Práctica 2 : Razonamiento difuso

Axel Guzman Godia

Introducción

El objetivo de esta práctica consiste en implementar un sistema de lógica difusa. Para ello haremos uso de la herramienta Fuzzy-Clips. Se han propuesto distintas ideas, de entre estas he elegido implementar la **bomba de calor.**

Estructura del problema

Este problema trata de diseñar un sistema experto capaz de gobernar una bomba de calor mediante lógica difusa. Para ello el sistema se basa en la temperatura actual y en la temperatura medida en el instante anterior. A partir de estos datos se debe proporcionar una salida, que es la temperatura que aplicará la bomba de calor.

Para modelar todo esto he definido tres variables difusas. Dos de ella modelan la lectura de la temperatura (cada una en un instante distinto) y la otra la salida (la temperatura de la bomba de calor). A continuación se detallan los “templates” utilizados para definir dichas variables:

```
(deftemplate tmpActual 0 50 grados
        ( (baja (0 1) (13 1) (16 0) )
        (media (14 0) (19 1) (24 1) (29 0))
        (alta (27 0) (28 1))))

(deftemplate tmpAnterior 0 50 grados
( (baja (0 1) (13 1) (16 0) )
        (media (14 0) (19 1) (24 1) (29 0))
        (alta (27 0) (28 1))))
```

Para modelar la temperatura he elegido un universo que va desde los 0 hasta los 50 grados. Las dos variables de temperatura tienen las mismas particiones . Las particiones se pueden observar en la definición de las plantillas.

```
(deftemplate tmpSalida 0 30 grados
((off (0 0)(0 1)(0 0))
        (max_frio (13 0)(14 1) (15 1) (16 0))
        (frio (15 0) (16 1) (17 1) (19 0))
        (calor (19 0) (19 1) (22 1) (24 0))
        (max_calor(22 0)(24 1) )
        ))
```

Por último, para modelas la salida se ha tomado un universo que va desde 0 a 30 grados. No obstante se asume que la bomba de calor tiene un rango de funcionamiento que va desde 14 grados hasta 30. El universo se ha modelado hasta el 0 para poder incluir el estado apagado (Salida = 0. Cabe destacar pues, que **cuando el sistema genere la salida 0** no significara que la bomba sacará una temperatura de cero grados, sino que **la bomba está apagada.**

Además de las variables difusas, también se hace uso de dos plantillas más. Estas plantillas servirán para modelar un determinado momento.:

```
(deftemplate estado
        (slot hora )
        (slot Actual (type FUZZY-VALUE tmpActual))
        (slot Anterior (type FUZZY-VALUE tmpAnterior))
        (slot evaluado ))

(deftemplate estado_eval
        (slot hora )
        (slot Actual (type FUZZY-VALUE tmpActual))
        (slot Anterior (type FUZZY-VALUE tmpAnterior))
        (slot Salida (type FUZZY-VALUE tmpSalida) )
        (slot evaluado ))
```

Estas plantillas sirven para modelar cada uno de los momentos en que el sistema toma una decisión. Para este caso se ha asumido que el sistema evalúa la temperatura cada hora, aunque en un caso real la frecuencia sería mucho mayor. Resumiendo, cada estado contendrá información de la temperatura actual, la anterior y la hora. Una vez se hayan procesado estos datos se utilizara la plantilla estado_eval, que contendrá los mismos datos y información adicional sobre la salida que debe producir la bomba de calor.

Reglas y funcionamiento

Una vez hemos definido las variables difusas y las plantillas, queda modelar el comportamiento del sistema. Las reglas se han modelado para representar el comportamiento especificado en el boletín de prácticas. (Véase imagen)

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

Las siguientes reglas sirven para modelar esta información:

```
(defrule maxfrio
        ?f<-(estado (hora ?h) (Actual alta) (Anterior ?ant) (evaluado no))
=>
        (assert (estado_eval (hora ?h) (Actual alta) (Anterior ?ant) (Salida max_frio) (evaluado
si)))
        )


(defrule frio
        ?f<-(estado (hora ?h) (Actual media) (Anterior alta) (evaluado no))
=>
        (assert (estado_eval  (hora ?h) (Actual media) (Anterior alta) (Salida frio) (evaluado
si)))
        )


(defrule off

        ?f<-(estado (hora ?h) (Actual media) (Anterior media) (evaluado no))
=>
        (assert (estado_eval  (hora ?h) (Actual media) (Anterior media) (Salida off) (evaluado
si)))
        )


(defrule calor
```

```
        ?f<-(estado (hora ?h) (Actual media) (Anterior baja) (evaluado no))
=>

        (assert (estado_eval  (hora ?h) (Actual media) (Anterior baja) (Salida calor) (evaluado
si)))


        )


(defrule max_calor
        ?f<-(estado (hora ?h) (Actual baja) (Anterior ?t) (evaluado no))
=>
        (assert (estado_eval  (hora ?h) (Actual baja) (Anterior ?t) (Salida max_calor) (evaluado
si)))
        )
```

Se puede ver claramente como a partir de las entradas de temperatura actual y anterior se "asserta" la salida definida en la tabla. Todas estas reglas tienen una estructura similar.

Defusificación

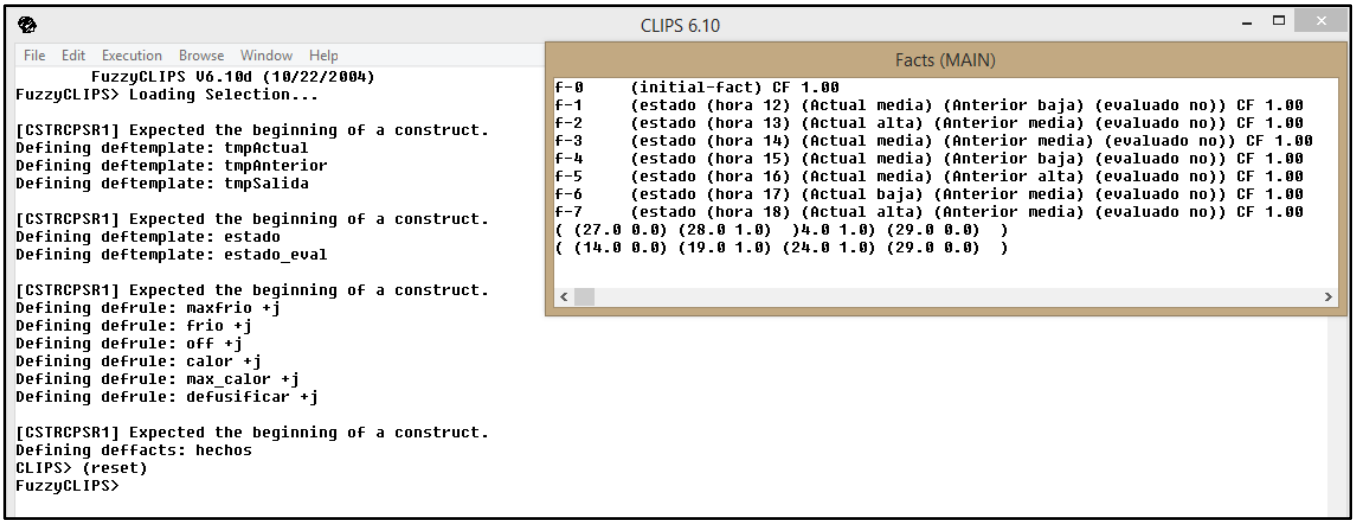
Por último, solo nos queda seleccionar un método de defusificación para transformar la salidas que produzca el sistema en valores concretos. Para esta propuesta he elegido el método de media de máximos (máximum-defuzzify). Tras realizar pruebas con los dos algoritmos que proporciona Clips, he comprobado que el método de centro de gravedad no ofrecía los resultados esperados . En concreto para el caso en que queremos que se produzca la salida off (0 grados) sólo se obtiene el resultado que queremos mediante el método de máximos. Esto se debe esencialmente a la forma en que proceden los algoritmos. Dicho esto, la regla que se usa para obtener valores concretos es la siguiente:

```
(defrule defusificar
        (declare (salience -2))

        ?f<-(estado_eval  (hora ?h) (Actual ?t3) (Anterior ?t) (Salida ?l) (evaluado si))

=>

        (bind ?e (maximum-defuzzify ?l))
        (if (!= ?e 0)
        then
                (if (< ?e 14)
                then (bind ?e 14))
                (if (> ?e 30)
                then (bind ?e 30)))
        (printout t "Para la hora " ?h " , la bomba se ajusta a:" ?e   crlf))
```

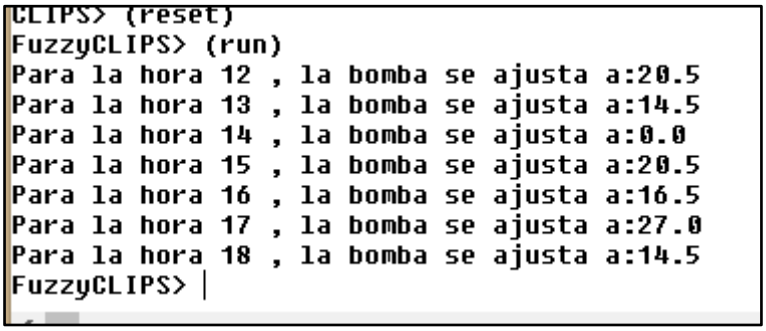
La sentencias if sirven para mantener los valores en el rango de funcionamiento de la bomba de calor.

Ejecución y resultados

Para comprobar la ejecución del sistema se han incluido una serie de estados como hechos declarados en una estructura deffacts. Se han incluido todas las instancias posibles para comprobar la ejecución. Los resultados son los siguientes:



En esta captura se muestra el estado antes de empezar la ejecución. Se ha ejecutado el comando reset y por lo tanto se han incorporado la base de hechos los estados definidos en deffacts. A continuación ejecutamos el comando run para obtener la salida:



Se muestra las distintas salidas que se han producido (expresadas en grados). Cabe destacar que la salida 0 significa que la bomba esta apagada.

Ampliación

Para mejorar el sistema de la bomba de calor se desea introducir un poco de interacción con el usuario. La motivación principal es que las personas percibimos la temperatura de manera relativa. Es decir a 15 grados una persona puede sentirse bien o tener frío.

Para conseguir esto, vamos a permitir que el usuario introduzca información de cómo se siente (Ej: si tiene un poco de frio el usuario esperara que el sistema produzca un poco más de calor de lo que está produciendo) . En este proyecto se va a usar una versión simplificada y se preguntará al usuario cada hora. Lo normal en un sistema real sería que el sistema funcionara independientemente de si el usuario solicita modificar la temperatura, y que esto pudiera

cambiar en cualquier instante (no a cada hora). No obstante no es objetivo de esta práctica diseñar esa parte del sistema.

Ejemplo del sistema: Tras analizar la temperatura actual y anterior, el sistema obtiene la salida 20 grados. Sin embargo el usuario está un poco acalorado. El usuario comunica al sistema que siente calor y este modifica la salida, cambiando a 18 grados.

Para modelar esto asumiremos que el usuario puede sentir : poco calor, calor , poco frio o frio. También asumimos que si está a gusto no necesitará modificar nada, por tanto no modelaremos este comportamiento.

Cada una de estas percepciones se asociará a una salida como sigue:

Poco calor -> frio calor ->mucho frio poco frio -> calor frio -> max_calor

La idea es que esta información se combine con la que obtiene el sistema para adaptar la salida. Esto se conseguirá mediante la aserción de un nuevo estado_eval. Este estado_eval se combinará con el que haya calculado el sistema, ya que si tenemos dos hechos difusos iguales se combinan automáticamente.

Para implementar esta funcionalidad se ha añadido una regla nueva, que es la siguiente:

```
(defrule Preguntar
        (declare (salience 3))
        ?f<-(estado  (hora ?h) (Actual ?t3) (Anterior ?t)  (evaluado no))
=>
        (printout t "Hora: " ?h " Como te sientes? (poco_calor, calor , poco_frio o frio)" crlf)
        (bind ?sent (read))
        (switch ?sent
                (case poco_calor then (assert (estado_eval  (hora ?h) (Actual ?t3) (Anterior ?t)
(Salida max_frio) (evaluado si))))
                (case calor then (assert (estado_eval  (hora ?h) (Actual ?t3) (Anterior ?t)
(Salida frio) (evaluado si))))
                (case poco_frio then (assert (estado_eval  (hora ?h) (Actual ?t3) (Anterior ?t)
(Salida calor) (evaluado si))))
                (case frio then (assert (estado_eval  (hora ?h) (Actual ?t3) (Anterior ?t)
(Salida max_calor) (evaluado si))))
                (default none)))
```

Esta regla modela la capacidad de preguntar al usuario. En caso de no obtener respuesta no se hará nada. La función switch sirve para relacionar las percepciones del usuario con la salidas. El efecto de la regla es asertar un estado_eval con una salida determinada en función de las necesidades del usuario. Al combinare esta salida con la que calcule el sistema, dará lugar a una leve modificación de la salida de forma que se adaptará a la situación. Veamos un ejemplo de ejecución:

```
CLIPS> (reset)
FuzzyCLIPS> (run)
Hora: 18 Como te sientes? (poco_calor, calor , poco_frio o frio)
frio
Hora: 17 Como te sientes? (poco_calor, calor , poco_frio o frio)
frio
Hora: 16 Como te sientes? (poco_calor, calor , poco_frio o frio)
calor
Hora: 15 Como te sientes? (poco_calor, calor , poco_frio o frio)
calor
Hora: 14 Como te sientes? (poco_calor, calor , poco_frio o frio)
calor
Hora: 13 Como te sientes? (poco_calor, calor , poco_frio o frio)
calor
Hora: 12 Como te sientes? (poco_calor, calor , poco_frio o frio)
calor
Para la hora 12 , la bomba se ajusta a:18.5
Para la hora 13 , la bomba se ajusta a:15.5
Para la hora 14 , la bomba se ajusta a:14
Para la hora 15 , la bomba se ajusta a:18.5
Para la hora 16 , la bomba se ajusta a:16.5
Para la hora 17 , la bomba se ajusta a:27.0
Para la hora 18 , la bomba se ajusta a:20.75
```

Comparemos este ejemplo con la salida de la versión sin interacción:

```
CLIPS> (reset)
FuzzyCLIPS> (run)
Para la hora 12 , la bomba se ajusta a:20.5
Para la hora 13 , la bomba se ajusta a:14.5
Para la hora 14 , la bomba se ajusta a:0.0
Para la hora 15 , la bomba se ajusta a:20.5
Para la hora 16 , la bomba se ajusta a:16.5
Para la hora 17 , la bomba se ajusta a:27.0
Para la hora 18 , la bomba se ajusta a:14.5
FuzzyCLIPS>
```

Se observa cómo han cambiado las salidas según el estado del usuario.

Conclusiones

Tras haber completado esta práctica las sensaciones son diversas. Primero destacar que el entorno de ejecución esta vez ha sido correcto, permitiendo visualizar la agenda , la base de hechos, etc... Esto ha permitido comprender mejor el funcionamiento y una depuración más eficaz. Segundo, trabajar con variables difusas tiene algunas pegas, como por ejemplo leer valores del teclado.

El resultado del sistema resulta correcto i ofrece valores de acuerdo a las entradas. El tiempo aproximado que he dedicado a esta práctica ha sido de aproximadamente unas 20 horas. Este tiempo se ha visto alargado principalmente por los problemas de trabajar con variables difusas, más que por la dificultad de diseñar el sistema.