TRABAJO TEMA 2 -

ESTUDIO DE LOS LENGUAJES DE LÓGICA DIFUSA FASILL Y BPL

Samuel Hidalgo Berlanga
Paula Cuadra Ruiz
Sergio Camacho Marín

Índice:

Estudio comparativo entre FASILL y Bousi-Prolog

FASILL y Bousi-Prolog son ambos entornos de desarrollo para la programación de lógica difusa. La programación de lógica difusa aparece como recurso auxiliar a la programación lógica, la cual es muy estricta y rígida , siendo así complicado automatizar procesos relacionados con casos reales cotidianos.

En el lenguaje natural, a diferencia de en la lógica clásica, los distintos elementos no tienen porqué tener resultados independientes entre sí. Puede haber un amplio espectro de elementos que se relacionen con un único significado. La lógica difusa se encarga de solucionar este problema unificando elementos mediante relaciones.

FASILL, una breve introducción

FASILL es un lenguaje de lógica difusa totalmente integrado que trabaja con anotaciones graduales, una inmensa variedad de conectivos y unificadores por similitud. Se define con la tupla $< \Pi, L, R >$, donde Π es el conjunto de las normas que se van a utilizar en el programa, L es el retículo de grados de verdad que va más allá de verdadero y falso, y R es el conjunto de relaciones de similitud. Hace uso del algoritmo de unificación débil, el cual se encarga de establecerlas relaciones en el retículo formado por elementos y funciones con los mismos.

Para poder explicar con mayor claridad cómo es capaz de adaptar un problema el entorno FASILL, hemos desarrollado un ejemplo práctico usando la herramienta online de la siguiente página web: https://dectau.uclm.es/fasill/sandbox.

Conjunto de normas Π:

```
velocidad(i-future) <- 0.4.
precio(typhone) <- 0.7.
atencion_cliente(i-future) <- 0.7.
promociones(typhone) <- 0.3.
inf_disponible(typhone, fibra) <- 0.8.
inf_disponible(i-future, lin_telef) <- 0.7.
buena_compania(X) <- @aver(inf_disponible(X,fibra), atencion_cliente(X), @div(velocidad(X), @dif(precio(X)))).
```

En este conjunto se definirán los datos de nuestro programa así como la función que queramos usar como salida. Para este ejemplo hemos decidido crear un programa que compare distintas compañías telefónicas para comprobar cual de ellas es la mejor.

Hemos colocado los datos de 2 compañías inventadas, *i-future* y *typhone* y hemos creado una función resultado que dirá cual de estas 2 es la mejor compañía.

La función resultado realiza una media entre la calidad del servicio de fibra óptica, el grado de atención al cliente y la velocidad de conexión partido el precio que ofrecen.

Retículo completo L:

En esta sección del programa se definirán las distintas reglas y funciones que se usarán en el resto del programa.
Podemos dividirlo en varias partes:

- La primera parte se encarga de definir los distintos valores que vamos a utilizar en el esquema. En ella declaramos entre otras cosas que elementos pueden aparecer cual es el máximo y mínimo al que pueden llegar y el criterio de ordenación que se va a utilizar.

```
% Elements
member(X) :- number(X), 0 =< X, X =< 1.
members([0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]).

% Distance
distance(X,Y,Z) :- Z is abs(Y-X).

% Ordering relation
leq(X,Y) :- X =< Y.

% Supremum and infimum
bot(0.0).
top(1.0).
supremum(X, Y, Z) :- Z is max(X, Y).
```

- La segunda parte se encarga de definir las distintas t-normas que se van a utilizar para resolver las relaciones. El modelo inicial del desarrollador web ya tiene introducidas las 3 normas más usadas en lógica difusa: la t norma de Łukasiewicz, la t norma del mínimo (ó de Gödel–Dummett) y la t norma del producto.

```
% Binary operations
and_prod(X,Y,Z) :- Z is X*Y.
and_godel(X,Y,Z) :- Z is min(X,Y).
and_luka(X,Y,Z) :- Z is max(X+Y-1,0).
or_prod(X,Y,Z) :- U1 is X*Y, U2 is X+Y, Z is U2-U1.
or_godel(X,Y,Z) :- Z is max(X,Y).
or_luka(X,Y,Z) :- Z is min(X+Y,1).
```

- A continuación se describen los agregadores (los cuales fueron usados para la función resultado anteriormente).
  Los agregadores que hemos definido son:
  - **@aver** : función con 3 valores de entrada. Devuelve el resultado de la media de 3 elementos.
  - **@dif** : función de una entrada. Devuelve el resultado de restarle a 1 el valor de entrada.

- **@div** : función de 2 entradas. Devuelve el resultado de la división de 2 elementos.

```
% Aggregators
agr_aver(X,Y,Z,A) :- A is (X+Y+Z)/3.
agr_dif(X,Y) :- Y is 1- X.
agr_div(X,Y,Z) :- Z is min(X/Y,1).
```

- Por último se define cual va a ser la t-norma que se usará por defecto. En este caso hemos decidido usar la norma del producto.

```
% Default connectives
tnorm(prod).
tconorm(prod).
```

Relaciones de similaridad R:

Tal y como se mencionó al comienzo del documento, la lógica difusa busca que los elementos lógicos no sean independientes unos de otros y que, por lo tanto se pueda conseguir más información a partir de un único dato conocido.

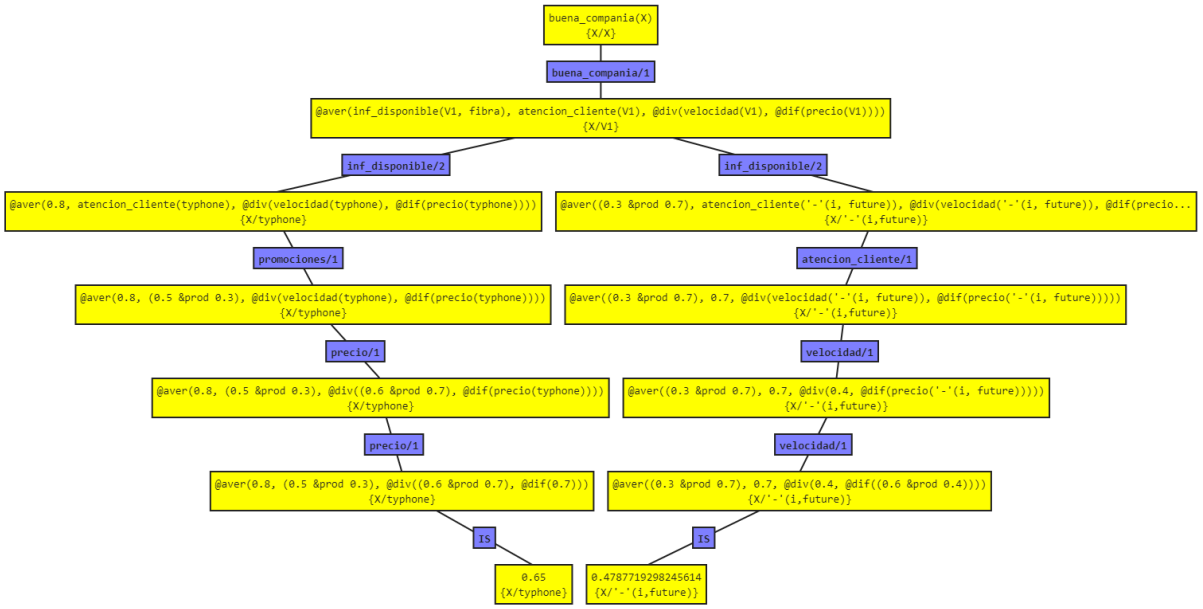
```
precio/1 ~ velocidad/1 = 0.6.
promociones/1 ~ atencion_cliente/1 = 0.5.
fibra ~ adsl = 0.5.
adsl ~ lin_telef = 0.6.
~tnorm = prod .
```

Valores de corte :

FASSIL permite seleccionar un valor de corte lambda. Este valor nos permite saber cuando una función no se cumple. Cualquier valor de la función resultado que se encuentre por encima del corte será considerado verdadero, y por consiguiente cualquier valor por debajo del mismo será considerado falso.

Árbol de derivación:

La herramienta web de FASILL cuenta además con una función para visualizar el árbol derivación, el cual muestra el proceso que han seguido los datos del programa hasta llegar a una solución. Esto resulta muy útil para comprender el funcionamiento del lenguaje.



Bousi~Prolog

Breve introducción

Es un lenguaje de programación de lógica difusa que utiliza un algoritmo de unificación difusa. Este se guía por grados de unificación, esto quiere decir que representa el grado de verdad asociado con la instancia calculada. Este es su núcleo de semántica operativa BPL(Relaciones difusas binarias reflexivas y simétricas, no necesariamente transitiva).

Versiones

A partir de la versión 3.5 se añadió las reglas graduadas como son las T-normas y aplicarlas para proporcionar un mayor grado de aproximación. Estas son: producto, luka , drástico, nilpotente y hamacher.

A partir de la versión 3.6, se puede empezar a trabajar con gramáticas de cláusulas definidas de forma nativa. Por otro lado, se implementaron nuevas técnicas que permiten optimización de recursividad de cola y recuperar aspectos de indexación de alto nivel.

Sintaxis

Se diferencia 3 tipos de relaciones dentro de Bousi-Prolog, son las siguientes:

- Relación de proximidad o similitud(~)
- Relaciones de orden parcial(~> <~)
- Relaciones configurables(~1~ ~2~ ~3~)

Todos estos no dejan de ser operadores que se utilizan para crear relaciones.

Relación de proximidad o similitud

La sintaxis BPL, utilizada por Bousi, es principalmente la sintaxis de Prolog pero con "~"(unificador débil, devuelve entre 0 y 1.0). Queda de la siguiente forma:

<símbolo> ~ <símbolo> = <grado de proximidad>

Esto sería una ecuación de proximidad, en la que el primer símbolo es aproximado en grado de proximidad al segundo símbolo. Sería una implementación de bajo nivel.Se pueden hacer cosas como lo siguiente:

Ejemplo sencillo

```
interesting(Novel) :- thriller(Novel).
adventure('The Treasure Island').
thriller ~ adventure = 0.5.
?- interesting('The Treasure Island').
```

Te dará que si, en un grado de aproximación de 0.5.

Funcionamiento interno

Bousi-Prolog está compuesto de 4 partes: el analizador, el motor de proximidad, el adaptador y el motor generador de código. Funciona de la siguiente manera, en el caso de que sea una relación borrosa:

1. Genera un cierre reflexivo y simétrico de la relación borrosa.
2. Luego se intenta unificar el objetivo con el actual.
3. Si existe una regla de proximidad entonces el proceso de unificación tiene un éxito y proporciona un grado de aproximación.

Gracias a BPL, se puede habilitar o deshabilitar el cierre transitivo de una relación difusa durante el proceso de compilación.

T-normas utilizadas

Por otro lado se puede especificar para la relación de transitividad la t-norma a utilizar(:-Transitivity<Option>), entre las que se puede usar destacan las siguientes:

- Si la transitividad está especificada como yes, pues utilizará la t-norma del mínimo.
- Si la transitividad está especificada como no, no hay transitividad.
- Si se especifica min, se utilizará la t-norma del mínimo.
- Si se especifica luka, se utilizará la t-norma de Łukasiewicz.
- Si se especifica product, se utilizará la t-norma del producto.

Esquema general de relaciones de proximidad

Otras relaciones de proximidad vendrían definidas de la siguiente manera, donde el operador puede ser cualquier operador descrito arriba:

<symbol_1> <operator> <symbol_2> = <aproximation_degree>

Ejemplo: Amarillo~Naraja = 0.6

Esquema general de unificadores

<term_1> <operator> <term_2> <comparator> <degree>

Los términos son identificadores, variables o términos lingüísticos como alguno de los términos compuestos. Los comparadores pueden ser los siguientes: =:=2, =\=3, >=, =<, > o <. Y por último, el grado de proximidad que está entre 0 y 1.

Directivas

1. La directiva lambda_cut sirve para establecer el grado mínimo de aproximación que se permitirá para los pasos débiles de unificación de un proceso de resolución WSLD, y para el término comparaciones realizadas con las expresiones explicadas anteriormente.
   :-lambda_cut(<cut_value>) :- lambda_cut(0.75)

2. La directiva de dominio se utiliza para declarar el dominio o universo de discurso de una variable lingüística llamada <nombre>, en la que se pueden definir términos lingüísticos.
   :-domain(<name>, <minimum>, <maximum>, <measure_unit>)
   Ejemplo:
   :- domain(age, 0, 100, years)

3. Esta directiva define los subconjuntos difusos asociados con una serie de etiquetas lingüísticas âC ‹âC‹ perteneciente a la variable denominada <dominio>, la cual debe haber sido declarada previamente con la directiva de dominio.
   :- fuzzy_set(<domain_name>, [<label_1>(<a>, <b>, <c>, <d>), <label_2>(<a>, <b>, <c>), ...]).
   Ejemplo:
   :- fuzzy_set(temperature, [cold(-40, -40, -10, 20), warm(-10, 20, 40), hot(10, 40, 80, 80)]).

4. Se puede establecer rangos de dominio para las variables de las siguientes maneras:
   Primera forma: <domain_name>#<minimum>#<maximum>
   Ejemplo:
   height#80#120
   Segunda forma:
   <range_modifier>#<domain_name>#<minimum>#<maximum>
   Ejemplo:
   about#speed#100#120
5. Se pueden establecer valores de punto:
   <domain_name>#<value>
   temperature#40
   Además se pueden establecer un modificador(este utilizará la función triangular, tal que (max(x – δ, n), x, min(x + δ, m))):
   <point_modifier>#<domain_name>#<value>
   about#age#36
6. Se pueden utilizar términos de lingüísticas compuestas como extremely, very, more_or_less and somewhat.
   <term_modifier>#<label>
   Ejemplo:
   extremely#fast
   somewhat#tall

Ejemplo de uso de directivas:

```
% DIRECTIVES
 :-domain(age, 0, 100, years).
 :-fuzzy_set(age,[young(0,0,30,50), middle(20,40,60,80),
                        old(50,80,100,100)]).
 :-domain(speed, 0, 40, km/h).
 :-fuzzy_set(speed,[slow(0,0,15,20), normal(15,20,25,40),
                          fast(25,30,40,40)]).

% FACTS AND RULES
 speed(X, fast) :- age(X, young).
age(robert, middle).
```

Si preguntas por lo siguiente: ? - velocidad (robert, algo # rápido). El sistema BPL responde "Sí con 0.375".

Negación

Hay dos tipos de negación dentro de Bousi-Prolog: Negación nítida como fracaso(**\+(<goal>)**) y Negación débil como fracaso(**not(<goal>)**).

- **Negación nítida como fracaso:** un objetivo negativo \ + (G) falla solo si el objetivo <G> tiene éxito con un grado de aproximación 1. De lo contrario, el meta negativa \+ (G) tiene éxito con grado de aproximación 1.
- **Negación débil como fracaso:** el operador no representa un tipo de negación difusa que, en el lenguaje Bousi-Prolog, se llama negación débil. Al igual que con \ +, una meta negativa no (G) falla solo cuando el argumento <G> tiene éxito con un grado de aproximación 1.

Principales diferencias entre Bousi-Prolog y FASILL

Bousi-Prolog y FASILL están enfocados al mismo objetivo, el cual es aportar un entorno de desarrollo para la programación lógica difusa. Sin embargo, a pesar de que comparten la mayoría de características, podemos notar varias que encontramos en uno de ellos y no en el otro.

Estrutura:

Por una parte FASILL se compone de los métodos de inferencia difusa y unificación débil, por el otro Bousi-Prolog utiliza una resolución sld débil además del método de la unificación débil.
La unificación débil es el método que se utiliza para relacionar los distintos valores de los programas. Su funcionamiento se ha visto en los ejemplos y se ha podido comprobar como este es similar en ambos entornos.

Por otra parte tanto el método de resolución SLD débil como el de la inferencia difusa son utilizados para hacer uso de estas relaciones y llegar a una solución. Pese a que estos no han sido nombrados explícitamente en la definición de los lenguajes, su funcionamiento ha quedado reflejado en los ejemplos prácticos.

Conjunto de normas:

Mientras que FASILL dispone del retículo completo L en el cual se agrupan los elementos con sus correspondientes valores usados en el programa, en Bousi-Prolog tenemos su equivalente llamado "directivas".
Estos dos conjuntos comparten muchas similitudes, pues en ellos se definen datos básicos del programa como rangos, variables o valores de corte. De cualquier forma, el Retículo completo de FASILL resulta más flexible que el conjunto de directivas puesto que permite la definición de funciones de una forma sencilla y rápida.

Interfaz:

Finalmente, como una opinión personal, pensamos que FASILL dispone de una interfaz más fácil de usar que Bousi-prolog. La forma que tiene de estructurar de forma separada sus 3 bloques principales hace que sea mucho más visual e intuitivo. Además de incluir la función para dibujar un árbol que refleja su funcionamiento.