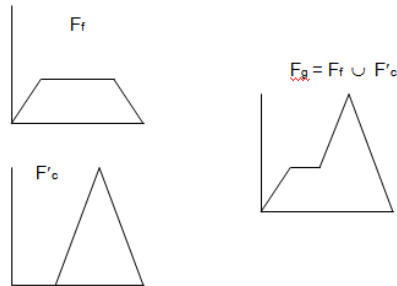
Podríamos utilizar una regla para defusificar dicho valor, y obtener un valor numérico (un ángulo) con el que girar la válvula (ver punto sobre defusificación):

```
(defrule defuzzificar
  (valvula ?val)
  =>
  (printout T "Apertura Valvula por moment: " (moment-defuzzify ?val) crlf)
  (printout T "Apertura Valvula por maximum: " (maximum-defuzzify ?val) crlf))
```

Nota sobre duplicación de hechos difusos

En Clips estándar si se aserta un hecho que ya existe no se permite duplicación de hechos. 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 una variable, 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 misma regla, con la nueva información, en la medida en que `(Temp templado)` satisfaga la premisa de la regla `(Temp frio)`.

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.

`(maximum-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-1:

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

(defrule fuzzy2
  (edad ?f)
  => (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
```

EJEMPLO-2:

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 supongamos los siguientes hechos iniciales:

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

Si ejecutamos, obtendremos:

```
FuzzyClips> 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.

Anexo. Hechos estructurados. Templates.

Fuzzy-Clips permite la declaración de clases o templates como hechos estructurados. Una template contiene slots, los cuales solo pueden tener valores crisp¹. Una template se define de la forma:

(deftemplate <nombre-template> <crisp-slot>+ *)

Los slots contienen valores crisp, del tipo que se indique:

<crisp-slot> ::= (slot <slot-name> [(type <tipo>)] [(default <valor>)])

donde **<tipo> ::= SYMBOL | INTEGER | STRING | ...**

Nota: si se declara de tipo entero, su valor por defecto es 0

Ejemplo: Podemos declarar el template:

```
(deftemplate persona
  (slot nombre (type SYMBOL))
  (slot altura (type INTEGER))
  (slot peso (type INTEGER))
  (slot seleccion (type SYMBOL) (default nil))
  (slot vive (type SYMBOL)))
```

Una vez declarado el template, los hechos iniciales se suelen declarar mediante la expresión deffacts:

```
(deffacts prueba
  (persona (nombre david) (altura 184) (vive Valencia))
  (persona (nombre juan) (altura 164) (peso 70) (vive Sevilla))
  (persona (nombre maria) (altura 174) (vive Madrid)))
```

O también realizar aserciones como: (assert (persona (nombre luis) (vive Valencia)))

Las reglas expresan patrones que deben instanciarse sobre los hechos.

```
(defrule selección
  ?f <- (persona (nombre ?n) (altura ?a) (vive ?v) (seleccion nil))
  (test (> ?a 175))
  =>
  (printout t "seleccionado " ?n crlf)
  (modify ?f (seleccion si)))
```

Para ver más ejemplos de reglas puede verse el ejemplo sobre jarras del Tema-1.

En fuzzy clips podemos mezclar hechos estructurados (solo con slots crisp) y hechos no estructurados (crisp o difusos).

En las reglas, podemos tener patrones sobre hechos estructurados y condiciones sobre valores de slots crisp, así como condiciones sobre valores de variables difusas.

¹ Realmente sí se puede definir un slot difuso, pero Fuzzy Clips necesita que se inicialice siempre a un valor (teniendo poca utilidad práctica si representa una conclusión que debe calcularse mediante inferencia).

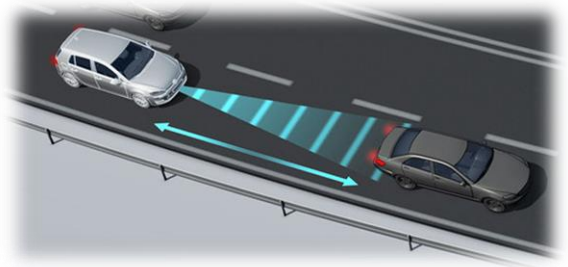
Evaluación de la Práctica

El objetivo de la práctica es aplicar el razonamiento difuso a un problema concreto utilizando FuzzyClips. Tras su desarrollo en la práctica, su evaluación consistirá en rediseñar y evaluar pequeñas modificaciones del problema propuesto.

Ejercicio. Control de crucero de un vehículo

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

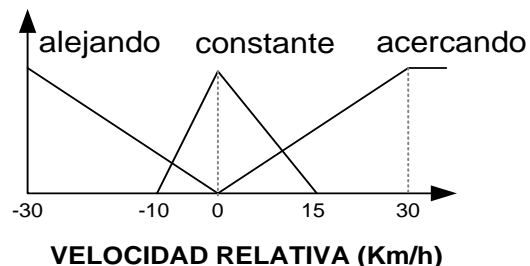
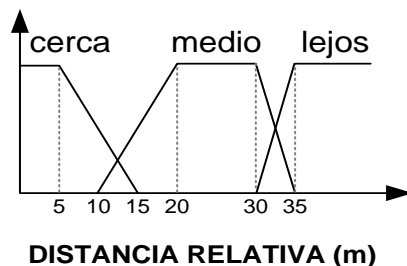
Con el control de crucero, el vehículo se adaptará automáticamente a la dinámica de una cola de vehículos, manteniendo una distancia de seguridad con el vehículo precedente, dependiente de su distancia y velocidad relativa, realizando suaves movimientos de aceleración o frenado.



El vehículo dispone de dos sensores de entrada:

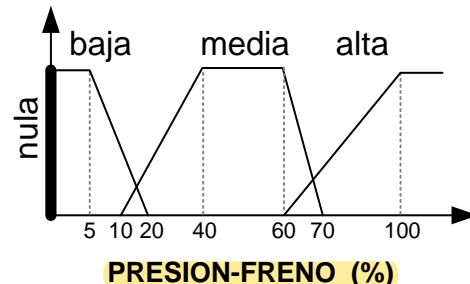
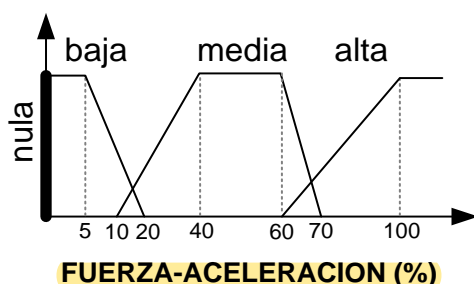
- El primer sensor mide la **distancia críp** que separa el vehículo con el que le precede en un rango de **0 a 50 metros**. La distancia relativa críp se fusifica con la variable difusa '**distancia**', con los valores difusos: '**cerca**', '**medio**' y '**lejos**'.
- El segundo sensor mide la **velocidad relativa entre los dos coches**, calculada en base a la diferencia de velocidades entre nuestro vehículo y el de delante, en un rango de **[-30, 30] km/h**. Es decir, si es un valor cercano a **-30 km/h** quiere decir que nuestro vehículo se está alejando del vehículo 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. La velocidad relativa críp se fusifica con la variable difusa '**velocidad**', que tiene los valores difusos '**alejando**', '**constante**', '**acercando**'.

Los grados de pertenencia de cada uno de estos valores (donde el universo de discurso es la distancia relativa en metros o la velocidad relativa en km/h) son:



En función de la distancia y velocidad relativas con el vehículo precedente, el vehículo a controlar debe **aumentar la velocidad (acelerar)** o **disminuir su velocidad (frenar)** para mantener una distancia de seguridad. Para acelerar, el sistema de control actúa sobre el motor del vehículo aplicando una **fuerza de aceleración {nula, baja, media, alta}**. Para frenar, actúa sobre el sistema de frenado aplicando una **presión de freno {nula, baja media, alta}**.

Los grados de pertenencia de estos valores (el universo de discurso es el porcentaje % de aceleración o frenado) son:



Los ingenieros de diseño han analizado distintas configuraciones y han definido la siguiente estrategia de control en función de la distancia y velocidad relativa con el vehículo precedente.

De esta forma, la “**fuerza de aceleración**” debe ser:

<i>Distancia-relativa vs. Velocidad-relativa</i>	Alejando	Constante	Acercando
Cerca	<i>media</i>	<i>nula</i>	<i>nula</i>
Medio	<i>more-or-less alta</i>	<i>media</i>	<i>nula</i>
Lejos	<i>very alta</i>	<i>alta</i>	<i>very baja</i>

Y la “**presión de freno**”:

<i>Distancia-relativa vs. Velocidad-relativa</i>	Alejando	Constante	Acercando
Cerca	<i>nula</i>	<i>very baja</i>	<i>alta</i>
Medio	<i>nula</i>	<i>nula</i>	<i>more-or-less media</i>
Lejos	<i>nula</i>	<i>nula</i>	<i>very-baja</i>

A partir de esta información, en el sistema Fuzzy Clips a desarrollar se definirá una **clase de vehículos** y una **instancia de vehículo** que contendrá la siguiente información no difusa:

- identificador del vehículo (symbol),
- distancia-crisp al vehículo de delante (Integer, en metros),
- velocidad-crisp relativa (Integer, km/h),
- fuerza de aceleración crisp a aplicar (Float, porcentaje %), valor decodificado por momentum
- fuerza de aceleración crisp a aplicar (Float, porcentaje %), valor decodificado por maximum
- presión de freno crisp a aplicar (Float, porcentaje %), valor decodificado por momentum
- presión de freno crisp a aplicar (Float, porcentaje %), valor decodificado por maximum

Los valores de los **tres primeros datos** serán solicitados al usuario (simulando una entrada real desde los sensores), mientras que los **dos valores para fuerza de aceleración crisp y presión de freno crisp** son los resultados a obtener a partir de la defusificación de los resultados difusos de unas variables **fuerza-aceleración y presión-freno**, los cuales serán obtenidos a partir del razonamiento difuso dado por las tablas y calculado en base a los valores difusos de las variables difusas velocidad y distancia. Estos valores difusos serán obtenidos a partir de la fusificación de las variables de entrada distancia-crisp y velocidad-crisp.

En otras palabras,

- 1) El sistema adquiere (entrada por teclado) los valores crisp de la velocidad y distancia relativa de la instancia del vehículo.
- 2) Estos valores se fusifican en las variables difusas ‘velocidad’ y ‘distancia’.
- 3) El razonamiento difuso obtiene los valores difusos de las variables difusas ‘fuerza-aceleración’ y ‘presión-freno’.
- 4) Se defusifican las variables difusas ‘fuerza-aceleración’ y ‘presión-freno’ en los valores crisp correspondientes de la instancia del vehículo, por **momentum y maximum**. Este será el resultado final que actuará sobre el acelerador/freno.

Para fusificar los valores **crisp** se utilizará la **función fuzzify** dada en el apartado 2.3.

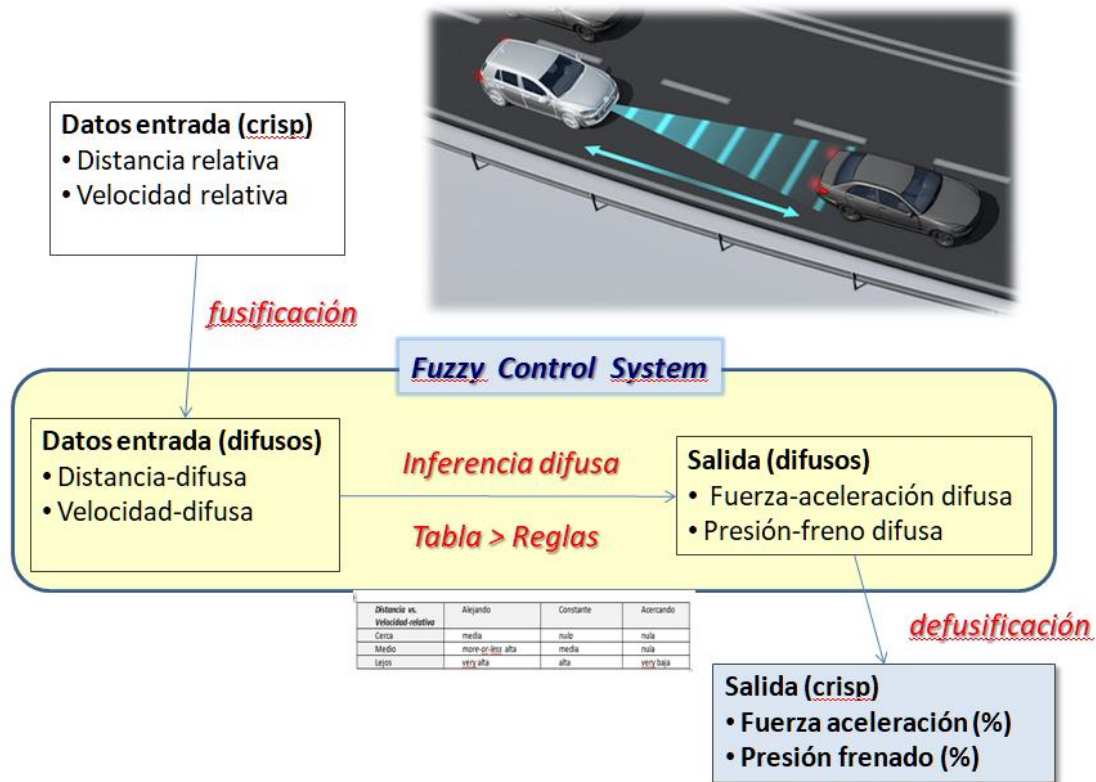
Para defusificar los valores difusos inferidos para la **fuerza de aceleración y presión de frenado**, se aplicarán los dos métodos ‘moment-defuzzify’ y ‘maximum-defuzzify’. Los resultados crisp se insertarán en los slots correspondientes de la instancia que representa el vehículo y que se aplicarían sobre el vehículo a controlar.

Una vez implementado el sistema difuso, se deberán evaluar diversos escenarios, analizando y comparando los resultados crisp obtenidos.

Notas Importantes:

- Dado que el esquema general de las reglas se repite, se recomienda diseñar una regla, probar su corrección y aplicarla en las restantes.

- Para **modificar** en una regla el valor de un slot de una instancia, instanciada en la parte izquierda, se puede utilizar el comando **modify** (ver Tema 1). Esto puede ser de utilidad en la aserción final de los valores crisp de aceleración y frenado (por ejemplo: (modify ?j2 (litros (+ ?lit2 ?lit1))))
- En el caso de un valor difuso de la forma (nulo (0 0) (1 0) (0 0)), su defusificación mediante 'maximum' obtendrá el valor crisp 0, pero por 'moment' obtendrá el valor crisp 50.
- En una aplicación práctica sobre el vehículo, los valores finales de aceleración o frenado se podrían obtener mediante la combinación aritmética de los porcentajes obtenidos



Como ejemplos de resultados, tenemos los siguientes valores, según la defusificación por momentum (mom) o máximo (max):

Velocidad (Km) Y Distancia (m) Relativas	-10 Km/h	0 Km/h	+ 25 Km/h
5 m	43.3 % aceleración max	0 % aceleración max	0 % aceleración max
	0 % frenado max	2.5 % frenado max	96.6% frenado max
	41.6 % aceleración mom	50 % aceleración mom	50 % aceleración mom
	50 % frenado mom	5.6 % frenado mom	86.3 % frenado mom
25 m	81.4 % aceleración max	50 % aceleración max	0 % aceleración max
	0 % frenado max	0 % frenado max	45.7 % frenado max
	80.7 % aceleración mom	44.2 % aceleración mom	50 % aceleración mom
	50 % frenado mom	50 % frenado mom	41.4 % frenado mom
32 m	81.4 % aceleración max	46 % aceleración max	0 % aceleración max
	0 % frenado max	0 % frenado max	42.1 % frenado max
	80.7 % aceleración mom	55.0 % aceleración mom	7 % aceleración mom
	50 % frenado mom	50 % frenado mom	36.7 % frenado mom