Un estudio sobre la inclusión de Conjuntos Borrosos en Prolog

Norma Verónica Ramírez Pérez

Instituto Tecnológico de Celaya

norma.ramirez@itcelaya.edu.mx

Martín Laguna Estrada

Instituto Tecnológico de Celaya

martin.laguna@itcelaya.edu.mx

Resumen

Este artículo presenta un estudio de la implementación de Prolog Borrosos, su semántica y sintáctica, con la finalidad de hacer una difusión sobre la combinación de la programación lógica con lógica borrosa que permiten la inclusión de información con razonamiento impreciso e incierto.

Palabras clave: Programación lógica, Prolog, lógica borrosa, razonamiento impreciso.

1. Introducción

La programación lógica ha sido considerada como una herramienta de representación del conocimiento utilizada en la inteligencia artificial, derivando en algunas implementaciones como: Sictus Prolog, Swi-Prolog, Ciao-Prolog y GNU-Prolog entre otras. Sin embargo, presenta algunas limitaciones para adaptarse al razonamiento impreciso. Paralelamente al desarrollo de la programación lógica, investigadores en el área de inteligencia artificial han continuado produciendo nuevos formalismos con el fin de obtener métodos de razonamiento y de representación de conocimiento. Esto ha motivado a realizar una combinación de la programación lógica y la lógica borrosa, que permitan la representación de la información incompleta e inconsistente y que capture aspectos del razonamiento impreciso que son difíciles de expresar en la programación lógica tradicional. Recientemente ha surgido el interés por diseñar lenguajes

declarativos que incluyan características de la lógica borrosa e incorporen entre sus recursos expresivos la posibilidad de tratar la información imprecisa de forma natural.

2. Definiciones, conceptos y teorías

2.1 Prolog

Prolog (PROgramming in LOGic) [3] es el primer lenguaje de programación para la programacion lógica, fue desarrollado en Marsella, Francia, iniciando con Alain Coulmeauer y Philippe Roussel. La primera implementación de Prolog se completó en Edimburgo en 1972 usando el compilador de ALGOL y los aspectos básicos del lenguaje actual que concluyeron en 1973. Como consecuencia de este lenguaje, surgieron nuevos intérpretes, todos ellos derivados del Prolog original. Entre éstos podemos encontrar SWI-Prolog[34], CIAO-Prolog[2], GNU-Prolog[10], e incluso algunos intérpretes comerciales, como el Sictus Prolog[33]. La característica principal de estos intérpretes es su sintaxis relativamente simple, caracterizada por su tratamiento generalizado de las variables, constantes y valores como “objetos” opacos. Es decir, no se especifica explícitamente el dominio de los objetos, los cuales se pueden manipular de manera intercambiable en la mayor parte de los casos.

Prolog es un lenguaje de programación declarativo que se diferencia de los lenguajes imperativos o procedurales [28], porque está basado en formalismos abstractos. 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 de las reglas, están cuantificadas universalmente, un ejemplo de ello es el programa Prolog con una regla y dos hechos.

$$\textit{Regla:}\quad \forall X, \forall Y, \forall Z, padre(X,Y) \wedge padre\ (Y,Z) \rightarrow abuelo(X,Z)$$

$$\textit{Hechos:}\quad padre(juan, luis).$$

$$padre(luis, pedro).$$

En la sintaxis de Prolog se omiten los cuantificadores y se sobreentiende que todas las variables están cuantificadas universalmente; el signo de implicaciones 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, mientras que las variables se escriben en mayúsculas y las constantes en minúsculas. Por tanto, el ejemplo anterior, se expresa en Prolog de la siguiente manera:

$$Regla: \quad abuelo(X,Z), padre(X,Y), padre\ (Y,Z)$$

$$Hechos: \quad padre(juan, luis).$$

$$padre(luis, pedro).$$

La introducción de este programa, permite realizar preguntas que corresponden a una conjunción de predicados cuyas variables están cuantificadas existencialmente. En Prolog, una pregunta es representada por un signo de interrogación seguido de uno o más predicados separados por comas. Si retomamos el ejemplo anterior, al hacer la pregunta de quién es el abuelo de pedro, ésta estaría representada de la siguiente manera$?\ abuelo(X, pedro)$. Prolog nos respondería "$X\ es\ Juan$", es decir la única respuesta correcta. Pero si se hace la pregunta $?abuelo(X,Y)$nos respondería "$X = juan;\ Y = pedro$", recordando que equivale a preguntar si existe X y algún Y, tales que X es abuelo de Y. Si la pregunta fuera $?padre(X,Y),$ obtendríamos dos respuestas válidas

$$(X = juan, Y = luis)\ y\ (X = luis, Y = pedro).$$

En los ejemplos que se han expuesto, sólo se han considerado variables y constantes, en general, un programa de Prolog puede contener términos de primer orden $t_{1,\ldots,}t_n$ en sus predicados, por ejemplo:

$$pariente\big(nieto(Z,X)\big):-hijo(Z,Y), hijo(Y,X)$$

$$pariente\big(tio(Z,X)\big):-hijo(Y,X), hermano(Y,Z)$$

$$pariente\big(sobrino(X,Z)\big):-pariente\,(tio,(Z,X))$$

De lo anteriormente expuesto podemos observar que tiene los siguientes términos:
$nieto(Z,X)$, $tio(Z,X)$, $sobrino(X,Z)$

Prolog proporciona dos puntos relevantes importantes para diseñar un interpretador eficiente y así obtener rápidamente las soluciones de un problema si existieran: la ejecución de la inferencia y la estrategia de encadenamiento de inferencias:

1. En la etapa de inferencia (unificación) interviene una operación de unificación que se define brevemente de la siguiente manera: teniendo dos predicados $P(t_{1,\ldots\ldots,}t_n)$ y $Q(s_{1,\ldots\ldots,}s_n)$, de los cuales se pretende realizar una sustitución $\sigma = \{X_1/r_1,..,X_k/r_k\}$ llamado mgu (unificador más general), que verifica lo siguiente:
    a) Si se sustituye en los predicados $P(t_{1,\ldots\ldots,}t_n)$ y $Q(s_{1,\ldots\ldots,}s_n)$, cada variable X_i por el término r_i indicando la substitución σ, entonces los dos predicados son sintácticamente iguales, es decir, $P(t_{1,\ldots\ldots,}t_n).\sigma$ y $Q(s_{1,\ldots\ldots,}s_n).\sigma$.
    b) Se puede obtener cualquier otro unificador σ' de $P(t_{1,\ldots\ldots,}t_n)$ y $Q(s_{1,\ldots\ldots,}s_n)$, por una operación de composición entre σ y una tercera substitución p, i.e. $\sigma' = \sigma * p$, más información [23].

La unificación es considerada una operación para reducir el tiempo que el interpretador necesita para encontrar soluciones, ya que interviene en cada inferencia. En la actualidad existen muchos algoritmos de unificación que han sido diseñados para obtener rápidamente un unificador más general.

Definida la unificación, se puede 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:

a) Se elige un átomo Q de la conjunción actual de los predicados y un hecho P, o una regla con el consecuente P si Q y P son unificables.
b) Obtener σ su unificador más general.
c) La conjunción de predicados se transforma, extrayendo Q si P es un hecho, o bien, reemplazando el predicado Q por el conjunto de predicados

antecedentes de la regla. El último paso consiste en aplicar σ al conjunto de átomos de la conjunción actual.

2. En la estrategia de búsqueda (SLD)[19,31], es considerado como 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 objetivo es el encadenar inferencias para obtener soluciones al problema que le hayan propuesto, considerando que una de sus propiedades del interpretador es que si existen soluciones al problema planteado, el encadenamiento de inferencias deberá ser tal que siempre encuentre todas las soluciones posibles, por lo que recíprocamente, cada respuesta aportada por el interpretador como una solución, debe ser realmente una solución del problema declarado en Prolog.

   Los interpretadores de Prolog están basados en un refinamiento de resoluciones SLD, donde 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, es decir, en ser resuelto).

2.2 Programación lógica con incertidumbre

La investigación sobre la incertidumbre en el marco de la programación lógica se ha producido a lo largo de los últimos años, se han tratado de diversas aproximaciones a la semántica declarativa y operacional, así como a la aplicación práctica de estos enfoques a la programación lógica con soporte para la incertidumbre y el razonamiento incierto.

L.A. Zadeh [40,41], definió la teoría de los conjuntos borrosos como una extensión de la teoría matemática de conjuntos, en la que la función de pertenencia de un conjunto, indicaba el grado de pertenencia de un elemento al conjunto mediante la asignación de un número real comprendido en el intervalo [0,1].

El uso de la incertidumbre en el ámbito de la lógica matemática y la programación, se ha globalizado en diferentes formas de programación lógica y teorías que se mencionan a continuación:

- Programación lógica multivaluada [24-27], donde los dos valores de verdad clásicos de falso o verdadero, son reemplazados por toda una serie de valores de verdad, y para los que se proponen una multitud de conectivas y agregadores lógicos que permitan interpretar de manera diferente, las cláusulas de un programa.
- Teoría de Posibilidad [4-6,16,30, 38], el cómputo de los grados de certidumbre se realiza atendiendo a la teoría de la posibilidad, un sistema deductivo que, siguiendo la idea de [29], se basa en la idea de que la fuerza de una conclusión coincide con la fuerza de su argumento más débil. Éstos guardan bastante relación con los enfoques anotados con los que comparten origen, y con los enfoques multi-valuados en que existen propuestas posibilistas que amplían el número de valores de verdad.
- Programación lógica con relaciones de similaridad SLP (Similarity Logic Programming) presentado en [31]. Este enfoque utiliza también el retículo [0,1] para tratar la incertidumbre al estilo de la programación lógica anotada, o de la programación lógica multi-valuada.
- Programación lógica con relaciones de proximidad [32]. Las relaciones de proximidad se utilizan para modelar la imprecisión derivada de la probabilidad de desigualdad entre dos valores en el dominio de discurso.

Todos estos tipos de programación y teorías, han sido empleados por autores que se han dedicado a la introducción y representación de conjuntos borrosos en PROLOG, o bien extenderlo, utilizando su motor de inferencia. Por motivos de espacio, sólo se describen la teoría de conjuntos borrosos, por lo que si el lector interesado en estos temas quiere saber más sobre estas lógicas y teorías, sugerimos vean la bibliografía indicada al final de este estudio.

2.3 Teoría de conjuntos borrosos

La teoría de conjuntos borrosos tiene su origen en la lógica booleana de los conjuntos. Dicha teoría fue extendida con los trabajos de Zadeh en [39], que introdujo la lógica borrosa, y la teoría de los conjuntos borrosos. Su razón de ser obedece a la necesidad

de tratar problemas con los conceptos de valores de verdad parcial dentro de los términos de absoluta verdad o certeza y de falsedad completa, en problemas que tratan de describir los fenómenos de la vaguedad y la ambigüedad. La teoría de los conjuntos borrosos se refiere en su aspecto más elemental, a la representación de la realidad en clases, categorías, estados de cosas o conjuntos, mediante funciones de pertenencia o membresía, que miden el grado de asociación de un determinado elemento, a una agrupación o clase dentro de las condiciones de pertenencia y no pertenencia absolutas.

Un conjunto borroso es una pieza de información con incertidumbre, que está especificada mediante una función que mapea el grado de pertenencia o certeza con la que se asocia cierto elemento a las diferentes categorías, atributos, estados de cosas o conjuntos que conforman el dominio. Un conjunto borroso es, a su vez, información que contiene datos objetivos y subjetivos. La información objetiva es el conjunto de elementos posibles que representan una determinada entidad variable, mientras que la información subjetiva está asociada a una función de pertenencia que define el grado de creencia, certeza, soporte, confianza, evidencia o preferencia sobre determinados valores del conjunto de datos posibles correspondientes a una variable.

De manera formal, un dato o información borrosa puede ser considerado como un conjunto borroso U para el cual los valores x asociados al dato, exhiben un grado de pertenencia expresada mediante una función μ(x) de tal forma que $0 \leq \mu(x) \leq 1$ donde:

$$F(x) = \{(x, \mu(x)) | x \in X\}$$

Y donde $(x, \mu(x))$, para todo x que pertenence a *X* representa la función de pertenencia y *X* es el dominio de la función $\mu(x)$ y del conjunto *x* [20].

3. Evolución y diversificación de Prolog borrosos

La programación lógica es importante en el desarrollo de sistemas en el área de inteligencia artificial, y uno de los lenguajes más eficientes y utilizado en este paradigma es el lenguaje Prolog descrito brevemente en el apartado 2 de este trabajo. Este

lenguaje originó que surgieran implementaciones como: Sictus-Prolog [33], Swi-Prolog [34], Ciao-Prolog [2], GNU-Prolog [10], que a su vez han sido utilizadas para el desarrollo de extensiones o módulos que permiten manejar información con incertidumbre.

En el manejo de la información con incertidumbre, Klir y Folger [17,18] diferenciaron dos etapas en la evolución de conocimiento, en la primera de ellas, el esfuerzo orientado a conocer aspectos del mundo y la segunda por conocer aspectos del propio conocimiento. Se puede suponer que esta segunda etapa, surge a consecuencia de fallos de la primera, para delimitar el alcance y validez del conocimiento adquirido previamente, donde la preocupación no se centra en la mera adquisición del conocimiento, sino que se intenta determinar en qué medida se conoce algo y con qué grado de certeza se puede asignar al conocimiento.

Los compiladores de Prolog se han distinguido por su versatibilidad y potencialidad en su mecanismo de inferencia. Sin embargo, surge la necesidad de tratar información con incertidumbre, lo que motiva a que se implementen nuevos sistemas que realicen este tratamiento, que requieren técnicas especiales para el modelamiento de esta información.

Prolog es considerado un lenguaje de programación muy útil para resolver problemas que implican objetos y relaciones entre ellos, basado en mecanismos como: unificación, estructura de datos basados en árboles y backtranking automático. Su sintaxis consiste en declarar hechos, relaciones y reglas sobre objetos. Esto ha permitido realizar algunas extensiones que trabajan directamente con los compiladores como swi-prolog, sictus-prolog y ciao-prolog, integrando módulos que permitan trabajar con información vaga e imprecisa.

Por las características de la programación lógica y lógica borrosa, Prolog permite el uso de estas dos áreas que hoy en día tienen entidad propia y han sido utilizadas para la resolución de problemas complejos que tengan cierto "*grado de inteligencia*", y que da origen al diseño de un sistema deductivo eficiente para la programación lógica borrosa. El primer artículo relacionado con este tema fue publicado a principios de los años 70´s, que escribió Lee [21], después en los años 80's fue propuesto 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 manera, nace una nueva línea de investigación donde se incluye la programación lógica y la lógica borrosa.

En el año de 1987 surgió el Sistema Prolog implementado por Martin B. & Pilsworth, mientras que en 1990 nace el sistema f-Prolog implementado por Li&Liu, basado en la lógica min-max. Este sistema, junto con el FPROLOG, tiene un gran parecido en su sintaxis. En 1987 fue implementado Fuzzy Sets Prolog por Umano, mientras que Ciao-Fuzzy-Prolog fue implementado por C. Vaucheret, S. Guadarrama, and S. Muñoz, en 2007. Vale la pena aclarar que Ciao Prolog fue implementado desde los años 90's, y a la fecha se le siguen añadiendo algunos módulos. Desde entonces, las implementaciones más recientes han sido por Bousi-Prolog por P. Julián-Iranzo, C. Rubio, y J. Gallardo, Universidad Castilla de la Mancha y Floper implementado a partir del 2007, por Pedro J. Morcillo y Ginés Moreno, de la Universidad Castilla de la Mancha.

3.1 Prolog ELF

El lenguaje prolog ELF, implementado por Mitsuru Ishizuka y Naoli Kanai de la Universidad de Tokio [15], incorpora la lógica borrosa y algunas funciones en el lenguaje PROLOG, para la construcción de sistemas de conocimiento con incertidumbre, además de sus aserciones de valores de verdad que pueden estar entre 1.0, 0.5 y 0 (para algunos casos excepcionales). Este sistema es de uso general debido a que hereda todas las características básicas de PROLOG y fue desarrollado en Pascal.

El Prolog-ELF es un sistema basado en la teoría propuesta por Lee en 1972 [21]. 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 borrosa puede tener uno o varios predicados borrosos. Así $T(C)$ es un valor global, que implícitamente es función de los grados de pertenencia relativa de las diferentes tuplas de los predicados borrosos declarados en C. En este caso, y desde un punto de vista lógico, substituyen la lógica bivaluada por la lógica borrosa max-min, definida como sigue:

- 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 máximo de sus conjuntivos.
- El valor de la interpretación de la disyunción es el máximo de sus disyuntivos.
- El valor de la interpretación de la negación de la función complemento.

3.2 Sistema FPROLOG

El sistema FPROLOG fue implementado por Martin B, & Pilsworth en 1987 [1]. Este sistema puede ser comparado con Prolog-ELF desde el punto de vista de su capacidad representativa. Sin embargo este sistema sólo permite la asignación de valores a los hechos, y a las reglas se les asocia el valor de 1.

En este sistema, los conjuntos borrosos A están representados por atributos borrosos A (se utilizó la misma notación para el conjunto borroso y para el atributo que lo modela, evitando la ambigüedad en el contexto en el que son utilizados). Y un conjunto de hechos $(A(u);\ \partial)$, donde u pertenece al universo de discurso U de A y ∂ al intervalo real unitario [0,1]. En cada par $(A(u);\ \partial)$, ∂ 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.

Algunas características de FPROLOG:

1. El sistema le ofrece al programador la posibilidad de utilizar otras funciones distintas al mínimo.
2. La negación en el sistema, puede deducirse de la siguiente manera: si un predicado P puede deducirse con un valor de verdad 1- α, entonces $no(P)$ se deduce con un valor de verdad α.
3. Fue implementado en Franz Lisps y su sintaxis está basada en listas en lugar de utilizar la conocida sintaxis de Edimburgo.
4. Trata de manera clara y explícita la vaguedad existente en un programa. Frecuentemente los predicados borrosos son, o se reducen, a un conjunto de predicados borrosos unarios, los cuales tienen un único universo U sobre el cual se definen conjuntos borrosos. 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 considerada vaga.

3.3 Sistema f-Prolog

LI & Liu implementaron el sistema f-Prolog, en 1990 [22], está basado en la lógica max/min. Un programa está formado por hechos-V y reglas-V (donde la V denota la vaguedad presente en los hechos y en las reglas):

1. Un hecho-V es un par de la forma $P(t1, \dots tn) : -[V]$, donde $P(t1, \dots tn)$ es un predicado borroso 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 t_i definidos sobre conjuntos borrosos A_i. El valor V indica el grado de pertenencia de una tupla n-aria $(s1, \dots sj), j \leq n$, al producto cartesiano $(Ax_1 \dots xA_j)$ de los conjuntos borrosos definidos en los argumentos de P. En general, se tiende a transcribir los hechos $-V$ como hechos borrosos unarios-V. El valor asignado a un hecho borroso unario-V es $\mu A(u)$, correspondiente al grado de pertenencia de u al conjunto borroso A definido sobre su único universo. Así, los conjuntos borrosos 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

$$P: -[V] - Q_1, \dots., Q_i, \dots Q_n,$$

Donde $P, Q_1, \dots., Q_i, \dots Q_n$ son predicados, siendo al menos uno, un predicado borroso, y V es un valor del intervalo real [0,1]. Si el valor numérico que modela la borrosidad en los predicados Q_i es $\mu(Q_i), 1 \leq i \leq n$, entonces el valor numérico de la conjunción de $Q1, \dots., Qi, \dots Qn$ se define como el mínimo de los valores

$\mu(Qi)\ 1 \leq i \leq n$, y lo denotaremos por $\mu(Q)$. El valor numérico de la conclusión se define como $\mu(P) = Vx\mu(Q)$. Esta ecuación significa que el grado de borrosidad del consecuente P aumenta proporcionalmente, ya que $V\epsilon\ [0,1]$, con respecto el grado de vaguedad de la conjunción del antecedente. Como en los hechos-V, los predicados utilizados en las reglas son en general predicados borrosos unitarios.

3.4 Sistema Fuzzy Sets Prolog

Fuzzy sets Prolog fue creado por Umano en 1987 [35]. Este sistema permite asociar hechos y reglas sus correspondientes valores numéricos de vaguedad, permiten también cuantificadores lingüísticos, tales como “cierto”, “bastante cierto”, etc., que son expresados por conjuntos borrosos. Utilizan predicados borrosos donde los términos de sus argumentos, formados por símbolos de función, de variables y de constantes, corresponden a funciones, variables y constantes borrosas. Para ilustrar la lógica borrosa y los mecanismos inferenciales, se ejemplifica el caso simple de la lógica proposicional borrosa. Una proposición borrosa, como “unos 26 años“, queda caracterizada por la función característica del conjunto borroso que le corresponde. Los cuantificadores borrosos, del tipo “es más o menos cierto”; “es cierto”, etc., se expresan también por medio de conjuntos borrosos.

3.5 Sistema Ciao Fuzzy Prolog

Implementación realizada por C. Vaucheret, S. Guadarrama [11,36], trabaja con un módulo llamado “fuzzy” con el motor de inferencia de Ciao-Prolog, funciona también con Swi-Prolog[34], Sictus Prolog[33], su programación está basada en la mezcla de dos paradigmas declarativos: la solución de restricciones y la programación lógica. Añaden la incertidumbre con CLP(R) en lugar de implementar una nueva resolución borrosa. Su enfoque está basado en dos aspectos:

1. Los valores de verdad son subintervalos [0,1], esto es, una unión finita de subintervalos. Tener un único valor, es un caso particular de modelar con un intervalo unitario.

2. Un valor de verdad es propagado a través de reglas por medio de un operador de agregación. La definición de un operador de agregación, es una generalización que subsume operadores como: conjuntivo (norma triangular [18], como min, prod, etc.), operadores disyuntivos (co-norma triangular, como max, sum, etc.), operadores promedio (como media aritmética, media cuasilineal, etc.) y operadores híbridos (combinación de los operadores anteriores).

3.6 Sistema BOUSI~ Prolog

BOUSI~Prolog, implementado por P. Julián-Iranzo, C. Rubio, and J. Gallardo, Universidad Castilla de la Mancha 2009 [12-14]. Es un lenguaje de programación lógica borrosa en donde el objetivo primordial es flexibilizar el proceso de respuestas a preguntas. Su mecanismo operacional es una extensión del principio de resolución SLD [31], que denominan resolución SLD débil, donde la unificación clásica se ha sustituido por un algoritmo de unificación borrosa, especificando un método genérico para la unificación de conjuntos borrosos que también es aplicable a otros lenguajes que se apoyen en un mecanismo de resolución débil. Posteriormente, esta relación borrosa se emplea de forma estándar y completamente integrada en el mecanismo de unificación borrosa del sistema Bousi~Prolog, lo que permite tratar las etiquetas lingüísticas en pie de igualdad, con otras constantes definidas en el código fuente de un programa. Consideran que su implementación es novedosa ya que es la primera vez que se introducen los conjuntos borrosos en el núcleo de un sistema Prolog mediante el empleo de relaciones borrosas; además de que la inclusión la hacen de manera muy natural, sin afectar la semántica operacional de su lenguaje Bousi~Prolog y lo hacen con apenas algunas modificaciones sintácticas, por lo que consideran que es una buena alternativa para ser empleada en otros sistemas Prolog borrosos.

3.7 Sistema FLOPER

Este sistema fue implementado por Pedro J. Morcillo y Ginés Moreno Universidad Castilla de la Mancha en 2008 [9], capaz de traducir directamente los programas de la lógica borrosa al código de Prolog, de tal manera que se pueda ejecutar en forma segura dentro de cualquier Prolog implementado con una representación de bajo nivel

de depuración, además de tener la capacidad de abrir en tiempo diseño, lo que permite que las tareas sean más sofisticadas para la manipulación de los programas.

Las cláusulas Prolog han sido extendidas con características borrosas, incluyendo un amplio repertorio de conectivas para manipular grados de verdad, en el que se obtiene más que un falso o verdadero. Su implementación, está basada en la línea de investigación que siguieron Voftas, et al. [37]. Establecen que un programa borroso es concebido como un conjunto de fórmulas de ponderación, donde el grado de verdad de cada cláusula es explícitamente anotada, debido a esto, la propagación de los grados de verdad es basada en una extensión del principio de resolución, mientras que el mecanismo de unificación (sintáctico) permanece intacto.

En Melvin Fitting [8], utilizan biretículos, considerado como el avance más moderno en el marco de la programación multi-adjunta [37] donde se involucra la programación lógica y la lógica borrosa, incluye un programa que puede ser visualizado como un conjunto de reglas cada uno de ellos, anotados con un grado de verdad. El objetivo es realizar una consulta al sistema, es decir, que un conjunto de átomos estén ligados con conectores denominados agregadores.

4. Conclusiones

Se puede mencionar que, aunque el trabajo de Lee [21] fue el pionero en este tema, le dio una relativa relevancia a las investigaciones sobre la programación lógica borrosa y que ésta aún no ha alcanzado todavía una fase de solidez y uniformidad comparable a la de Prolog clásico. La existencia de diferentes tipos de variantes difusas y, en consecuencia, de diferentes lógicas para representar la incertidumbre, hace que el diseño de interpretadores y compiladores en Prolog borrosos sean un poco complejos.

Los sistemas de Prolog borrosos tienen inmediata aplicación en diferentes áreas, como en bases de datos relacionales y deductivos, las cuales contienen información incierta. Además, tienen una aplicación directa en los sistemas expertos y de control que utilizan un lenguaje de atributos, objetos y valores.

En base al estudio realizado en este tema, es posible concluir que existen pocas investigaciones sobre implementaciones de Prolog borrosos, tanto de instituciones

públicas como empresas de software interesadas en el diseño y comercialización de un software de este tipo; sin embargo, en este trabajo se mencionan las aplicaciones que a la fecha están en uso y a disposición de la comunidad científica.

5. Bibliografía

[1] Baldwin, J. F., Martin, T.P. Pilsworth, B.W., "The implantation of FPROLOG: A fuzzy Prolog interpreter", Fuzzy sets and Systems 23, pages 119-129, 1987.

[2] Ciao, Disponible en http://www.ciaohome.org.

[3] Colmerauer, A., "The Birth of Prolog. In the Second ACM-SIGPLAN History of Programming Lan-guages Conference," ACM SIGPLAN Not., pp. 37-52, 1993.

[4] Dubois, D., Prade, H. Possibilistic logic: a retrospective and prospective view. Fuzzy Sets and Systems, pages 3-23, 2004.

[5] Dubois, D., Prade, H., Possibilistic Theory, New York: Plenum, 1988.

[6] Dubois, D. "Forty years of fuzzy sets., Fuzzy sets and System", Vol.

i. 156, 2005.

[7] Dubois, D., Prade, H. Fuzzy sets and probability: misundestadings, bridges and gaps, In: Proceedings of the 2nd IEEE international Conference on Fuzzy Systems (FUZZ-IEEE93), San Francisco, CA, 1993.

[8] Fitting, M. "Bilattices and the semantics of logic programming". Journal of Logic Programming, pages 91-116, 1991.

[9] FLOPER: A Fuzzy Logic Programming Environment for Research. http://www.dsi.uclm.es/investigacion/dect/FLOPERpag e.htm, 2010.

[10] GNU-PROLOG. Disponible en http://www.gprolog.org

[11] Julián, P. and Rubio C. "A declarative semantics for Bousi_Prolog". In PPDP'09: Proceedings of the 11th ACM SIGPLAN, Conference on Principlesand practice of declarative programming, pages 149-160, Coimbra, Portugal, September7-9 2009. ACM.

[12] Guadarrama, S., Muñoz, S. and Vaucheret, C. "Fuzzy PROLOG: A new approach using softconstraint propagation. Fuzzy Sets and Systems", pages 127-150, 2004.

[13] Julián, P., Rubio, C. and Gallardo J. "Bousi_Prolog: a Prolog exten-sion language for flexible query answering". In J. M. Almendros-Jiménez, editor, Proceedings of the Eigh the Spanish Conference on Programming and Computer Languages (PROLE 2008), volume 248of ENTCS, pages 131-147, Gijón, Spain, October 7-10 2009. Elsevier.

[14] Julián,Pascual I. and Rubio, Clemente. "A similarity-based WAM for Bousi_PROLOG. In Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN2009)", volume 5517 of LNCS, pages 245-252, Salamanca, Spain, 10-12 June 2009. SpringerBerlin / Heidelberg.

[15] Ishizuka, M. and Kanai N.," PROLOG-ELF incorporating fuzzy logic". In Aravind K. Joshi, editor, Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85), pages 701-703, Los Angeles, CA, USA, August 1985. Morgan Kaufmann. Formato, F., Gerla G., and Sessa, M. I. "Similarity-based unification. Fundamental Informatic", 40:1-22, 2000.

[16] Kentel, E. Aral, M.M., 2D Monte Carlo versus 2D fuzzy Monte Carlo health risk assessment, Stochastic Environmental Research on Risk Assessment, Vol. 19, 2005.

[17] Klir, G. J. and Folger, T. A.. Fuzzy sets, Uncertainty, and Information. Prentice-Hall International, NY, 1988.

[18] Klir, G. Yuan, B. Fuzzy sets and fuzzy Logic, Theory and applications ,Prentice Hall PTR ed), New Jersey, 1995.

[19] Kowalski, R. and Kuehner, D. "Resolution with selection function. Artificial Intelligence", pages 227-260, 1970.

[20] Kreinovich, V. Nguyen, H.T., "Which fuzzy logic is the best: Pragmatic approach(and its theorical analysis), Fuzzy Sets and Systems", Vol. 157, No. 5, 2006. Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85), pages 701-703,

[21] Lee, R.T.C., Fuzzy logic and the resolution Principle, Journal of the Association for Computing Machinery, 1972.Los Angeles, CA, USA, August 1985. Morgan Kaufmann.

[22] Li, D. and Liu, D. A Fuzzy Prolog Database System. John Wiley & Sons, 1990.

[23] Lloyd, J.W. Foundations of Logic Programming, Springer-Verlag,1987. Fuzzy Sets and Systems", pages 162-181, 2009.

[24] Mares, M. How to handle fuzzy quantities?, Kibernetika, Vol. 13, No. 1, 1977.

[25] Medina, J., Ojeda M. and Vojtáš, P. "A procedural semantics for multi-adjoint logic programming". In P. Brazdil and A. Jorge, editors, Progress in Artificial Intelligence (EPIA'01),volume 2258 of LNAI, pages 290-297. Springer-Verlag, 2001.

[26] Medina, J., Ojeda M., and Vojtás, P. "Multi-adjoint logic programming with continuous semantics". In T. Eiter, W. Faber, and M. Ruszczinski, editors, Logic Programming and Non Monotonic Reasoning (LPNMR'01), volume 2173 of LNAI, pages 351-354. Springer-Verlag, 2001.

[27] Nuel D. Belnap, Jr. "A useful four-valued logic". In J. Michael Dunn and G. Epstein, editors, Modern Uses of Multiple Valued Logic, pages 8-37. D. Reidel Pub. Co., 1977.

[28] Pyton, S. J. The haskell 98 languaje. Journal of functional Programming, 13(1):7-255, January 2003.

[29] Resher, N. Plausible Reasoning . Van Gorcum, Amsterdam, 1976.

[30] Schweizer, B. and A. Sklar. Probabilistic metric spaces. North Holland, Amsterdam, 1983.

[31] Sessa, M. I. "Approximate reasoning by similarity-based SLD resolution". Theoretical ComputerScience, pages 389-426, 2002.

[32] Shenoi, S. and Melton, A. "Proximity relations in the fuzzy relational database model". Fuzzy Sets and Systems, pages 51-62, 1999.

[33] SICS AB. Sictus PROLOG. Disponible en http://www.sictus.se/sictus

[34] SWI-PROLOG, disponible en http://www.swi-PROLOG.org.

[35] Umano, M. "Fuzzy Sets Prolog", preprints of the 2nd. Inter. Fuzzy Systems Assoc. (IFSA) Congress, Tokio, Japan, pages 750-753, 1987.

[36] Vaucheret, C., Guadarrama, S., and Muñoz, S. "Fuzzy PROLOG: A simple general implementation using CLP(R)". In M. Baaz and A. Voronkov, editors, Logic for Programming, Artificial Intelligence,and Reasoning (LPAR'02), volume

2514 of LNCS, pages 450-463, Tbilisi, Georgia, October 14-18 2002. Springer Berlin/Heidelberg .

[37] Vojtáš, P. "Fuzzy logic programming". Fuzzy Sets and Systems, pages 361-370, 2001.

[38] Zadeh, L.A. Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, 1978.

[39] Zadeh, L.A. "Fuzzy sets. Information and Control", pages 333-353, 1965.

[40] Zadeh, L.A. "Similarity relations and fuzzy orderings". Information Sciences, pages 177-200, 1971.

[41] Zadeh, L.A., Calculus of Fuzzy restrictions. in: L.A. Zadeh, KS. Fu, Tanaka and M. Shimura, eds. Fuzzy Sets and Their Applications to Congnitive and Decision Processes, Academic Press, New York, 1975.