# PRINCIPIOS DE PROGRAMACION LOGICA CON INFORMACION INCIERTA. DESCRIPCION DE ALGUNOS DE LOS SISTEMAS MAS RELEVANTES

## (*Principles of Logic Programming with Uncertain Information. Description of Some of the Most Relevant Systems*)

**Gonzalo ESCALADA-IMAZ**[1]
**Felip MANYÀ SERRES**[1,2]
**Alejandro SOBRINO**[3]

[1] Institut d'Investigació en Intel.ligència Artificial, Consejo Superior de Investigaciones Científicas, 48.013 Bellaterra, Barcelona. E-mail: gonzalo@iiia.csic.es

[2] Departament d'Informàtica i Enginyeria Industrial, Universidad de Lleida, 25080 Lleida. E-mail: felip@eup.udl.es

[3] Area de Lóxica e Filosofía da Ciencia, Facultade de Filosofía, Universidad de Santiago de Compostela, 15706 Santiago de Compostela. E-mail: lflgalex@usc.es

BIBLID [ISSN 0495-4548 (1996) Vol. 11: No 27; p. 123-148]

RESUMEN: El objetivo de este artículo es presentar los principios de la programación lógica borrosa y de sus principales variantes, ilustrándolas a través de un conjunto de aproximaciones que, a nuestro entender, son representativas de los avances en esta área. También incluimos la descripción de otros sistemas de programación lógica que se sustentan en lógicas de la incertidumbre diferentes de la lógica borrosa. En esta presentación presuponemos que la mayoría de los lectores no son expertos en programación lógica; para seguirla sólo se requiere un conocimiento básico de la lógica de predicados.

Descriptores: programación lógica, programación lógica multivalorada, programación lógica probabilística, fuzzy Prolog.

ABSTRACT: *The purpose of this paper is to present the principles of the fuzzy logic programming, exemplifying them by a couple of proposals that we think are representatives of the advances in this field. We include also the description of another systems of logic programming with uncertain information that are based on other logics of uncertainty which are different from fuzzy logic. This article only presuppose an elementary knowledge of the classical first-orden logic.*

Keywords: *logic programming, multiple-valued logic programming, possibilistic logic programming, fuzzy Prolog.*

## 1. Introducción

La idea de utilizar la lógica clásica como un lenguaje de programación dio origen al lenguaje Prolog. Un programa Prolog consiste en una serie de cláusulas de Horn, donde cada cláusula tiene como máximo un predicado sin negar. Estas cláusulas tienen una capacidad expresiva notable; en particular, permiten representar: 1) hechos H cuya

verdad es conocida; 2) reglas de implicación "Si A y B y C son verdad entonces D es verdad" y 3) preguntas "¿P y Q y R son simultáneamente verdad?".

Sin embargo, todo formalismo de representación del conocimiento debe ir acompañado de mecanismos eficientes de inferencia que permitan extraer soluciones a los problemas planteados con el formalismo en cuestión. La regla de resolución de Robinson (Robinson, 65) ha dado lugar a mecanismos de inferencia relativamente eficientes en lógica clásica. En el caso de Prolog, se reconoce como eficiente la estrategia llamada SLD (Lloyd, 87).

La programación lógica y la lógica borrosa surgieron en el marco de la Inteligencia Artificial. En la actualidad, ambas tienen entidad propia y han sido ampliamente utilizadas para abordar problemas complejos cuya solución requiere un cierto "grado de inteligencia": procesamiento de lenguaje natural, sistemas expertos, bases de datos, etc. En todas estas aplicaciones destaca la necesidad de representar información imprecisa, indeterminada o vaga. Aunque hay varias lógicas que se encargan de esta misión, hay un cierto consenso en reconocer que la lógica borrosa gestiona, de manera adecuada, los formatos más usuales de conocimiento incierto o vago.

En lo que respecta al diseño de un sistema deductivo eficiente para la programación lógica borrosa, un primer paso para conseguir mecanismos homólogos de inferencia vaga fue encontrar reglas de resolución borrosas adecuadas. El primer trabajo en este ámbito, publicado a principios de los setenta, se debió a Lee (Lee, 72).

Una vez que se disponía de la regla de resolución en lógica borrosa, el siguiente paso consistía en diseñar un sistema inferencial eficiente, homólogo a la estrategia SLD para la programación clásica. Este sistema, llamado interpretador, fue propuesto a mediados de los ochenta por Ishizuka y Naoki, los cuales desarrollaron el primer lenguaje (Prolog-Elf) de programación lógica borrosa basado en la regla de resolución borrosa de Lee.

De esta forma, nace una nueva línea de investigación que se conoce como programación lógica borrosa (PLB), síntesis de dos disciplinas bien consolidadas: la programación lógica y la lógica borrosa.

En este artículo pretendemos dar al lector una visión general sobre la PLB, presentando algunos de los resultados más importantes y referenciando aquellas publicaciones donde se pueden ampliar los conceptos aquí mencionados. Para ello, el texto se estructura como sigue. En la sección 2, presentamos la programación lógica clásica y repasamos algunos conceptos básicos de Prolog. En la sección 3, expondremos sucintamente la lógica borrosa y la PLB. En el apartado 4 se dan descripciones breves de algunas variantes de PLB. En la última sección, se describen algunos sistemas relevantes de programación lógica basadas en otras lógicas de la incertidumbre, que recogen aspectos de imprecisión o incertidumbre diferentes a la vaguedad lingüística. Por fin, se señalan algunas conclusiones.

## 2. La programación lógica

La programación lógica surgió en los setenta, propiciando un nuevo enfoque respecto al paradigma imperativo de lenguajes dominantes hasta entonces como Fortran y Algol. La nueva perspectiva redefine la tarea del programador: éste debe encargarse sólo de especificar en lógica de predicados el conocimiento necesario para resolver un

problema; la automatización de la prueba permitirá "liberarle" de la tarea de especificar *cómo manipular ese conocimiento para obtener una solución*. Dicho de otra forma, el programador declara su conocimiento del problema y un sistema de deducción, llamado interpretador, procesa automáticamente la información para determinar sus posibles soluciones.

Aunque existen otros lenguajes, la programación lógica se identifica a menudo con el Prolog. En esta sección centraremos nuestra atención en este lenguaje, describiendo algunos de los conceptos y procedimientos más importantes. El lector interesado en profundizar más en este tema puede recurrir a (Bratko, 90; Clocksin&Mellish, 81; Lloyd, 87).

Un programa Prolog está formado por un conjunto de hechos y reglas, también llamadas cláusulas del programa. Un hecho corresponde a un predicado y una regla corresponde a una implicación cuyo antecedente es una conjunción de predicados y cuyo consecuente es un predicado. Tanto las variables de los hechos como las de las reglas están cuantificadas *universalmente*. Así,

| | | |
|---|---|---|
| Regla: | $\forall X,\forall Y,\forall Z,$ | $padre(X, Y) \wedge padre(Y, Z) \rightarrow abuelo(X, Z)$ |
| Hechos: | | padre(juan, luis) |
| | | padre(luis, pedro) |

es un ejemplo de programa Prolog con una regla y dos hechos. En la sintaxis de Prolog se omiten los cuantificadores y se sobreentiende que todas las variables están cuantificadas universalmente; el signo de implicación se sustituye por :-, colocando el antecedente a su derecha y el consecuente a su izquierda; la conjunción de predicados del antecedente de una regla se representa por comas, las variables se escriben en mayúsculas y las constantes en minúsculas. Por tanto, el ejemplo anterior se expresa en Prolog de la siguiente forma:

| | | |
|---|---|---|
| Regla: | abuelo(X, Z) :- | padre(X, Y), padre(Y, Z). |
| Hechos: | | padre(juan, luis). |
| | | padre(luis, pedro). |

Una vez escrito un programa, podemos someterle a preguntas, que corresponden a una conjunción de predicados cuyas variables están cuantificadas *existencialmente*. En Prolog, una pregunta se representa por un signo de interrogación seguido de uno o más predicados separados por comas. En el ejemplo anterior, podríamos preguntar quién es el abuelo de Pedro de la siguiente forma:

*?- abuelo(X, pedro).*

Prolog nos respondería "X=juan", es decir la única respuesta correcta. Si preguntáramos *?- abuelo(X, Y)* -recordemos que equivale a preguntar si existe algún X y algún Y tales que X es abuelo de Y-, Prolog nos respondería "X=juan; Y=pedro". Si la pregunta fuese *?- padre(X, Y)*, obtendríamos las dos respuestas válidas (X=juan, Y=luis) y (X=luis, Y=pedro).

En el ejemplo de antes sólo hemos considerado variables y constantes pero, en general, un programa Prolog puede contener términos de primer orden $t_1,...,t_n$ en sus predicados. Por ejemplo:

```
pariente(nieto (Z, X)) :- hijo(Z, Y), hijo(Y, X)
pariente(tío (Z, X)) :- hijo(Y, X), hermano(Y, Z)
pariente(sobrino (X, Z)) :- pariente(tío (Z, X))
```

tiene los siguientes términos: nieto(Z, X), tío(Z, X) y sobrino(X, Z).

Dos puntos relevantes requieren especial atención para diseñar un interpretador eficiente y así obtener rápidamente las soluciones (si existen) de un problema: 1) la ejecución de la inferencia y 2) la estrategia de encadenamiento de las inferencias.

*1) La inferencia (unificación).* En la etapa de inferencia interviene principalmente una operación de unificación que se define sucintamente así: dados dos predicados $P(t_1,...,t_n)$ y $Q(s_1,...,s_n)$, hallar una substitución $\sigma=\{X_1/r_1,...,X_k/r_k\}$, llamada unificador más general, la cual verifica:

A) Si sustituimos en los predicados $P(t_1,...,t_n)$ y $Q(s_1,...,s_n)$ cada variable $X_i$ por el término $r_i$ indicado de la substitución $\sigma$, entonces los dos predicados son sintácticamente iguales, es decir, $P(t_1,...,t_n).\sigma = Q(s_1,...,s_n).\sigma$.

B) Se puede obtener cualquier otro unificador $\sigma'$ de $P(t_1,...,t_n)$ y $Q(s_1,...,s_n)$ por una operación de composición entre $\sigma$ y una tercera substitución $\rho$, i.e. $\sigma' = \sigma*\rho$ (el lector interesado en más detalles puede consultar (Lloyd, 87)).

La unificación es una operación capital para reducir el tiempo que el interpretador necesita para encontrar soluciones, ya que interviene en cada inferencia. Actualmente, existen aproximadamente *diez algoritmos de unificación* diseñados para obtener rápidamente el unificador más general.

Una vez definida la unificación, ya podemos indicar el proceso de ejecución de una inferencia y su papel en la resolución del problema. En Prolog, la ejecución de una inferencia recorre los siguientes pasos. Primero, elegir un átomo Q -de la conjunción actual de predicados- y un hecho P; o una regla con un consecuente P si Q y P son unificables. Segundo, obtener $\sigma$ su unificador más general. Tercero, la conjunción de predicados se transforma, bien extrayendo Q si P es un hecho o bien, reemplazando el predicado Q por el conjunto de los predicados antecedentes de la regla. El último paso consiste en aplicar $\sigma$ al conjunto de átomos de la conjunción actual. Una posible ilustración de la inferencia sería:

| | | |
|---|---|---|
| Sea la pregunta actual | $? Q_1, Q_2,..., Q_n$ | |
| Sea una regla del programa | $P :- P_1,..., P_m$ | (si m=0 es un hecho) |
| | ---------- | |
| | $? (Q_1,..., Q_{i-1}, P_1,..., P_m, Q_{i+1},..., Q_n).\sigma$ | |

*2) La estrategia de búsqueda (Prolog SLD-Resolution).* Es el encadenamiento de inferencias que realiza el interpretador para obtener las posibles soluciones. El interpretador es el programa que recibe como entrada un problema en Prolog. Su misión es encadenar inferencias para obtener soluciones al problema propuesto. Una propiedad indispensable del interpretador es que si existen soluciones del problema

planteado, el encadenamiento de inferencias debe ser tal que siempre halle todas. Recíprocamente, cada respuesta aportada por el interpretador como una solución debe ser realmente una solución del problema declarado en Prolog.

Los interpretadores de Prolog se basan en un refinamiento de la resolución conocido como SLD (Lloyd, 87). En este refinamiento, la conjunción de átomos de un estado que modela un subproblema se considera ordenada y se procesa como una pila FIFO (el primer predicado en entrar es el primero en salir, i.e. en ser resuelto). El encadenamiento de inferencias se desarrolla según las siguientes elecciones prefijadas, ateniéndose al orden de declaración de las cláusulas hechas por el usuario:

I ) El primer átomo del subproblema, i.e. $Q_1$, y
II) El primer hecho unificable con $Q_1$ o la primera regla cuyo consecuente sea unificable con $Q_1$.

La estrategia SLD de búsqueda se puede analizar según tres patrones de situaciones:

1) Si no existe una cláusula de tipo II, el interpretador topa con una situación de fracaso para solucionar el primer literal del subproblema: ninguna cláusula sirve para resolver $Q_1$. La estrategia SLD retrocede al subproblema previo y trata de aplicar otras cláusulas alternativas que no han sido aplicadas a su primer predicado. Si este procedimiento de retroceso a un subproblema, eligiendo una nueva alternativa pendiente -i.e., no empleada-, para su primer literal, se agota, volviendo al problema inicial y sin ninguna alternativa sin explorar, entonces se determina que el problema planteado en Prolog no tiene soluciones.

2) Cuando el consecuente de la cláusula elegida unifica con el átomo $Q_1$, el estado actual o subproblema se transforma, como se ha indicado previamente, en la descripción de una inferencia. Este proceso se realiza progresivamente con cada primer predicado de los subproblemas y si se obtiene un subproblema sin ningún predicado, situación que es posible gracias a la utilización de hechos en las inferencias de la pregunta que extraen un predicado y no añaden ninguno, se ha obtenido una solución. Si se desean obtener otras soluciones, se retrocede al subproblema precedente.

3) Si la pregunta inicial contiene predicados con variables, a cada estado del interpretador o subproblema se le asocia, además de la conjunción de predicados, una substitución que "resuma" los enlaces variables-términos de las diferentes unificaciones que condujeron al presente estado o subproblema. Inicialmente, la unificación es $\varepsilon=\{\}$, la cual, al no comportar ningún enlace, es compatible con cualquier substitución. La actualización de la substitución asociada a cada subproblema se realiza mediante la composición de las dos substituciones: la del subproblema previo y la obtenida en la presente inferencia. Para cada una de las soluciones del problema, obtendremos un subproblema sin predicados y una substitución. Tomando los enlaces variables-términos correspondientes exclusivamente a las variables existentes en los predicados del problema inicial, se obtiene una solución al problema planteado.

## 3. Programación lógica borrosa

Frecuentemente, la vaguedad de una información está ligada a los adjetivos calificativos de las oraciones: joven, fuerte, alto, grande, relativamente cierto, etc. Así

"Juan es joven" introduce vaguedad en ámbitos donde se requiere saber los años de Juan, aunque el hecho de saber que "Juan es joven" nos permite delimitar sus años, p. ej., entre 18 y 30 años. La vaguedad puede también manejarse en sentido dual. Es decir según sea la edad de Juan, 15, 25, 35 o 50 años, no se correspondería con la realidad particionar las afirmaciones sobre "Juan que tiene u años es joven" en falsas y verdaderas. Es decir, si "Juan tiene 31" años es relativamente más verdad que "Juan es joven" que si "Juan tiene 50 años".

La generalización de la función característica (Zadeh, 65; Zadeh 87) de un conjunto clásico a un conjunto vago fue la base para el desarrollo de lógicas que, como la borrosa, fuesen capaces de captar información vaga. Un conjunto clásico A, puede definirse exactamente a partir de su función característica $\mu_A$: U -> {0,1}, donde U es el universo de elementos sobre el cual está definido A. Los elementos u de U que pertenecen al conjunto A, son tales que $\mu_A(u)=1$, que se distinguen de los que no pertenecen (u') a A, definidos por $\mu_A(u')=0$. Un conjunto vago se representa por su función característica $\mu_A$: U -> [0,1], donde [0, 1] es el intervalo unitario de los reales. En este caso, existen tres tipos de elementos de U: los que pertenecen a A, $\mu_A(u) = 1$, los que no pertenecen a A, $\mu_A(u')=0$ y los que tienen un grado relativo de pertenencia a A, $0 < \mu_A(u") < 1$.

Ejemplos de *atributos* vagos son: alto(u), joven(u) y fuerte(u), cuyos universos son el conjunto U de personas. Así, alto(juan) tendría un cierto valor de verdad numérico, que indicaría el grado de pertenencia gradual de la estatura de Juan al conjunto vago 'alto'.

Un ejemplo de *predicado* vago es:

clima-agradable(lugar, temperatura, humedad, lluvias, nubosidad)

donde lugar es una variable cuyas instanciaciones posibles son elementos del dominio "lugares", y los demás argumentos son *variables vagas* que pueden ser instanciadas por conjuntos vagos tales como muy-bajo, bajo, medio, bastante-alto, etc. Entonces, cada tupla definida sobre "clima-agradable" está compuesta por un "lugar" y sus respectivos grados de pertenencia a conjuntos borrosos.

Dos ejemplos del caso anterior serían:

* clima-agradable('Caribe', bastante-alta, muy-baja, bajo-nivel, muy-baja)
* clima-agradable('Escocia', bastante-baja, muy-alta, alto-nivel, muy-alta)

Estas tuplas del predicado vago "clima-agradable", procesadas, por una parte, con adecuadas funciones características de pertenencia $\mu_A$ y por otra, por funciones que obedecen el contexto real sobre el cual se definen los grados de pertenencia, e.g., media aritmética, mínimo, etc., deberían llevar a la conclusión de que "Caribe" es un lugar con un clima agradable, mientras que "Escocia" no lo es.

Como hemos visto, los atributos vagos A(t) son un caso particular de predicados vagos donde su único argumento t tiene un cierto grado de pertenencia al conjunto vago A. Es señalable, también, los predicados vagos que tienen *un único universo de vaguedad*, que es su valor numérico asociado. Asimismo, en lógica clásica se conoce que un predicado de aridad n puede descomponerse en n-1 predicados de aridad dos. Por ejemplo, el predicado "clima-agradable" puede hacerlo en:

temp(lugar, temperatura), hum(lugar, humedad),
lluv(lugar, lluvias) y nubos(lugar, nubosidad).

De esta manera, los predicados vagos manipulados pueden ser siempre traducidos a predicados vagos unitarios. Por tanto, el grado de pertenencia de una tupla de un predicado vago a un producto cartesiano de conjuntos vagos, se podrá calcular más intuitivamente como una función de la conjunción de los valores numéricos de sus predicados vagos unitarios. Esta función puede ser la media aritmética, el producto u otras.

De hecho, esta división de un predicado vago de aridad n en n-1 predicados de aridad dos que son predicados vagos unarios se utiliza en la mayoría de sistemas de PLB. Una de las ventajas que presenta es que permite al usuario reflejar la vaguedad de forma más precisa, solicitándole la única función característica $\mu_A(u)$ sobre su único universo U.

Como veremos, existen diferentes maneras de representar y procesar las informaciones vagas. Una característica común a todos los lenguajes de PLB es que asocian un valor a los predicados y/o hechos y/o reglas del programa. Este valor acostumbra a ser un real del intervalo unitario, un intervalo o una etiqueta lingüística y, como veremos después, reciben diferentes interpretaciones dependiendo del sistema de que se trate. Las valoraciones en la deducción se deben propagar adecuadamente en las sucesivas transformaciones de subproblemas, asociadas a una operación de resolución vaga, en el proceso de búsqueda de soluciones del problema expresado en PLB. La naturaleza de la propagación es específica para cada sistema de los que se citan en la bibliografía.

A cada solución del problema, representada por enlaces variables-términos, se le asocia un valor que indica su grado deductivo. Este valor está en consonancia con el hecho de que, puesto que la información de la que se dispone es vaga, el interpretador no puede obtener una solución de tipo binario, sino que sólo puede responder con un cierto grado deductivo entre 'verdadero' o 'falso'.

Desde el punto de vista de la implantación del lenguaje, cabe destacar que casi todos los sistemas publicados presentan sólo intérpretes y no compiladores. El diseño de compiladores "off-line", como se realizan para el Prolog clásico, permitiría obtener una ganancia importante en el tiempo de ejecución "on-line" en la determinación de soluciones. En general, los intérpretes propuestos en el ámbito de la PLB, al igual que sucede con el Prolog clásico, efectúan un recorrido prioritario en profundidad, estrategia de encadenamiento-hacia-atrás y retroceso cronológico, es decir utilizan el refinamiento SLD.

Sin embargo, pensamos que podrían obtenerse intérpretes más eficientes para la PLB, utilizando otras estrategias de búsqueda (Escalada-Imaz, Manyà-Serres; 95; Escalada-Imaz, Manyà-Serres; 96) distintas a la SLD. En efecto, en el proceso SLD no se consideran los valores numéricos. Con la estrategia propuesta en las dos referencias precedentes, se puede lograr una ganancia computacional significativa en la búsqueda de soluciones.

## 4. Sistemas de programación lógica borrosa

A continuación analizamos varios sistemas de PLB, estudiamos sus principales características y referenciamos aquellos artículos en donde el lector interesado puede ampliar la información que a continuación se expone. Los sistemas que veremos en esta sección son diferentes variantes de la PLB.

### 4.1. Sistema Prolog ELF (Ishizuka&Naoki, 85)

El Prolog ELF (Ishizuka&Naoki, 85) es un sistema basado en la teoría propuesta en (Lee, 72). Se asocia un valor T(C) a las cláusulas C del programa -hechos y reglas- tal que $0 \leq T(C) \leq 1$, puesto que cada cláusula vaga puede tener algún(os) predicado(s) vago(s). Así, T(C) es un valor global, que implícitamente es función de los grados de pertenencia relativa de las diferentes tuplas de los predicados vagos declarados en C. En este caso, y desde un punto de vista lógico, los autores substituyen la lógica bivaluada por la lógica borrosa max-min, definida así:

-El conjunto de valores de verdad es el intervalo real unitario;
-Una interpretación I asigna un valor a cada fórmula atómica;
-El valor de la interpretación de la conjunción es el mínimo de sus conjuntivos;
-El valor de la interpretación de la disyunción es el máximo de sus disyuntivos; y
-El valor de la interpretación de la negación es la función complemento.

Una fórmula A es insatisfactible si y sólo si para toda I, $I(A) < 0'5$. Una fórmula A es consecuencia lógica de un conjunto de fórmulas S si y sólo si, para toda interpretación I, $I(S) \geq 0'5$ implica $I(A) \geq 0'5$.

La estrategia SLD se adapta como sigue: se comienza por la conjunción de predicados $C_0$ que forman el problema inicial propuesto por el usuario y se considera el caso general; es decir, aquel en el que se desean conocer todas sus soluciones. La resolvente R(C, C') de dos cláusulas C y C' se define como en el caso clásico: la conjunción de predicados C es el subproblema presente y C' es una cláusula -hecho o regla- del programa declarado. Su valor numérico T(R(C, C')) verifica la siguiente inecuación:

$$\alpha = \text{Min}\,(T(C), T(C')) \leq T(R(C, C')) \leq \text{Max}\,(T(C), T(C')) = \beta$$

Los límites $[\alpha, \beta]$ son función del predicado sobre el cual se realiza la resolución.

También se permite al usuario establecer un umbral $\pi$ de manera que el subproblema sea considerado en el proceso sólo si su cota inferior $\alpha$ iguala o sobrepasa al umbral $\pi$; es decir, si $\alpha \geq \pi$. En general, se tiene que $\alpha$ (resp. $\beta$) es el mínimo (resp. máximo) de las cotas inferiores (resp. superiores) de los subproblemas recorridos desde el problema inicial hasta el subproblema actual.

En el caso de que se requiera la mejor o las mejores soluciones, se hace una búsqueda exhaustiva, explorando todas las posibles alternativas; es decir, por cada subproblema se consideran todas las cláusulas del programa -hechos y reglas- que unifican con el primer predicado de la conjunción del subproblema.

Si $R(C, C')=\emptyset$ es un subproblema que no contiene ningún predicado, entonces extrayendo los enlaces de las variables-términos correspondientes a las variables de

la pregunta inicial se obtiene una solución T(Ø), cuyo intervalo de vaguedad es [αø, βø].

Prolog ELF permite emplear diferentes modos de operación según se requiera: 1) obtener y devolver la primera solución encontrada; 2) determinar todas las posibles soluciones; y 3) determinar las n mejores soluciones atendiendo a su valor numérico.

### 4.2. Sistema FPROLOG (Martin&Baldwin&Pilsworth, 87)

Este sistema (Martin et al., 87) sólo permite la asignación de valores a los hechos. A las reglas se les asocia el valor 1. Por tanto, desde el punto de vista de la capacidad representativa, este sistema parece ser comparativamente inferior al precedente Elf-Prolog, puesto que la implicación de una regla puede estar afectada por algún tipo de incertidumbre.

Los conjuntos vagos A están representados por atributos vagos A (como se mencionó, utilizamos la misma notación para el conjunto vago y para el atributo que lo modeliza, siempre que no exista ambiguedad en el contexto en el que son utilizados) y un conjunto de hechos (A(u); ∂), donde u pertenece al universo de discurso U de A y ∂ al intervalo real unitario [0,1]. En cada par (A(u); ∂) ∂ indica el grado de pertenencia de u a A. El grado de pertenencia de los elementos u' que no han sido declarados se obtienen por aproximación lineal a partir de los pares (A(u); ∂) declarados.

*Ejemplo:*

```
pareja_joven(X, Y) :- edad(X, X1), edad(Y, Y1), joven(X1), joven(Y1).
edad(juan, 40).
edad(rosa, 35).
μjoven(u)={(μjoven(0),0),(μjoven(10),0),(μjoven(15),0'2),
    (μjoven(20),1), (μjoven(25),1), (μjoven(35),0'8), (μjoven(40),0'6),
    (μjoven(45),0'4), (μjoven(50),0).

? pareja_joven(juan, rosa).
pareja_joven(juan, rosa)=0'6
```

El valor numérico asociado a la conclusión de una regla es, por defecto, el mínimo de los valores asociados a sus predicados antecedentes. En nuestro caso, tenemos las siguientes asociaciones numéricas: edad(juan, 40) y edad(rosa, 35) a 1, puesto que son predicados clásicos deducibles. joven(40) y joven(35) tiene valores asociados de 0'6 y 0'8, respectivamente. Por tanto, el valor deductivo de pareja_joven(juan, rosa) es Min{1, 1, 0'6, 0'8}=0'6, que corresponde a la respuesta dada por el sistema.

Algunas características de FPROLOG son:

- FPROLOG ofrece al programador la posibilidad de utilizar otras funciones distintas al mínimo.
- FPROLOG trata la negación así: si un predicado P puede deducirse con un valor de verdad 1-α, entonces no(P) se deduce con un valor de verdad α.
- FPROLOG fue implementado en Franz Lisp y su sintaxis está basada en listas en lugar de utilizar la conocida sintaxis de Edimburgo, aunque nosotros hemos utilizado ésta para facilitar su lectura.

- FPROLOG trata de manera clara y explícita la vaguedad existente en un programa. En efecto, frecuentemente los predicados vagos son, o se reducen a, un conjunto de predicados vagos unarios, los cuales tienen un único universo U sobre el cual se definen conjuntos vagos. Para cada uno de estos conjuntos A se declara la función $\mu_A(u)$. De esta forma, el valor numérico asociado a un predicado vago unitario es el grado de pertenencia de u a A determinado por $\mu_A(u)$. En consecuencia, el valor numérico que devuelve FPROLOG es fiel respecto a la información vaga considerada.

Esta característica no se aprecia tan claramente en Prolog-ELF, puesto que T(C) aparece más bien como una estimación global de la información vaga representada por los predicados vagos de C.

### 4.3. El sistema f-Prolog (Li&Liu, 90)

A este lenguaje (Li&Liu, 90, Liu, 92) subyace la lógica *max/min.* Un programa está formado por hechos-V y reglas-V (la V denota la vaguedad presente tanto en los hechos como en las reglas).

1) Un *hecho-V* es un par de la forma $P(t_1,...,t_n)$:- [V], donde $P(t_1,...,t_n)$ es un predicado vago y V es un valor del intervalo real [0,1] que expresa la "creencia" de que el consecuente sea verdadero cuando el antecedente es verdadero. Un hecho-V tiene algunos de sus términos ti definidos sobre conjuntos vagos Ai. El valor V indica el grado de pertenencia de una tupla j-aria $(s_1,...,s_j)$, $j \leq n$, al producto cartesiano $(A_1x...xA_j)$ de los conjuntos vagos definidos en los argumentos de P. En general, se tiende a transcribir los hechos-V como hechos.vagos.unarios-V. El valor asignado a un hecho.vago.unario-V es $\mu_A(u)$, correspondiente al grado de pertenencia de u al conjunto vago A definido sobre su único universo. Así, los conjuntos vagos se describen por un conjunto de pares $(A(t); \mu_A(t))$, siendo t un término declarado en el programa.

2) Una *regla-V* es una expresión de la forma $P:-[V]- Q_1,...,Q_i,...,Q_n$, donde P, $Q_1,...,Q_i,...,Q_n$ son predicados, siendo al menos uno, un predicado vago, y V es un valor del intervalo real [0,1]. Si el valor numérico que modela la vaguedad en los predicados $Q_i$ es $\mu(Q_i)$, $1 \leq i \leq n$, entonces el valor numérico de la conjunción de $Q_1,...,Q_i,...,Q_n$ se define como el mínimo de los valores $\mu(Q_i)$, $1 \leq i \leq n$, y lo denotaremos por $\mu(Q)$. El valor numérico de la conclusión se define como $\mu(P)=V_x \mu(Q)$. Esta ecuación significa que el grado de vaguedad del consecuente P aumenta proporcionalmente, ya que $V \in [0,1]$, con respecto al grado de vaguedad de la conjunción del antecedente. Como en los hechos-V, los predicados utilizados en las reglas son en general predicados vagos unitarios.

Las preguntas pueden ser de dos tipos:

- ?-[f]- $Q_1,...,Q_n$, donde $Q_1,...,Q_n$ son predicados y f es un valor del intervalo real [0,1]. Este tipo de preguntas pueden leerse como: ¿Puede ser deducida la pregunta con un valor mayor o igual al umbral f?
- ?-[F]- $Q_1,...,Q_n$, donde $Q_1,...,Q_n$ son predicados y F es una variable. Este tipo de preguntas pueden leerse como: ¿Con qué valor máximo se satisface el problema $Q_1,...,Q_n$?.

*Ejemplo:* Dado el siguiente programa:

```
lleva_zapatos_grandes(X) :-[0'9]- alto(X) -.
alto(santiago):-[1]-.
alto(tomas):-[0'9]-.
alto(juan):-[0'8]-.
alto(blas):-[0'6]-.
```

Recordemos que el usuario debe declarar los valores $A(t)=\mu_A(t)$, para cada término t del programa y del universo correspondiente a A, como se ha indicado previamente. En este ejemplo A es el atributo "alto". He aquí una sesión de consulta:

```
?-[0'81]- lleva_zapatos_grandes(tomás):     => Sí.
?-[0'81]- lleva_zapatos_grandes(juan):      => No
?-[0'81]- lleva_zapatos_grandes(X)          => (X=santiago); (X=tomás)
?-[F]- lleva_zapatos_grandes(santiago)      => F=0'90
?-[F]- lleva_zapatos_grandes(X)             => (X=santiago, F=0'90)
                                            => (X=tomas, F=0'81)
                                            => (X=juan, F=0'72)
                                            => (X=blas, F=0'54)
```

Obsérvese que la similaridad entre f-Prolog y FPROLOG es notable.
Algunas características de f-Prolog son las siguientes:

- En f-Prolog se suponen conocidos los términos t del programa, así como el grado de pertenencia de t al conjunto vago A, es decir los pares $(A(t); \mu_A(t))$ son explícitamente declarados como hechos-V. Por ejemplo:

```
alto(santiago):-[1]-.
alto(juan):-[0'8]-.
alto(tomás):-[0'9]-.
alto(blas):-[0'6]-.
```

Lo cual es equivalente a declarar:

$\mu_{Alto}(X)$= {(alto(santiago),1), (alto(tomás),0'9), (alto(juan), 0'8), (alto(blas), 0'6)}

- En FPROLOG se suponen conocidas las funciones características $\mu_A(u)$ de todos los conjuntos vagos A del problema. Esta definición se describe a partir de un atributo A que representa al conjunto vago. Es misión del operario declarar un conjunto finito de pares $(A(u),\mu_A(u))$ de manera que por interpolación lineal se pueda calcular cualquier otro valor $\mu_A(u')$ tal que $(A(u'),\mu_A(u'))$ no haya sido explícitamente declarado. Un ejemplo sería:

```
alto(1'85),  μAlto(1'85)=1
alto(1'80),  μAlto(1'80)=0'9
alto(1'75),  μAlto(1'75)=0'8
alto(1'75),  μAlto(1'70)=0'6
```

que equivale a:

$\mu_{alto}(X)$= {(alto(1'85), 1), (alto(1'80), 0'9), (alto(1'75), 0'8), (alto(1'85), 0'6)}

Así, si se desea conocer cuál es el grado en el que una persona es alta, podemos utilizar la regla en FPROLOG:

persona_alta(X) <-- [1] persona(X), alto(X);

$\mu_{Alta(X)}$= {(alto(1'85), 1), (alto(1'80), 0'9), (alto(1'75), 0'8), (alto(1'85), 0'6)}

Supongamos que sabemos que Tomás mide 1'80 m., entonces de:

persona_alta (tomas) <- persona(tomas), alto(1'80);

deducimos que Tomás es una persona-alta con un grado 0'9=Min{1, 0'9}.

Recordamos que: I) los predicados clásicos, tales como persona(X), tienen un valor numérico de 1 (resp. 0) si se pueden (resp. no se pueden) deducir, e.g. persona(tomás) recibe el valor 1; y II) el valor del consecuente es el mínimo de los valores de los antecedentes, i.e. Min{1, 0'9}=0'9.

Por tanto, vemos que la manera de proceder en f-Prolog es similar a la de FPROLOG.

El sistema de (Li&Liu, 90) permite también asociar a hechos y reglas tanto valores de verdad numéricos como lingüísticos, e.g. seguro, muy posible, bastante posible,..., imposible. Cada una de estas etiquetas lingüísticas corresponde a un subintervalo $[a_i, a_{i+1}]$ de [0,1]. A fin de facilitar el diseño del interpretador de este lenguaje de PLB basado en etiquetas lingüísticas, f-Prolog considera un número finito de etiquetas lingüísticas.

*Ejemplo:* Dado el programa siguiente:

conduce(X, coche_grande):-[posible]- americano(X).
americano(robert):-[0'7].

y los siguientes correlatos numéricos para los valores de verdad lingüísticos: seguro [1,1]; muy-posible [1,0'9]; posible [0'75,0'9]; bastante-posible [0'525,0'75]; podemos pedir al sistema que responda con un término lingüístico:

?-[F]- conduce(X, coche_grande): X=robert F= bastante_posible

Como se trata de una lógica max/min como se mencionó al principio de la sección, resulta la siguiente secuencia de operaciones:

Min{posible, 0'7}= Min{posible, bastante_posible}= bastante_posible.

### 4.4. Fuzzy Sets Prolog (Umano, 87)

La principal característica del *Fuzzy Sets Prolog* de (Umano, 87) es que no sólo permite asociar a hechos y reglas sus correspondientes valores numéricos de vaguedad, sino que también se permiten *cuantificadores lingüísticos*, tales como "cierto", "bastante cierto", etc., que son expresados por conjuntos vagos. Permite también la utilización de predicados vagos donde los términos de sus argumentos, formados por símbolos de función, de variables y de constantes, corresponden a funciones, variables y constantes (conjuntos) vagas.

Para ilustrar la lógica borrosa y los mecanismos inferenciales que le subyacen, consideraremos el caso simple de la lógica proposicional borrosa. Una proposición vaga, como "unos 26 años", queda caracterizada por la función característica del conjunto borroso que le corresponde. Los cuantificadores vagos, del tipo "es más o menos cierto"; "es cierto", etc., se expresan también por medio de conjuntos borrosos.

*Ejemplo:* Sea el enunciado, "Es más o menos cierto que Juan tiene unos 26 años". Vemos que el cuantificador lingüístico es "es mas o menos cierto", y el conjunto vago es "tiene unos 26 años".

Su expresión en *Fuzzy Sets Prolog* es como sigue:

(La notación ($n_1/u_1$ ;...; $n_k/u_k$) expresa que al elemento $u_i$ de su universo U se le asocia el grado de pertenencia $n_i$ en [0,1]).

"más o menos cierto" = (0'8/1; 1/0'9; 0'5/0'8)
"Juan tiene unos 26 años" = (0'6/25; 1/26; 0'7/27)

Dados los dos conjuntos vagos indicados, se le pregunta al sistema:

- ? Cuantificador, ? edad(juan).

ofreciendo la siguiente respuesta:

(0'5/0'8; 1/0'9; 0'8/1) ; (0'6/25; 1/26; 0'7/27)
"más o menos cierto" - "Juan tiene unos 26 años"

de la cual se puede deducir, por ejemplo:

(0'5/0'8; 1/0'9; 0'8/1); (0'6/25; 1/26; 0'7/27) -> "más o menos cierto" que "Juan tiene unos 26 años"
(0'8, (0'6/25; 1/26; 0'7/27)) -> "es más o menos cierto que Juan tiene unos 26 años"
-> "es cierto con grado 0'8, que Juan tiene unos 26 años";
(0'5, (0'6/25; 1/26; 0'7/27)) -> "es cierto con grado 0'5, que Juan tiene unos 26 años";
-> "es medianamente cierto, que Juan tiene unos 26 años";
-> "es medianamente falso, que Juan tiene unos 26 años";
((0'5/0'8; 1/0'9; 0'8/1); 0'6) -> "es más o menos cierto que con grado 0'6, Juan tiene 25 años";
(0'8; 0'6) -> "es cierto con grado 0'8, que con grado 0'6 Juan tiene 25 años".

Otra opción posible, más compleja que la anterior, sería preguntar:

¿Cómo es de "cierto" que Juan, que tiene unos 26 años, es joven?

?- Cuantificador?, #joven (juan),

donde # denota un conjunto vago predefinido en el sistema, en este caso, joven(u). Así, joven(juan) se obtiene mediante los siguientes pasos:

dado: (0'6/joven(25); 1/joven(26); 0'7/joven(27))
y
dado: #joven(u)=(0'875/25; 0'82/26, 0'755/27)
se obtiene: (0'6/0'875 ; 1/0'82; 0'7/0'755) para joven(juan)

Puesto que el cuantificador de la pregunta no está instanciando a ningún cuantificador, el sistema busca el máximo grado de verdad posible de joven(juan). En este caso, sólo conoce el cuantificador "más o menos cierto".

Por tanto, el sistema devuelve:

"más o menos cierto"-"#joven(juan)" =
(0'5/0'8; 1/0'9; 0'8/1)-(0'6/0'875; 1/0'82; 0'7/0'755)

Supongamos que la siguiente pregunta al sistema es:

-? Entre "cierto" y "más o menos cierto" -? "Juan tiene unos 26 años"

Para responder de modo completo a la pregunta planteada, el sistema debe obtener el conjunto borroso correspondiente a entre "cierto" y "más o menos cierto", lo que consigue mediante la conjunción de las distribuciones de posibilidad Pd correspondientes a "cierto" y "más o menos cierto" (ver Umano, 87) para la definición de esta operación), que produce:

Entre "más o menos cierto" y "cierto":
Pd((0'5/0'8; 0'8/1; 1/0'9), (0'6/0'875; 1/0'82; 0'7/0'755) =
{0'7/0'755; 0'5/0'8; 1/0'82; 0'6/0'875}.

Por tanto la respuesta del sistema a:
-? Entre "cierto" y más o menos cierto" -? "Juan que tiene unos 26 años es joven" es:

(0'7/0'755;0'5/0'8;1/0'82;0'6/0'875); (0'6/0'875;1/0'82;0'7/0'755)
Entre "cierto" y "más o menos cierto" #joven (juan))

## 5. Programación lógica basada en otras lógicas de la incertidumbre

A continuación se exponen varios sistemas de programación lógica donde la lógica borrosa coexiste con otras lógicas de la incertidumbre, que tratan de representar aspectos como el grado de confianza en una regla, grados de verdad y falsedad, etc.

### 5.1. Programación vaga y valores de confianza (Mukaidono et al., 89)

En (Shen et al., 88; Mukaidono et al., 89) se propone una lógica de confianzas que desde el punto de vista de la lógica borrosa max-min tiene la siguiente particularidad:

- A cada grado de pertenencia $\mu_A(t)$ de un elemento t a un conjunto vago A se le asocia un "coeficiente confianza" $CC(\mu_A(t))$. Por tanto, CC es una función CC: $\mu_A(u) \in [0,1]$ --> [-1,1]. El valor CC particiona las aserciones en dos clases:

I) *Las aserciones nítidas*, i.e. verdadaderas $\mu_A(u) \approx 1$ o falsas $\mu_A(u) \approx 0$.
Si $CC(\mu_A(u) \approx 1) \approx 1$ (resp. $CC(\mu_A(u) \approx 0) \approx -1$), u pertenece (resp. no pertenece) al conjunto vago A.

II) *Las aserciones ambiguas*, i.e. $\mu_A(u) \approx 0'5$, $CC(\mu_A(u) \approx 0'5)=0$. Los puntos u tales que $\mu_A(u) \approx 0'5$ no aportan nada sobre la verdad/falsedad de que u pertenezca al conjunto A.

Como se definió en §-3, un predicado vago $P(t_1,...,t_i,...,t_n)$ tiene al menos uno de sus elementos $t_i$ definido en un universo $U_i$, representado por una variable vaga $X_i$, la cual puede ser instanciada a diversos conjuntos vagos $A_{i1},...,A_{im}$. Generalizando el punto precedente, si $(t_1,...,t_i,...,t_n)$ es una tupla, su grado de pertenencia a la relación vaga P se obtiene:

I) Instanciando las variables vagas de los argumentos de P a conjuntos vagos.

II) Calculando el grado de pertenencia de cada $t_i$ a su respectivo conjunto vago $A_i$ en el producto cartesiano $(A_1xD_2x...xA_ix...xA_{n-1}xD_n)$ asociado a P, donde $A_i$ indica un conjunto vago y $D_i$ un domino clásico.
   A) El grado de pertenencia de un término t a su conjunto clásico D es 1 o 0.
   B) El grado de pertenencia de un término t a su conjunto vago A está dado por $\mu_A(t)$.

III) El valor de pertenencia $\mu_{(A_1xD_2x...xA_ix...xA_{n-1}xD_n)}(t_1,...,t_i,...,t_n)$ se calcula como una función de los valores $\mu_{A_i}(t_i) \in \{0,1\}$ y los valores $\mu_{A_i}(t_i) \in [0,1]$. Esta función, como se dijo previamente, puede ser el mínimo, el producto u otras.

Así, dado un predicado $P(t_1,...,t_n)$ cuya tupla $(t_1,...,t_n)$ tiene un valor numérico $T(\{P(t_1,...,t_n)\})$ de pertenencia al producto cartesiano correspondiente, su confianza $CC(T(\{P(t_1,...,t_n)\}))$ se define como:

$CC(T(\{P(t_1,...,t_n)\})) = (T(\{P(t_1,...,t_n)\}) - 0'5) \times 2 \in [-1,1]$, y su valor absoluto; $|CC(T(\{P(t_1,...,t_n)\}))| \in [0,1]$, mide la distancia al punto 0'5 de máxima ambigüedad.

La notación $T(\{..\})$ indica que el grado de vaguedad T no sólo se aplica a predicados sino a conjuntos de predicados, que consideraremos cláusulas.

Nótese que cuando el grado de vaguedad es exactamente 0 ó 1; es decir, cuando no hay vaguedad, el coeficiente de confianza CC asociado es 1 ó -1, respectivamente. Este caso incluye a los predicados clásicos; es decir, sus argumentos están definidos exclusivamente en dominios y no en universos. Por el contrario, cuando el grado de pertenencia es de máxima vaguedad, es decir 0'5, no se tiene ninguna confianza $CC(T(\{P\}))=0$.

El grado de confianza se obtiene así:

- Sean dos cláusulas $C_1 \vee P(t_1,...,t_n)$ y $C_2 \vee \neg P'(t_1',...,t_n')$. El valor de confianza CC de la resolvente $C_1 \vee C_2$ se define como:

$$CC(C_1 \vee C_2) = (Max\{T(\{P(t_1,...,t_n)\})), T(\{\neg P'(t_1',...,t_n')\})\} - 0'5) \times 2.$$

El coeficiente de confianza de esta resolvente tiene la propiedad deseada: cuanto mayor sea el grado de pertenencia de la tupla $(t_1,...,t_n)$ al producto cartesiano de los conjuntos vagos de P establecidos en $C_1 \vee P$ y menor sea el grado de pertenencia al producto cartesiano de los conjuntos vagos de P' establecidos en $C_2 \vee \neg P$, más confianza se tiene de que la resolvente $(C_1 \vee C_2)$ sea nítida y no ambigua, es decir $|CC(C_1 \vee C_2)| \approx 1$.

Si se obtienen varios subproblemas sin conjunción de predicados, es decir, varias soluciones, se considera como solución la que posea el mayor coeficiente CC.

Si se conoce el coeficiente de una cláusula CC(T(Cl)), entonces el valor del predicado que ha sido utilizado en la resolución se puede calcular por la expresión:

$$\max\{T(\{P(t_1,\ldots,t_n)\}),T(\{\neg P'(t_1',\ldots,t_n')\})\} = CC(T(Cl)) \times 0'5 + 0'5$$

Si se supone que $T(P) = 1-T(\neg P')$, es decir $T(P) + T(\neg P')=1$, que es estrictamente verdad en LPB, entonces:

$$Max\{T(\{P\})),\ T((\{\neg P'\}))\} = CC(T(Cl)) \times 0'5 + 0'5$$

Si $T(P) > T(\neg P')$ entonces: $T(P) = CC(T(Cl)) \times 0'5 + 0'5$
Si $T(P') < T(\neg P)$ entonces: $T(\neg P') = CC(T(Cl)) \times 0'5 + 0'5$

Esta metodología introduce algunas diferencias con la resolución de Lee. En efecto, hay que conocer el valor de pertenencia de una tupla a cada predicado vago para poder calcular la confianza de las resolventes. Sin embargo, se elimina la restricción de Lee, que consiste en que las cláusulas deben tener un valor asociado mayor que 0'5 para poder ser empleadas en una resolución.

El problema de tener que conocer el valor de verdad de los predicados, vagos o no, para calcular la confianza de las resolventes se subsana asignando pesos a las reglas. Dada una regla $P\rightarrow Q$, su peso se define como sigue: $w(P\rightarrow Q)=CC_p.CC_q$, donde $CC_p$ y $CC_q$ son, respectivamente, la confianza de la premisa y la de la conclusión. De esta forma, una regla $P\rightarrow Q$ es significativa si y sólo si, $|CC_p| \geq |w(P\rightarrow Q)|$ y $|CC_q| \geq |w(P\rightarrow Q)|$. Es claro que cada $w(P\rightarrow Q) \in [-1,1]$ puesto que $CC_p, CC_q \in [-1,1]$.

*Ejemplo:* Ilustraremos cómo procede la PLB con coeficientes de confianza para determinar T({C}) y CC(T({C})) del siguiente programa en lógica proposicional borrosa. La estrategia de búsqueda no es la clásica SLD que hemos expuesto en la sección 2, sino su "dual", llamada de encadenamiento-hacia-adelante, que procede realizando inferencias de los hechos hacia la pregunta.

1. $A \rightarrow B$ {w1=0'3}
2. $B \rightarrow C$ {w2=-0'4}
3. $A \rightarrow D$ {w3=-0'7}
4. $D \rightarrow E$ {w4=-0'1}
5. A {T(A)=0'8}

Puesto que:

| | |
|---|---|
| $CC(T(\{A\})) = (T(\{A\})-0'5) \times 2 = 0'6$ | implica: |
| $\|CC(T\{A\})\| > \|w1\|$ | implica: |
| $CC(T(\{B\})) = w1 + CC(T(\{A\})) = 0'5$ | implica: |
| $\|CC(T(\{B\}))\| > \|w2\|$ | implica: |
| $CC(T(\{C\}))= w2 + CC(T(\{B\})) = -0'8$ | implica: |
| $T(C) = CC(T(\{C\})) \times 0'5 + 0'5= 0'1$ | |

Otro camino de deducción de CC(T({C})) podría ser, eventualmente a través de la regla 3, pero esta no es significativa ya que |CC(T({A}))|=0'6<|w3|. Por tanto, se tiene definitivamente que T(({C}))=0'1.

Explicitamos las operaciones requeridas en la obtención de las resolventes, su valor numérico deductivo T({C}) y su coeficiente de confianza CC(T({C})), que conducen a la resolvente buscada {C}.

I) Resolvente {A},
$T(\{A\}) = 0'8$
$CC(T(\{A\})) = (T(\{A\}) - 0'5) \times 2 = 0'6$

II) Resolvente {B} = R({A}, {¬A ∨ B}),
$CC(T(\{B\})) = w1 \div CC(\{A\}) = 0'3 \div 0'6 = 0'5$
$T(\{B\}) = CC(T(\{B\})) \times 0'5 + 0'5 = 0'5 \times 0'5 + 0'5 = 0'725$

III) Resolvente {C} = R({B}, {¬B ∨ C})
$CC(T(\{C\})) = w2 \div CC(T(\{B\})) = - 0'4 \div 0'5 = - 0'8$
$T(\{C\}) = CC(T(\{C\})).0'5 + 0'5 = 0'1$

En este caso, sólo existe una combinación de cláusulas que conducen a la resolvente {C} y por tanto, su coeficiente de confianza es CC(T({C}))=0'1. Si existieran otros caminos de prueba, se consideraría como valor de confianza de {C} la que tuviera el valor absoluto máximo de confianza de las resolventes {C}.

La manera de trabajar con predicados vagos es similar a la utilizada en FPROLOG.

### 5.2. Fuzzy Operator Logic Programming (Weigert et al., 93)

El sistema propuesto en (Weigert et al., 93) trabaja con una lógica de la incertidumbre que, en sentido representacional, subsume a la lógica borrosa. En efecto, en (Weigert et al., 93) a diferencia de otros sistemas, ha sido diseñada una lógica para trabajar en contextos de incertidumbre donde la vaguedad es sólo uno de los factores causantes de esa incertidumbre. Dos ejemplos sencillos de situaciones reales son:

1) Se debe captar la subjetividad imposible de evitar en la definición de los conjuntos vagos por el usuario. Por ejemplo, si se pidiera a una persona que definiera la función característica del conjunto vago joven (seccion § 3), es muy probable que fuese diferente de la que dibujaría una segunda persona. Lo natural es que esas dos funciones sean "próximas", pero no idénticas. En este caso, la lógica considerada ha de captar la imprecisión y la vaguedad intrínseca de la definición de joven.

2) En otras aplicaciones puede ocurrir que la información de que se dispone no sea completa; es decir, que no se tengan a mano ciertas variables o parámetros referentes al contexto de trabajo. En este caso necesitamos representar la incompletud de la información.

Estos dos ejemplos indican que aspectos como la imprecisión, la vaguedad o el conocimiento parcial están presentes y son comunes a muchos contextos reales.

Por ello (Weigert et al., 93) han desarrollado la *Fuzzy Operator Logic* y han implementado un mecanismo de inferencia que extiende el Prolog clásico para trabajar con lógica borrosa, conocimiento subjetivo y contextos con incertidumbre debidos a diversas causas. Veamos algunos ejemplos donde la incertidumbre aparece sobre todo por la subjetividad del experto.

*Ejemplos*:

1) A --> 0'9B indica que, si la "moneda está débil" entonces el grado subjectivo de pertenencia de la "la tendencia actual de la bolsa" al conjunto vago la "bolsa cae" es de 0'9. El parámetro 0'9, como se ha dicho anteriormente, depende de la subjetividad del experto al manejar situaciones reales con incertidumbre: conocimiento parcial, conjuntos vagos, imprecisión en la percepción de los hechos, etc. Por tanto, 0'9 es un grado de incertidumbre.

2) 0'6(A --> 0'9B) indica que el grado de confianza GC que daría, por ejemplo, un asesor de inversiones, a la regla A --> 0'9B es de 0'6. Este coeficiente viene dado por el conocimiento que un especialista tiene sobre el hecho en cuestión. Nótese que el grado de confianza se aplica a un conocimiento que de por sí, tiene un grado de incertidumbre GI.

Para operar con grados de incertidumbre-confianza GIC, (Weigert et al., 93) proponen usar la siguiente función de combinación, $\otimes$:

$$\lambda_1 \otimes \lambda_2 = (2\lambda_1\text{-}1) \times \lambda_2 - \lambda_1 + 1, \text{ donde } \lambda_1, \lambda_2 \in [0,1]$$

GIC(H) se define sobre fórmulas de la manera siguiente:

Si H= $\lambda$.G entonces GIC($\lambda$.G) = $\lambda \otimes$ GIC(G)
Si H= F $\wedge$ G entonces GIC(F $\wedge$ G) = Min {GIC(F), GIC(G)}
Si H= F $\vee$ G entonces GIC(F $\vee$ G) = Max {GIC(F), GIC(G)}

*Ejemplo:*

Sea S = 1 $P_1 \wedge$ 0'7 (0'9 $P_2 \vee$ 0'2 $P_3$).
Supongamos GIC($P_1$)=GIC($P_2$)=1, GIC($P_3$)=0.
Obtengamos ahora los GIC progresivamente hasta determinar GIC(S);

GCI(1 $P_1$) = 1 $\otimes$ 1 = (2.1 - 1).1 -1+1 = 1
GCI(0'9 $P_2$) = 0'9 $\otimes$ 1 = (2.0'9 - 1).1 -0'9+1 = 0'9
GCI(0'2 $P_3$) = (2.0'2 - 1).0 -0'2+1 = 0'8
GCI(0'9 $P_2 \vee$ 0'2 $P_3$) = Max{GCI(0'9 $P_2$), GCI(0'2 $P_3$)} = Max{0'9, 0'8} = 0'9
GCI(0'7.(0'9 $P_2 \vee$ 0'2 $P_3$)) = 0'7 $\otimes$ GCI(0'9 $P_2 \vee$ 0'2 $P_3$) = 0'7 $\otimes$ 0'9 = 0'66
GCI(S) = Min{GCI(1 $P_1$), GCI(0'7(0'9 $P_2 \vee$ 0'2 $P_3$))} = Min{1, 0'66} = 0'66

En conclusión, GCI(S) = 0'66.

La *Fuzzy Operator Logic Programming* incluye también una modificación del principio de resolución, que viene exigido por el establecimiento de un umbral ($\Lambda$)

para particionar las afirmaciones graduadas en afirmaciones ciertas -por encima del umbral- y falsas -por debajo del mismo-. Algunas de las particularidades que introduce esta modificación, respecto a la definición de la negación y, por tanto, a la posible eliminación de predicados en la resolución, son las siguientes:

Sea una cláusula $C=(l_1 A_1 \vee l_2 A_2 \vee ... \vee l_n A_n)$, $n \geq 1$.

1) Si para todo $l_i$, $l_i \leq 1-\Lambda$ o para todo $l_i$, $l_i \geq \Lambda$;
2) Existe una sustitución $\sigma$ tal que $A_1.\sigma=A_2.\sigma=...=A_n\sigma$, entonces $C.\sigma$ es una $\Lambda$– cláusula.

Sean $l_1 A_1$ y $l_2 A_2$ dos predicados. Si existe una sustitución $\sigma$ tal que $(l_1A_1 \vee (1-l_2) A_2).\sigma$ es una $\Lambda$–cláusula unidad, entonces se dice que $l_1 A_1$ y $l_2 A_2$ son $\Lambda$–complementarios bajo la sustitución $\sigma$. Se puede entonces considerar una resolución adaptada al $\Lambda$-operador, que se basará en esta noción relativa de complementariedad.

*Ejemplo:*

Sea S = {0'7 (1A $\wedge$ (0B $\vee$ 1C)), 0'8 (0A $\vee$ 0D $\vee$ 0C)}.
Mostrar que F = 0'3 (0B $\vee$ 0D) es una consecuencia lógica de S.
– En primer lugar, se debe hallar la forma clausal de S $\cup$ {0F}:

I) La forma clausal de S es:
1. 0'7A
2. 0'3B $\vee$ 0'7C
3. 0'2A $\vee$ 0'2D v 0'2 C

II) La forma clausal de 0F es:
4. 0'7B
5. 0'7D

III) Usando la $\Lambda$–resolución (para $\Lambda$=0'69), se obtiene:
6. 0'7C (2, 4)
7. 0'2D $\vee$ 0'2C (1, 3)
8. 0'2C (5, 7)
9. $\square$ (6, 8)

Como se obtiene la cláusula vacía ($\square$), entonces F es $\Lambda$-consecuencia lógica de S o, de manera similar, se puede decir que la formula S $\wedge$ F es $\Lambda$= 0'69 válida.

Otra característica de este sistema de programación es la utilización de valores lingüísticos (verdadero, extremadamente probable, altamente improbable, bastante probable,...) en lugar de valores numéricos, aunque el mecanismo de inferencia los trata como valores numéricos puesto que a cada valor lingüístico se le asocia un número.

### 5.3. Intuitionistic Fuzzy Prolog (Atanassov&Georgiev, 93)

En este lenguaje (Atanassov&Georgiev, 93), las proposiciones vagas se valoran estableciendo intervalos, que marcan los límites superior e inferior [v, v'] del grado

de pertenencia de un elemento a un conjunto vago. Igualmente, se define un intervalo de no pertenencia [f, f'], que indica los límites superior e inferior con los que un elemento no pertenece a un conjunto vago. La idea de la lógica intucionista es considerar el intervalo [v, v'] (resp. [f, f']) como intervalo de verdad (resp. falsedad).

Una de las particularidades de este lenguaje es que se define un intervalo de pertenencia y otro de no pertenencia al conjunto vago. Por otra parte, en los lenguajes que hemos visto hasta ahora, se consideraba que el grado de no pertenencia a un conjunto vago era la unidad menos el grado de pertenencia, es decir $\mu_{no.A}(u)=1-\mu_A(u)$. En este caso no es así, lo que permite una mayor flexibilidad al exigir solamente la desigualdad: $0 \leq \mu_{no.A}(u)+\mu_A(u) \leq 1$ o, de manera equivalente: $0 \leq \mu_{no.A}(u)+\mu_A(u)$ y $\mu_{no.A}(u) \leq 1-\mu_A(u)$.

Este hecho está en consonancia con nuestra *intuición* y valoración de la realidad. Supongamos que a una persona de una cierta estatura u, le asociamos un grado de pertenencia al conjunto vago 'Alt', $\mu_{Alt}(u)$. Sin embargo, es posible que este grado, para una misma estatura, no fuera el mismo si conociéramos la información adicional de que "la persona es un hombre" o bien "la persona es una mujer".

Con la información adicional tenemos que $\mu_{no\_Alto}(u)=1-\mu_{Alto}(u)$ (resp. $\mu_{no\_Alta}(u)=1-\mu_{Alta}(u)$) porque existirían dos conjunto vagos: 'Alto' y 'Alta', para los hombres y mujeres respectivamente. Pero cuando no se sabe si la persona es un hombre o una mujer tenemos dos conjuntos vagos posibles. Una manera razonable e intuitiva de tratar este contexto en ausencia de información es flexibilizar la igualdad $\mu_{no.Alt}(u)=1-\mu_{Alt}(u)$ por la inecuación, $\mu_{no.Alt}(u) \leq 1-\mu_{Alt}(u)$.

De esta forma, vemos que la información vaga se puede "combinar" con la ausencia de ciertas informaciones, por ejemplo en nuestro caso particular, el conocimiento de si la persona es un hombre o una mujer. En efecto, supongamos que la estatura de un hombre h (resp. una mujer m) es Estatura(h) (resp. Estatura(m)). Si sabemos que una persona p es un hombre (resp. una mujer) tenemos la igualdad $\mu_{no.Alto}(h)+\mu_{Alto}(h)=1$ (resp. $\mu_{no.Alta}(m)+\mu_{Alta}(m)=1$). Pero si no sabemos que la persona p es un hombre o una mujer entonces podemos recurrir a una función característica del conjunto vago de personas altas 'Alt' que verifique: $\mu_{no.Alt}(p)+\mu_{Alt}(p) \leq 1$.

La idea de los intervalos permite representar expresiones como la siguiente:
El hecho "[0'5, 0'7] [0'1, 0'3] Alto(pepe)" significa que Pepe pertenece al conjunto vago Alto con un intervalo de pertenencia entre 0'5 y 0'7 y con un intervalo de no pertenencia comprendido entre 0'1 y 0'3.

Por tanto, podemos decir que la utilización de los intervalos permiten tratar conjuntamente información vaga, sustentada en la lógica borrosa, conocimiento parcial basado en la inecuación $\mu_{no.Alt}(m)+\mu_{Alt}(m) \leq 1$, imprecisión basada en el empleo de intervalos y no en puntos concretos, y otros aspectos de la incertidumbre.

(Atanassov&Georgiev, 93) proponen una modalidad de resolución basada en el empleo de una lógica de intervalos de la forma:

$$<[\mu, \pi] , [\gamma, \partial]>, \text{ donde } \mu, \pi, \gamma, \partial \in [0,1] \text{ y } \pi+\partial \leq 1.$$

Las cláusulas son denominadas InF-cláusulas *(Intuitionistic Fuzzy Clauses)* formadas por InF-estructuras y presentan el formato general siguiente:

$$[\mu^h, \pi^h] , [\gamma^h, \partial^h] \; P :- Q_1,...,Q_n \; [\mu^b, \pi^b] , [\gamma^b, \partial^b]$$

En (Atanassov&Georgiev, 93) se define una modalidad de la regla de resolución, o más apropiadamente de Modus Ponens, para InF-claúsulas, que determina los intervalos del consecuente P en función de los intervalos deducidos para cada predicado del antecedente Qi. El funcionamiento de este interpretador se describe a continuación:

El problema inicial tiene la forma $Q_1,...,Q_n \; [\mu^1, \pi^1],[\gamma^1, \partial^1]$. Este problema consiste en saber si existen tuplas que pertenecen a los productos cartesianos de cada $Q_i$, $1 \le i \le n$, tal que el grado de pertenencia (o verdad) está entre $[\mu^1, \pi^1]$ y el grado de no pertenencia (o falsedad) está entre $[\gamma^1, \partial^1]$. Un subproblema en un estado avanzado del interpretador en la búsqueda de soluciones del problema tiene la forma:

$$Q_{i(j),j},...,Q_{n(j),j}(V^j,F^j)\#,...,(V^2,F^2)\#Q_{i(1),1},...,Q_{n(1),1}(V^1,F^1)$$

$$\text{donde } V^k = [\mu^k, \pi^k] \text{ y } F^k = [\gamma^k, \partial^k], \; 1 \le k \le j$$

Para reducir este subproblema a otro más sencillo, como se explicó en §2, se deben verificar las condiciones siguientes, las cuales son una adaptación a la lógica intuicionista, del procedimiento clásico explicado en §2.

*Condiciones de aplicación de una inferencia*

1) Primero se busca un hecho P o una regla cuyo consecuente P sea unificable con el primer predicado del subproblema actual $Q_{i(j),j}$.

2) Para poder aplicar la resolución, también se debe cumplir la restricción impuesta a los intervalos de $Q_{i(j),j},...,Q_{n(j),j}(V^j,F^j)$. Concretamente, si $[\mu^h,\pi^h] \; [\gamma^h,\partial^h] \; P$ es el consecuente de la regla aplicada para reducir $Q_{i(j),j}$, los intervalos del antecedente al cual pertenece $(V^j, F^j)$ deben verificar las relaciones:

$$[\mu^h,\pi^h] \subseteq V^j = [\mu^j,\pi^j] \quad \text{y} \quad [\gamma^h,\partial^h] \subseteq F^j = [\gamma^j,\partial^j]$$

*Actualización del subproblema por la aplicación de la inferencia*

3) Según sea una regla o un hecho, existen dos posibilidades:

3.a) Si para la resolución de $Q_{i(j),j}$ se ha aplicado una regla cuyo consecuente P es un predicado vago, entonces se reemplaza $Q_{i(j),j}$ del subproblema por el conjunto de predicados vagos del antecedente de la regla concatenados por sus intervalos antecedentes.

3.b) Si para la resolución se ha utilizado un hecho, entonces simplemente se extrae $Q_{i(j),j}$ del antecedente.

4) Se aplica la substitución al subproblema obtenido en el primer paso (1).

5) Cuando el conjunto de predicados vagos del antecedente de la regla Rj de (3.a), que es vacío en el caso de aplicar un hecho (3.b), ha sido resuelto, entonces se procede a substituir los intervalos previos en el subproblema $[\mu j\text{-}1, \pi j\text{-}1]$ y $[\gamma j\text{-}1, \partial j\text{-}1]$ por el intervalo resultante del consecuente obtenido al aplicar la regla:

$$Rj: \quad [\mu_j,\pi_j],[\gamma_i,\partial_i]\ P_i :- Q_{1,j},\ldots,Q_{i,j},\ldots,Q_{n(j),j}\ [\mu j,\pi j],[\gamma j,\partial j]$$

(Los intervalos del consecuente se distinguen de los del antecedente porque los primeros tienen supra-índices mientras que los segundos tienen sub-índices.)

Por tanto, de un subproblema pasamos a otro más sencillo, en el sentido de aproximarse a una solución (si existe) -ver §-2, mediante la aplicación de las etapas (1)-(5).

Los intervalos del consecuente obtenidos son función de los intervalos del consecuente de la regla declarados $[\mu_j,\pi_j],[\gamma_j,\partial_j]$ y de los intervalos obtenidos para cada predicado del antecedente.

Los intervalos de un átomo del cuerpo Q son igualmente intervalos de medidas de pertenencia (verdad) y no pertenencia (falsedad) a un conjunto vago. Para que un consecuente P de una regla sea deducible, tiene que verificarse que:

Para cada Q del antecedente de la fórmula existe un hecho o consecuente (V F P) deducido previamente con intervalos V de pertenencia (o Verdad) y F de no pertenencia (o Falsedad) tales que:

$$V \cap [\mu^b, \pi^b] = [\mu^{(V,b)}, \pi^{(V,b)}] \neq \varnothing$$
$$F \cap [\gamma^b, \partial^b] = [\gamma^{(F,b)}, \partial^{(F,b)}] \neq \varnothing$$

En la segunda etapa se calculan los intervalos de pertenencia y no pertenencia (verdad y falsedad propiamente) de todo el cuerpo de la cláusula. Así, se comienza por hallar el intervalo para cada elemento de Qi. Como pueden existir J diferentes hechos o átomos deducidos unificables con cada Qi del cuerpo de la cláusula, es decir, diferentes J encadenamientos posibles para probar Qi, aplicamos la regla de la disyunción propuesta por los autores:

$$[\mu_i^{(V,b)}, \pi_i^{(F,b)}] = [\ \text{Max}\ \{\mu_j^{(V,b)}: 1\leq j\leq J\}\ ;\ \text{Min}\ \{\gamma_j^{(F,b)}: 1\leq j \leq J\}]$$
$$[\gamma_i^{(V,b)}, \partial_i^{(F,b)}] = [\ \text{Max}\ \{\gamma_j^{(V,b)}: 1\leq j\leq J\}\ ;\ \text{Min}\ \{\partial_j^{(F,b)}: 1\leq j \leq J\}]$$

Seguidamente, para obtener los valores límites de los intervalos de pertenencia y no pertenencia (o bien, verdad y falsedad) de todo el cuerpo de la regla, aplicamos el operador de la conjunción:

$$[\mu^{(V,B)}, \gamma^{(V,B)}] = [\ \text{Min}\ \{\mu_i^{(V,b)} : 1\leq i \leq n\}\ ;\ \text{Max}\ \{\gamma_i^{(V,b)} : 1\leq i \leq n\}\ ]$$
$$[\pi^{(V,B)}, \partial^{(V,B)}] = [\ \text{Min}\ \{\pi_i^{(V,b)} : 1\leq s \leq n\}\ ;\ \text{Max}\ \{\partial_i^{(V,b)} : 1\leq i \leq n\}\ ]$$

6) Determinados los intervalos vagos de todos los átomos del antecedente y dado el intervalo del consecuente de la regla, los valores $<\mu H, \gamma H>$ del consecuente deducido se calculan mediante las fórmulas siguientes:

$$(1)\ \mu H \text{ <-- } \mu^h + \alpha\mu\ (\pi^h - \mu^h) \qquad (2)\ \gamma H \text{ <-- } \gamma^h + \alpha\gamma\ (\partial^h - \gamma^h)$$

*donde:*

Si $\pi^b > \mu^b$ entonces: $\alpha\mu = (\pi^{(V,B)} - \mu^b) \div (\pi^b - \mu^b)$
en otro caso: $\alpha\mu = 1/2$

Si $\partial^b > \gamma^b$ entonces: $\alpha\gamma = ( \partial^{(V,B)} - \gamma^b ) \div (\partial^b - \gamma^b)$
en otro caso: $\alpha\gamma = 1/2$

Los cálculos de los dos otros coeficientes $\pi H$ y $\partial H$ son similares.

El intervalo $< \mu H, \gamma H >$ significa que el grado de pertenencia (o grado de verdad) de una tupla $t_1,...,t_n$ al producto cartesiano de los dominios y universos de los argumentos de P, es como mínimo $\mu H$ y el valor de no pertenencia (o grado de falsedad) es como máximo $\gamma H$.

*Ejemplo:*

El siguiente programa InF-Prolog consta de las siguientes reglas y hechos:

Reglas:

```
( 1 )   [0'6, 0'8] [0'1, 0'2]   d(X) :- p(X), l(X) [0'4, 0'7] [0, 0'2].
( 2 )   [0'4, 0'7] [0'1, 0'2]   d(X) :- c(X) [0'5, 0'8] [0, 0'1].
( 3 )   [0'5, 0'8] [0'15, 0'2]   p(X) :- e(X), r(X) [0'3, 0'75] [0, 0'2].
```

Hechos:

```
( 4 )   [0'7]  [0'2]  r(a).
( 5 )   [0'8]  [0'2]  l(a).
( 6 )   [0'6]  [0'1]  c(a).
( 7 )   [0'6]  [0'2]  e(a).
( 8 )   [0'8]  [0'1]  r(b).
( 9 )   [0'9]  [0'0]  l(b).
( 1 0 )  [0'9]  [0'1]  e(b).
( 1 1 )  [0'6]  [0'3]  c(b).
```

Una cláusula de tipo [0'7] [0'2] r(a) considera sus valores 0'7 y 0'2 como umbrales. Es decir, r(a) es satisfecha con un valor de pertenencia (o verdad) mínimo de 0'7 y un valor de no pertenencia (o falsedad) máximo de 0'2.

Supongamos ahora que deseamos conocer alguna solución de d(X) y con qué grado mínimo de pertenencia (verdad) y grado máximo de no pertenencia (falsedad), es decir:

$$?- d(X), < \mu H, \gamma H >$$

La interpretación de esta pregunta es la siguiente. Como es usual en los intérpretes de Prolog basados en la estrategia SLD, la primera cláusula examinada es aquella que contiene el objetivo, es decir, la número 1. Supongamos por un momento que el subobjetivo p(X) está probado. El siguiente subobjetivo es l(X). Pero aunque se encuentran dos unificaciones posibles para l(X), no puede ser probado porque las restricciones que indican el intervalo de grados posibles de pertenencia [0'4, 0'7], no

incluyen ni el de I(a) que es ≥0'8 ni el de I(b) que es ≥0'9. Por tanto, la primera cláusula aplicable conduce a una situación de fracaso.

El siguiente paso es considerar la segunda cláusula. La sustitución c(a) tiene grados de verdad <0'6, 0'1>, que pertenecen a los intervalos [0'5, 0'8] y [0, 0'1]. Por tanto, puede calcularse la valoración de d(X) con las operaciones sobre intervalos dadas y sustituyendo las variables por sus respectivas instanciaciones se obtiene la respuesta:

$$X = a\ [0'85,\ 0'15].$$

## 6. Para concluir

Aunque el trabajo pionero de Lee en 1972 le dio ya una relativa relevancia a las investigaciones sobre PLB, ésta no ha alcanzado todavía una fase de solidez y uniformidad comparable a la del Prolog estándar. La existencia de diferentes tipos de vaguedad y, en consecuencia, de diferentes lógicas para representar la vaguedad, hace que el diseño de interpretadores y compiladores de Prolog vago sean una empresa compleja y difícil.

Los fuzzy Prologs tienen inmediata aplicación en las bases de datos relacionales y deductivas, las cuales frecuentemente contienen atributos vagos. Otra aplicación directa se encuentra en los sistemas expertos, que utilizan un lenguaje atributo-objeto-valor.

La ausencia de compiladores de fuzzy Prolog y de compañías de software interesadas en su diseño y producción comercial, repercute en el número de validaciones prácticas de lo que se está desarrollando de modo teórico. En este ámbito, como en tantos otros, las ideas teóricas se verían beneficiadas por las sugerencias prácticas que los usuarios de estos tipos de software pudiesen hacer.

Además de los sistemas de PLB presentados aquí, que consideramos representativos de los trabajos existentes en esta área, en la bibliografía existen otros tipos de Prolog basados en otras lógicas y algunas de sus correspondientes referencias son: Prolog posibilístico (Dubois et al., 91), Prolog en teoría evidencial (Baldwin et al., 1995), Prolog modal (Fariñas del Cerro, 1986), Prolog en lógicas multi-valoradas (Rescher, 69; Tamburrini, Termini, 92; Escalada-Imaz & Manyà-Serrés, 1995; Escalada-Imaz & Manyà-Serrés, 1996), etc.

## Agradecimientos

Este trabajo ha sido financiado por los proyectos DISCOR: TIC 94-0847-C02-01) de la CICYT (Institut d'Investigació en Intel.ligència Artificial, CSIC); 94-13 de la Universitat de Lleida y PB92-0394 de la DGICYT (Univ. de Santiago de Compostela).

## BIBLIOGRAFIA

Atanassov, K., Georgiev, C.: 1993, 'Intuitionistic Fuzzy Prolog', *Fuzzy Sets and Systems* 53, 121-129.

Baldwin, J.F., Martin, T.P., Pilsworth, B.W.: 1995, *Fril: Fuzzy and Evidential Reasoning in Artificial Intelligence*, John Wiley & Sons.

Bratko, I.: 1990, *Prolog Programming for Artificial Intelligence (2nd. ed.)*, Addison-Wesley.

Clocksin, W., Mellish, C., 1981,*Programming in Prolog*, Springer-Verlag.

Dubois, D., Jang, J., Prade, H.: 1991, 'Towards Possibilistic Logic Programming', *Proc. Eighth International Conference on Logic Programming*, Paris.

Escalada-Imaz, G., Manyà-Serres, F.: 1995, 'Efficient Interpretation of Propositional Multiple-valued Logic Programs', *Lecture Notes in Computer Science. Advances in Intelligent Computing*,. Springer-Verlag, Vol. 945, pp. 428-439.

Escalada-Imaz, G., Manyà-Serres, F.: 1996, 'On Multiple-valued Logic Programming', in Dahl, V., Sobrino, A. (eds.): *Estudios sobre programación lógica y sus aplicaciones*, Publicaciones de la Universidad de Santiago de Compostela, 387-419.

Fariñas del Cerro, L.: 1986, 'Molog: A system that extends Prolog with modal Logic', *New Generation Computing*.

Ishizuka, M., Naoki, K: 1985, 'Prolog-Elf incorporating fuzzy logic', *Proceedings of the 9th Inter. Joint Conf. on Artificial Intelligence (IJCAI 85)*, Los Angeles, U.S.A., 701-703.

Lee, R.C.T.: 1972, 'Fuzzy Logic and the Resolution Principle', *Journal of the Association for Computing Machinery* 19/1, 109-119.

Li, D., Liu, G.: 1990, *A Fuzzy Prolog Database System*, Research Studies Press and John Wiley and Sons.

Liu, D.: 1992, 'A Fuzzy Linguistic Prolog', *Proceedings of the V International Congress on Knowledge Engineering*, Sevilla, pp. 126-130.

Lloyd, J.W.: 1987, *Foundations of Logic Programming*, Springer-Verlag.

Martin, T.P., Baldwin, J.F., Pilsworth, B.W.: 1987, 'The implementation of fprolog: A fuzzy prolog interpreter', *Fuzzy Sets and Systems* 23, 119-129.

Mukaidono, M., Shen, Z., Ding, L.: 1989, 'Fundamentals of fuzzy prolog', *International Journal of Approximated Reasoning* 3, 179-193.

Rescher, N.: 1969, *Many-valued Logics*, McGraw-Hill.

Robinson, J.A.: 1965, 'A machine oriented logic based on the resolution principle', *Journal of the Association for Computing Machinery* 12/1, 23–41.

Shen, Z., Ding, L., Mukaidono, M.: 1988, 'A Theoretical Framework of Fuzzy Prolog', in Gupta, M.M., Yamakawa, T. (eds.): *Fuzzy Computing: Theory, Hardware and Applications*, Amsterdam, North-Holland, 89-100.

Tamburrini, G., Termini, S.: 1992, 'Towards a resolution in a fuzzy logic with Lukasiewicz implication', *Proceedings of IPMU'92*, 271-277.

Umano, M.: 1987, 'Fuzzy Sets Prolog', *Preprints of the 2nd. Inter. Fuzzy Systems Assoc. (IFSA) Congress*, Tokyo, Japan, 750-753.

Weigert, T.J., Tsai, J., Liu, X.: 1993, 'Fuzzy Operator Logic and Fuzzy Resolution', *Journal of Automated Reasoning* 10, 59-78.

Zadeh, L.A.: 1987, *Fuzzy Sets and Applications. Selected Papers* by *L.A. Zadeh*, edited by R.R. Yager, S. Ovchinnikov, R.M. Tong and H.T. Nguyen, John Wiley & Sons.

**Gonzalo Escalada-Imaz** was Assistant Professor of the University of Toulouse, Doctorate Professor of the Polytechnic University of Catalonia, Doctorate Professor of the Autonomous University of Barcelona and Master Professor of the CINVESTAV-IPN, México, D.F. At present is Researcher at the Artificial Intelligence Research Institute, Barcelona. His current research areas of interest include automated deduction and optimal algorithms design.

**Felip Manyà** received the degree in Computer Science from the Universitat Autònoma de Barcelona in 1987. He joined the Centre d'Estudis Avançats de Blanes (CSIC) in 1989, where he was involved in an european SPRIT project, as well as in several Spanish CICYT projects. Since 1992 is lecturer of Computational Logic at the Universitat de Lleida. His current research interest include automated deduction and logic programming in many-valued logics.

**Alejandro Sobrino** received the Ph. D. degree in Philosophy from the University of Santiago de Compostela, where he is currently an Associate Professor at the Department of Logic and Philosophy of Science. Member of the Spanish Society of Logic and Philosophy of Science and Vocal of the Spanish Association of Fuzzy Logic and Technologies (FLAT), he is also member of the ACM. His current areas of interest are logic and natural language processing with emphasis on vague and imprecise information.